

**ĐẠI HỌC HUẾ**  
**TRƯỜNG ĐẠI HỌC KHOA HỌC**  
**KHOA CÔNG NGHỆ THÔNG TIN**  
∞  ∞

**GIÁO TRÌNH**  
**LÝ THUYẾT HỆ ĐIỀU HÀNH**

**BIÊN SOẠN: NGUYỄN KIM TUẤN**

**Huế 06/2004**

# MỤC LỤC

---

Trang

## Chương I: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

<b>I.1. Chức năng và lịch sử phát triển của hệ điều hành .....</b>	<b>1</b>
I.1.1. Chức năng của hệ điều hành .....	1
I.1.2. Lịch sử phát triển của hệ điều hành .....	3
<b>I.2. Một số khái niệm của hệ điều hành .....</b>	<b>5</b>
I.2.1. Tiến trình (Process) và tiểu trình (Thread) .....	5
I.2.2. Bộ xử lý lệnh (Shell) .....	5
I.2.3. Sự phân lớp hệ thống (System Layering) .....	6
I.2.4. Tài nguyên hệ thống (System Resources) .....	7
I.2.5. Lời gọi hệ thống (System Calls) .....	7
<b>I.3. Hệ điều hành và phân loại hệ điều hành .....</b>	<b>8</b>
I.3.1. Hệ điều hành là gì? .....	8
I.3.2. Phân loại hệ điều hành .....	9
<b>I.4. Thành phần và cấu trúc của hệ điều hành .....</b>	<b>12</b>
I.4.1. Các thành phần của hệ điều hành .....	12
I.4.2. Các cấu trúc của hệ điều hành .....	16
<b>I.5. Hệ điều hành Windows95 .....</b>	<b>21</b>
I.5.1. Giới thiệu về hệ điều hành Windows95 .....	22
I.5.2. Cấu trúc của windows95 .....	24
I.5.3. Bộ nhớ ảo trong windows95 .....	25
<b>I.6. Hệ điều hành Windows 2000 .....</b>	<b>26</b>
I.6.1. Giới thiệu về hệ điều hành Windows 2000 .....	26
I.6.2. Một số đặc tính của Windows 2000 .....	27
I.6.3. Một số khái niệm trong Windows 2000 .....	28
I.6.4. Kiến trúc của Windows 2000 .....	31
<b>I.7. Hệ điều hành Linux .....</b>	<b>37</b>

## Chương II: QUẢN LÝ TIẾN TRÌNH

<b>II.1. Tổng quan về tiến trình .....</b>	<b>41</b>
I.1.1. Tiến trình và các loại tiến trình .....	41

<b>I.1.2.</b> Mô hình tiến trình .....	42
<b>I.1.3.</b> Tiểu trình và tiến trình .....	45
<b>I.1.4.</b> Các trạng thái tiến trình .....	46
<b>I.1.5.</b> Cấu trúc dữ liệu của khối quản lý tiến trình .....	50
<b>I.1.6.</b> Các thao tác điều khiển tiến trình .....	52
<b>II.2.</b> Tài nguyên căng và đoạn căng .....	53
<b>II.2.1.</b> Tài nguyên căng (Critical Resource) .....	53
<b>II.2.2.</b> Đoạn căng (Critical Section) .....	57
<b>II.2.3.</b> Yêu cầu của công tác điều độ qua đoạn căng .....	59
<b>II.3.</b> Điều độ tiến trình qua đoạn căng .....	60
<b>II.3.1.</b> ..... Các	
giải pháp phần cứng .....	60
<b>II.3.2.</b> ..... Các	
giải pháp dùng biến khoá .....	62
<b>II.3.3.</b> ..... Các	
giải pháp được hỗ trợ bởi hệ điều hành và ngôn ngữ lập trình.....	63
<b>II.3.4.</b> ..... Hai	
bài toán điều phối làm ví dụ .....	72
<b>II.4.</b> Tắc nghẽn (Deadlock) và chống tắc nghẽn .....	79
<b>II.4.1.</b> ..... T	
ắc nghẽn .....	79
<b>II.4.2.</b> Điều kiện hình thành tắc nghẽn .....	81
<b>II.4.3.</b> Ngăn chặn tắc nghẽn (Deadlock Prevention) .....	81
<b>II.4.4.</b> Nhận biết tắc nghẽn (Deadlock Detection) .....	81
<b>II.5.</b> Điều phối tiến trình	
<b>II.5.1.</b> Mục tiêu điều phối .....	83
<b>II.5.2.</b> Tổ chức điều phối .....	86
<b>II.5.3.</b> Các chiến lược điều phối .....	87
<b>II.6.</b> Tiến trình trong Windows NT .....	89

### **Chương III: QUẢN LÝ BỘ NHỚ**

<b>III.1.</b> Nhiệm vụ của quản lý bộ nhớ .....	93
<b>III.2.</b> Kỹ thuật cấp phát bộ nhớ ( nạp chương trình vào bộ nhớ chính) 95	
III.2.1. Kỹ thuật phân vùng cố định (Fixed Partitioning) .....	95
III.2.2. Kỹ thuật phân vùng động (Dynamic Partitioning) .....	97

III.2.3.Kỹ thuật phân trang đơn (Simple Paging) .....	103
III.2.4. Kỹ thuật phân đoạn đơn (Simple Segmentation).....	106
<b>III.3. Kỹ thuật bộ nhớ ảo (Virtual Memory).....</b>	<b>109</b>
<b>III.3.1.</b> Bộ nhớ ảo .....	109
<b>III.3.2.</b> Kỹ thuật bộ nhớ ảo .....	112
<b>III.4. Quản lý bộ nhớ RAM của DOS .....</b>	<b>126</b>
<b>III.5.a.</b> .....	Program Segment Prefix (PSP) .....
<b>III.5.b.</b> .....	Chương trình COM và EXE .....
<b>III.5.c.</b> .....	Memory Control Block (MCB) .....
<b>III.5.Sự phân trang/đoạn trong hệ điều hành Windows NT .....</b>	<b>130</b>
<b>III.5.a.</b> Segmentation .....	130
<b>III.5.b.</b> Paging .....	132
<b>III.6. Các thuật toán thay trang .....</b>	<b>133</b>
<b>III.7. Cấp phát khung trang .....</b>	<b>136</b>
<b>III.8. Một số vấn đề về quản lý bộ nhớ của Windows 2000 .....</b>	<b>137</b>
III.8.1. Nhiệm vụ quản lý bộ nhớ của Windows 2000 .....	137
III.8.2. Các dịch vụ trình quản lý bộ nhớ cung cấp .....	138
III.8.3. Address Space Layout .....	141
III.8.4. Chuyển đổi địa chỉ .....	142

## **Chương IV: QUẢN LÝ FILE và ĐĨA**

<b>IV.1. Tổng quan về quản lý tập tin và đĩa.....</b>	<b>148</b>
1. Tập tin và hệ thống quản lý tập tin.....	148
2. Bảng danh mục và tập tin chia sẻ.....	151
3. Quản lý không gian đĩa .....	153
4. Quản lý các block chứa file trên đĩa.....	155
5. An toàn trong quản lý tập tin.....	158
6. Hiệu suất hệ thống file .....	162
<b>IV.2. Các điều khiển hệ thống tập tin.....</b>	<b>164</b>
<b>IV.3. Các hệ thống file trên các hệ điều hành hiện nay .....</b>	<b>166</b>
<b>IV.4. Tổ chức đĩa của MS_DOS .....</b>	<b>167</b>
<b>IV.5. Quản lý file trên đĩa của MS_DOS.....</b>	<b>172</b>

<b>IV.6. Tổ chức bảng thư mục gốc của Windows98 .....</b>	<b>185</b>
<b>IV.7. Tổ chức đĩa của windows 2000 .....</b>	<b>188</b>
IV.7.1.Các loại partition .....	188
IV.7.2.Các loại volume multipartition.....	192
<b>IV.8. Quản lý lưu trữ file trên đĩa của windowsNT/2000 .....</b>	<b>195</b>
IV.8.1. Một số chức năng được hỗ trợ bởi NTFS của windows 2000.	195
IV.8.2. Cấu trúc của MFT .....	196
IV.8.3. Quản lý danh sách các block chứa file trên đĩa .....	203
<b>IV.9. Một số kỹ thuật được hỗ trợ bởi hệ thống file NTFS .....</b>	<b>206</b>
IV.9.1.Lập bảng chỉ mục.....	206
IV.9.2.Ánh xạ Bad-cluster .....	207
<b>IV.10. Tổ chức lưu trữ file trên đĩa CD_ROM.....</b>	<b>209</b>
<b>Mục lục.....</b>	<b>212</b>
<b>Tài liệu tham khảo .....</b>	<b>215</b>

## Chương I

# TỔNG QUAN VỀ HỆ ĐIỀU HÀNH

---

*Nếu không có phần mềm, máy tính chỉ là một thiết bị điện tử thông thường. Với sự hỗ trợ của phần mềm, máy tính có thể lưu trữ, xử lý thông tin và người sử dụng có thể gọi lại được thông tin này. Phần mềm máy tính có thể chia thành nhiều loại: chương trình hệ thống, quản lý sự hoạt động của chính máy tính. Chương trình ứng dụng, giải quyết các vấn đề liên quan đến việc sử dụng và khai thác máy tính của người sử dụng. Hệ điều hành thuộc nhóm các chương trình hệ thống và nó là một chương trình hệ thống quan trọng nhất đối với máy tính và cả người sử dụng. Hệ điều hành điều khiển tất cả các tài nguyên của máy tính và cung cấp một môi trường thuận lợi để các chương trình ứng dụng do người sử dụng viết ra có thể chạy được trên máy tính. Trong chương này chúng ta xem xét vai trò của hệ điều hành trong trường hợp này.*

*Một máy tính hiện đại có thể bao gồm: một hoặc nhiều processor, bộ nhớ chính, clocks, đĩa, giao diện mạng, và các thiết bị vào/ra khác. Tất cả nó tạo thành một hệ thống phức tạp. Để viết các chương trình để theo dõi tất cả các thành phần của máy tính và sử dụng chúng một cách hiệu quả, người lập trình phải biết processor thực hiện chương trình như thế nào, bộ nhớ lưu trữ thông tin như thế nào, các thiết bị đĩa làm việc (ghi/đọc) như thế nào, lỗi nào có thể xảy ra khi đọc một block đĩa, ... đây là những công việc rất khó khăn và quá khó đối với người lập trình. Nhưng rất may cho cả người lập trình ứng dụng và người sử dụng là những công việc trên đã được hệ điều hành hỗ trợ nên họ không cần quan tâm đến nữa. Chương này cho chúng ta một cái nhìn tổng quan về những gì liên quan đến việc thiết kế cài đặt cũng như chức năng của hệ điều hành để hệ điều hành đạt được mục tiêu: Giúp người sử dụng khai thác máy tính dễ dàng và chương trình của người sử dụng có thể chạy được trên máy tính.*

## **I.8. Chức năng và lịch sử phát triển của hệ điều hành**

### **I.1.7. Chức năng của hệ điều hành**

Một hệ thống máy tính gồm 3 thành phần chính: phần cứng, hệ điều hành và các chương trình ứng dụng và người sử dụng. Trong đó hệ điều hành là một bộ phận quan trọng và không thể thiếu của hệ thống máy tính, nhờ có hệ điều hành mà

người sử dụng có thể đối thoại và khai thác được các chức năng của phần cứng máy tính.

Có thể nói hệ điều hành là một hệ thống các chương trình đóng vai trò trung gian giữa người sử dụng và phần cứng máy tính. Mục tiêu chính của nó là cung cấp một môi trường thuận lợi để người sử dụng dễ dàng thực hiện các chương trình ứng dụng của họ trên máy tính và khai thác triệt để các chức năng của phần cứng máy tính.

Để đạt được mục tiêu trên hệ điều hành phải thực hiện 2 chức năng chính sau đây:

- **Giả lập một máy tính mở rộng:** Máy tính là một thiết bị vi điện tử, nó được cấu thành từ các bộ phận như: Processor, Memory, I/O Device, Bus, ... , do đó để đối thoại hoặc khai thác máy tính người sử dụng phải hiểu được cơ chế hoạt động của các bộ phận này và phải tác động trực tiếp vào nó, tất nhiên là bằng những con số 0,1 (ngôn ngữ máy). Điều này là quá khó đối với người sử dụng. Để đơn giản cho người sử dụng hệ điều hành phải che đậy các chi tiết phần cứng máy tính bởi một máy tính mở rộng, máy tính mở rộng này có đầy đủ các chức năng của một máy tính thực nhưng đơn giản và dễ sử dụng hơn. Theo đó khi cần tác động vào máy tính thực người sử dụng chỉ cần tác động vào máy tính mở rộng, mọi sự chuyển đổi thông tin điều khiển từ máy tính mở rộng sang máy tính thực hoặc ngược lại đều do hệ điều hành thực hiện. Mục đích của chức năng này là: *Giúp người sử dụng khai thác các chức năng của phần cứng máy tính dễ dàng và hiệu quả hơn.*

- **Quản lý tài nguyên của hệ thống:** Tài nguyên hệ thống có thể là: processor, memory, I/O device, printer, file, ..., đây là những tài nguyên mà hệ điều hành dùng để cấp phát cho các tiến trình, chương trình trong quá trình điều khiển sự hoạt động của hệ thống. Khi người sử dụng cần thực hiện một chương trình hay khi một chương trình cần nạp thêm một tiến trình mới vào bộ nhớ thì hệ điều hành phải cấp phát không gian nhớ cho chương trình, tiến trình đó để chương trình, tiến trình đó nạp được vào bộ nhớ và hoạt động được. Trong môi trường hệ điều hành đa nhiệm có thể có nhiều chương trình, tiến trình đồng thời cần được nạp vào bộ nhớ, nhưng không gian lưu trữ của bộ nhớ có giới hạn, do đó hệ điều hành phải tổ chức cấp phát bộ nhớ sao cho hợp lý để đảm bảo tất cả các chương trình, tiến trình khi cần đều được nạp vào bộ nhớ để hoạt động. Ngoài ra hệ điều hành còn phải tổ chức bảo vệ các không gian nhớ đã cấp cho các chương trình, tiến trình để tránh sự truy cập bất hợp lệ và sự tranh chấp bộ nhớ giữa các chương trình, tiến trình, đặc biệt là các tiến trình đồng thời hoạt động trên hệ thống. Đây là một trong những nhiệm vụ quan trọng của hệ điều hành.

Trong quá trình hoạt động của hệ thống, đặc biệt là các hệ thống đa người dùng, đa chương trình, đa tiến trình, còn xuất hiện một hiện tượng khác, đó là nhiều

chương trình, tiến trình đồng thời sử dụng một không gian nhớ hay một tập tin (dữ liệu, chương trình) nào đó. Trong trường hợp này hệ điều hành phải tổ chức việc chia sẻ và giám sát việc truy xuất đồng thời trên các tài nguyên nói trên sao cho việc sử dụng tài nguyên có hiệu quả nhưng tránh được sự mất mát dữ liệu và làm hỏng các tập tin.

Trên đây là hai dẫn chứng điển hình để chúng ta thấy vai trò của hệ điều hành trong việc quản lý tài nguyên hệ thống, sau này chúng ta sẽ thấy việc cấp phát, chia sẻ, bảo vệ tài nguyên của hệ điều hành là một trong những công việc khó khăn và phức tạp nhất. Hệ điều hành đã chi phí nhiều cho công việc nói trên để đạt được mục tiêu: *Trong mọi trường hợp tất cả các chương trình, tiến trình nếu cần được cấp phát tài nguyên để hoạt động thì sớm hay muộn nó đều được cấp phát và được đưa vào trạng thái hoạt động.*

➤ Trên đây là hai chức năng tổng quát của một hệ điều hành, đó cũng được xem như là các mục tiêu mà các nhà thiết kế, cài đặt hệ điều hành phải hướng tới. Các hệ điều hành hiện nay có các chức năng cụ thể sau đây:

- Hệ điều hành cho phép thực hiện nhiều chương trình đồng thời trong môi trường đa tác vụ - **Multitasking Environment**. Hệ điều hành multitasking bao gồm: Windows NT, Windows 2000, Linux và OS/2. Trong hệ thống multitasking hệ điều hành phải xác định khi nào thì một ứng dụng được chạy và mỗi ứng dụng được chạy trong khoảng thời gian bao lâu thì phải dừng lại để cho các ứng dụng khác được chạy.

- Hệ điều hành tự nạp nó vào bộ nhớ - **It loads itself into memory**: Quá trình nạp hệ điều hành vào bộ nhớ được gọi là quá trình **Bootting**. Chỉ khi nào hệ điều hành đã được nạp vào bộ nhớ thì nó mới cho phép người sử dụng giao tiếp với phần cứng. Trong các hệ thống có nhiều ứng dụng đồng thời hoạt động trên bộ nhớ thì hệ điều hành phải chịu trách nhiệm chia sẻ không gian bộ nhớ RAM và bộ nhớ cache cho các ứng dụng này.

- **Hệ điều hành và API: Application Programming Interface**: API là một tập các hàm/thủ tục được xây dựng sẵn bên trong hệ thống, nó có thể thực hiện được nhiều chức năng khác nhau như shutdown hệ thống, đảo ngược hiệu ứng màn hình, khởi động các ứng dụng, ... Hệ điều hành giúp cho chương trình của người sử dụng giao tiếp với API hay thực hiện một lời gọi đến các hàm/thủ tục của API.

- Nạp dữ liệu cần thiết vào bộ nhớ - **It loads the required data into memory**: Dữ liệu do người sử dụng cung cấp được đưa vào bộ nhớ để xử lý. Khi nạp dữ liệu vào bộ nhớ hệ điều hành phải lưu lại địa chỉ của bộ nhớ nơi mà dữ liệu được lưu ở đó. Hệ điều hành phải luôn theo dõi bản đồ cấp phát bộ nhớ, nơi dữ liệu và chương trình được lưu trữ ở đó. Khi một chương trình cần đọc dữ liệu, hệ điều hành sẽ đến các địa chỉ bộ nhớ nơi đang lưu trữ dữ liệu mà chương trình cần đọc để đọc lại nó.



□ Hệ điều hành biên dịch các chỉ thị chương trình - **It interprets program instructions**: Hệ điều hành phải đọc và giải mã các thao tác cần được thực hiện, nó được viết trong chương trình của người sử dụng. Hệ điều hành cũng chịu trách nhiệm sinh ra thông báo lỗi khi hệ thống gặp lỗi trong khi đang hoạt động.

□ Hệ điều hành quản lý tài nguyên - **It manages resources**: Nó đảm bảo việc sử dụng thích hợp tất cả các tài nguyên của hệ thống như là: bộ nhớ, đĩa cứng, máy in, ...

## **I.1.8. Lịch sử phát triển của hệ điều hành**

### **I.1.2.a. Thế hệ 1 (1945 - 1955):**

Vào những năm 1950 máy tính dùng ống chân không ra đời. Ở thế hệ này mỗi máy tính được một nhóm người thực hiện, bao gồm việc thiết kế, xây dựng chương trình, thao tác, quản lý, ....

Ở thế hệ này người lập trình phải dùng ngôn ngữ máy tuyệt đối để lập trình. Khái niệm ngôn ngữ lập trình và hệ điều hành chưa được biết đến trong khoảng thời gian này.

### **I.1.2.b. Thế hệ 2 (1955 - 1965):**

Máy tính dùng bán dẫn ra đời, và được sản xuất để cung cấp cho khách hàng. Bộ phận sử dụng máy tính được phân chia rõ ràng: người thiết kế, người xây dựng, người vận hành, người lập trình, và người bảo trì. Ngôn ngữ lập trình Assembly và Fortran ra đời trong thời kỳ này. Với các máy tính thế hệ này để thực hiện một thao tác, lập trình viên dùng Assembly hoặc Fortran để viết chương trình trên phiếu đục lỗ sau đó đưa phiếu vào máy, máy thực hiện cho kết quả ở máy in.

Hệ thống xử lý theo lô cũng ra đời trong thời kỳ này. Theo đó, các thao tác cần thực hiện trên máy tính được ghi trước trên băng từ, hệ thống sẽ đọc băng từ, thực hiện lần lượt và cho kết quả ở băng từ xuất. Hệ thống xử lý theo lô hoạt động dưới sự điều khiển của một chương trình đặc biệt, chương trình này là hệ điều hành sau này.

### **I.1.2.c. Thế hệ 3 (1965 - 1980)**

Máy IBM 360 được sản xuất hàng loạt để tung ra thị trường. Các thiết bị ngoại vi xuất hiện ngày càng nhiều, do đó các thao tác điều khiển máy tính và thiết bị ngoại vi ngày càng phức tạp hơn. Trước tình hình này nhu cầu cần có một hệ điều hành sử dụng chung trên tất cả các máy tính của nhà sản xuất và người sử dụng trở nên bức thiết hơn. Và hệ điều hành đã ra đời trong thời kỳ này.

Hệ điều hành ra đời nhằm điều phối, kiểm soát hoạt động của hệ thống và giải quyết các yêu cầu tranh chấp thiết bị. Hệ điều hành đầu tiên được viết bằng ngôn ngữ Assembly. Hệ điều hành xuất hiện khái niệm đa chương, khái niệm chia

sẽ thời gian và kỹ thuật Spool. Trong giai đoạn này cũng xuất hiện các hệ điều hành Multics và Unix.

#### **I.1.2.d. Thế hệ 4 (từ 1980)**

Máy tính cá nhân ra đời. Hệ điều hành MS\_DOS ra đời gắn liền với máy tính IBM\_PC. Hệ điều hành mạng và hệ điều hành phân tán ra đời trong thời kỳ này.

➤ Trên đây chúng tôi không có ý định trình bày chi tiết, đầy đủ về lịch sử hình thành của hệ điều hành, mà chúng tôi chỉ muốn mượn các mốc thời gian về sự ra đời của các thế hệ máy tính để chỉ cho bạn thấy quá trình hình thành của hệ điều hành gắn liền với quá trình hình thành máy tính. Mục tiêu của chúng tôi trong mục này là muốn nhấn mạnh với các bạn mấy điểm sau đây:

- Các ngôn ngữ lập trình, đặc biệt là các ngôn ngữ lập trình cấp thấp, ra đời trước các hệ điều hành. Đa số các hệ điều hành đều được xây dựng từ ngôn ngữ lập trình cấp thấp trừ hệ điều hành Unix, nó được xây dựng từ C, một ngôn ngữ lập trình cấp cao.
- Nếu không có hệ điều hành thì việc khai thác và sử dụng máy tính sẽ khó khăn và phức tạp rất nhiều và không phải bất kỳ ai cũng có thể sử dụng máy tính được.
- Sự ra đời và phát triển của hệ điều hành gắn liền với sự phát triển của máy tính, và ngược lại sự phát triển của máy tính kéo theo sự phát triển của hệ điều hành. Hệ điều hành thực sự phát triển khi máy tính PC xuất hiện trên thị trường.
- Ngoài ra chúng tôi cũng muốn giới thiệu một số khái niệm như: hệ thống xử lý theo lô, hệ thống đa chương, hệ thống chia sẻ thời gian, kỹ thuật Spool, ..., mà sự xuất hiện của những khái niệm này đánh dấu một bước phát triển mới của hệ điều hành. Chúng ta sẽ làm rõ các khái niệm trên trong các chương sau của tài liệu này.

### **I.9. Một số khái niệm của hệ điều hành**

#### **I.2.6. Tiến trình (Process) và tiểu trình (Thread)**

Tiến trình là một bộ phận của chương trình đang thực hiện. Tiến trình là đơn vị làm việc cơ bản của hệ thống, trong hệ thống có thể tồn tại nhiều tiến trình cùng hoạt động, trong đó có cả tiến trình của hệ điều hành và tiến trình của chương trình người sử dụng. Các tiến trình này có thể hoạt động đồng thời với nhau.

Để một tiến trình đi vào trạng thái hoạt động thì hệ thống phải cung cấp đầy đủ tài nguyên cho tiến trình. Hệ thống cũng phải duy trì đủ tài nguyên cho tiến trình trong suốt quá trình hoạt động của tiến trình.

Ở đây cần phân biệt sự khác nhau giữa tiến trình và chương trình, chương trình là một tập tin thụ động nằm trên đĩa, tiến trình là trạng thái động của chương

trình.

Các hệ điều hành hiện đại sử dụng mô hình đa tiểu trình, trong một tiến trình có thể có nhiều tiểu trình. Tiểu trình cũng là đơn vị xử lý cơ bản trong hệ thống, nó cũng xử lý tuần tự đoạn code của nó, nó cũng sở hữu một con trỏ lệnh, một tập các thanh ghi và một vùng nhớ stack riêng và các tiểu trình cũng chia sẻ thời gian xử lý của processor như các tiến trình.

Các tiểu trình trong một tiến trình chia sẻ một không gian địa chỉ chung, điều này có nghĩa các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình, có thể truy xuất đến stack của tiểu trình khác trong cùng tiến trình. Như vậy với mô hình tiểu trình, trong hệ thống có thể tồn tại nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ bộ nhớ, các dòng xử lý này hoạt động song song với nhau.

### **I.2.7. Bộ xử lý lệnh (Shell)**

Shell là một bộ phận hay một tiến trình đặc biệt của hệ điều hành, nó có nhiệm vụ nhận lệnh của người sử dụng, phân tích lệnh và phát sinh tiến trình mới để thực hiện yêu cầu của lệnh, tiến trình mới này được gọi là tiến trình đáp ứng yêu cầu.

Shell nhận lệnh thông qua cơ chế dòng lệnh, đó chính là nơi giao tiếp giữa người sử dụng và hệ điều hành, mỗi hệ điều hành khác nhau có cơ chế dòng lệnh khác nhau, với MS\_DOS đó là con trỏ lệnh và dấu nhắc hệ điều hành (C:\>\_), với Windows 9x đó là nút Start\Run. Tập tin Command.Com chính là Shell của MS\_DOS.

Trong môi trường hệ điều hành đơn nhiệm, ví dụ như MS\_DOS, khi tiến trình đáp ứng yêu cầu hoạt động thì Shell sẽ chuyển sang trạng thái chờ, để chờ cho đến khi tiến trình đáp ứng yêu cầu kết thúc thì Shell trở lại trạng thái sẵn sàng nhận lệnh mới.

Trong môi trường hệ điều hành đa nhiệm, ví dụ như Windows 9x, sau khi phát sinh tiến trình đáp ứng yêu cầu và đưa nó vào trạng thái hoạt động thì Shell sẽ chuyển sang trạng thái sẵn sàng nhận lệnh mới, nhờ vậy Shell có khả năng khởi tạo nhiều tiến trình đáp ứng yêu cầu để nó hoạt động song song với nhau, hay chính xác hơn trong môi trường hệ điều hành đa nhiệm người sử dụng có thể khởi tạo nhiều chương trình để nó hoạt động đồng thời với nhau.

➤ **Chú ý:** Hầu hết các ngôn ngữ lập trình đều hỗ trợ các công cụ để người sử dụng hay người lập trình có thể gọi shell ngay trong các ứng dụng của họ. Khi một ứng dụng cần gọi thực hiện một chương trình nào đó thì:

- Trong Assembly, các ứng dụng gọi hàm 4Bh/21h của MS\_DOS.
- Trong Pascal, các ứng dụng gọi thủ tục Exec.
- Trong Visual Basic, các ứng dụng gọi hàm/ thủ tục Shell. Ví dụ dòng lệnh sau: Shell "C:\Windows\notepad.exe" có thể gọi thực

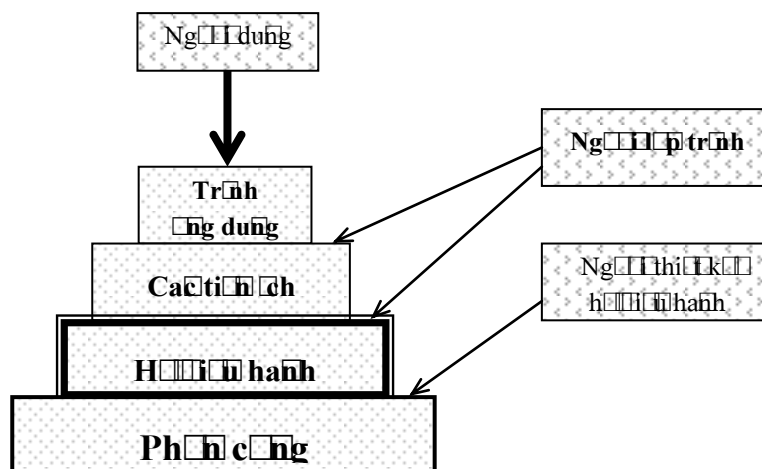
hiện chương trình Notepad của Windows.

- Trong Windows 9x/ Windows NT, các ứng dụng gọi hàm ShellExecute.

### I.2.8. Sự phân lớp hệ thống (System Layering)

Như đã biết, hệ điều hành là một hệ thống các chương trình bao quanh máy tính thực (vật lý) nhằm tạo ra một máy tính mở rộng (logic) đơn giản và dễ sử dụng hơn. Theo đó, khi khai thác máy tính người sử dụng chỉ cần tác động vào lớp vỏ bọc bên ngoài của máy tính, mọi sự giao tiếp giữa lớp vỏ bọc này với các chi tiết phần cứng bên trong đều do hệ điều hành thực hiện.

Mỗi người sử dụng khác nhau yêu cầu khai thác hệ điều hành ở những mức độ khác nhau. Người sử dụng thông thường chỉ cần một môi trường thuận lợi để họ thực hiện các ứng dụng, các lập trình viên cần có một môi trường lập trình tốt để họ có thể triển khai các ứng dụng, các chuyên viên lập trình hệ thống cần hệ điều hành cung cấp cho họ các công cụ để họ can thiệp sâu hơn vào hệ thống phần cứng máy tính, ... Để đáp ứng yêu cầu của nhiều đối tượng người sử dụng khác nhau hệ điều



Hình 1.1 Sự phân lớp hệ thống

hành thực hiện phân lớp các chương trình bao quanh máy tính. Các hệ thống như vậy được gọi là hệ thống phân lớp. Hình vẽ 1.1 ở trên minh họa cho một hệ thống phân lớp.

Ta có thể hình dung một hệ thống phân lớp được tổ chức như sau:

- Trong cùng là hệ điều hành.
- Tiếp theo là các ngôn ngữ lập trình
- ...
- Ngoài cùng là các chương trình ứng dụng .

Người sử dụng tác động vào lớp trong cùng sẽ gặp nhiều khó khăn hơn khi tác động vào lớp ngoài cùng.

### **I.2.9. Tài nguyên hệ thống (System Resources)**

Tài nguyên hệ thống là những tồn tại về mặt vật lý tại một thời điểm nhất định hoặc tại mọi thời điểm, và nó có khả năng tác động đến hiệu suất của hệ thống. Một cách tổng quát có thể chia tài nguyên của hệ thống thành hai loại cơ bản:

- Tài nguyên không gian: là các không gian lưu trữ của hệ thống như đĩa, bộ nhớ chính, quan trọng nhất là không gian bộ nhớ chính, nơi lưu trữ các chương trình đang được CPU thực hiện.
- Tài nguyên thời gian: chính là thời gian thực hiện lệnh của processor và thời gian truy xuất dữ liệu trên bộ nhớ.

#### ➤ **Sau đây là một vài tài nguyên hệ thống:**

❑ **Bộ nhớ:** Đặc trưng cơ bản của bộ nhớ là thời gian truy cập trực tiếp, thời gian truy cập tuần tự, và dung lượng nhớ. Bộ nhớ được gọi là thực hiện nếu processor có thể thực hiện một câu lệnh trong nó, loại bộ nhớ này có thời gian truy cập trực tiếp và tuần tự là như nhau. Bộ nhớ trong (RAM) của PC là bộ nhớ thực hiện và nó được quản lý bởi hệ thống.

Khi sử dụng bộ nhớ ta cần phân biệt 2 khái niệm: bộ nhớ và truy cập tới bộ nhớ. Bộ nhớ chỉ vùng vật lý chứa dữ liệu, truy cập bộ nhớ là quá trình tìm đến dữ liệu trên bộ nhớ. Có thể xem đây là 2 loại tài nguyên khác nhau vì chúng tồn tại độc lập với nhau.

❑ **Processor:** Là tài nguyên quan trọng nhất của hệ thống, nó được truy cập ở mức câu lệnh và chỉ có nó mới làm cho câu lệnh thực hiện hay chỉ có Processor mới đưa tiến trình vào trạng thái hoạt động. Trong thực tế khi xem xét về processor người ta chỉ chú ý đến thời gian xử lý của processor.

❑ **Tài nguyên ảo/ tài nguyên logic (Virtual Resources):** Là loại tài nguyên cung cấp cho chương trình người sử dụng dưới dạng đã được biến đổi, nó chỉ xuất hiện khi hệ thống cần tới nó hoặc khi hệ thống tạo ra nó và nó sẽ tự động mất đi khi hệ thống kết thúc hay chính xác hơn là khi tiến trình gắn với nó đã kết thúc. Tài nguyên ảo có thể là: Đĩa ảo trong môi trường MS\_DOS. Điều khiển in trong môi trường mạng của Windows 9x/NT. Nội dung thư mục Spool trong Windows 9x.

➤ Trên khía cạnh cấp phát tài nguyên cho các tiến trình đang hoạt động đồng thời thì tài nguyên hệ thống được chia thành 2 loại:

❑ **Tài nguyên phân chia được:** là những tài nguyên mà tại một thời điểm nó có thể cấp phát cho nhiều tiến trình khác nhau, các tiến trình song song có thể đồng thời sử dụng các tài nguyên này. Bộ nhớ chính và Processor là 2 tài

nguyên phân chia được điển hình nhất, bởi tại một thời điểm có thể có nhiều tiến trình cùng chia nhau sử dụng không gian lưu trữ của bộ nhớ chính và có thể có nhiều tiến trình thay nhau sử dụng thời gian xử lý của processor.

□ **Tài nguyên không phân chia được:** là những tài nguyên mà tại một thời điểm nó chỉ có thể cấp phát cho một tiến trình duy nhất. Máy in là một tài nguyên không phân chia được điển hình nhất.

Vấn đề đặt ra đối với hệ điều hành là phải biến các tài nguyên không phân chia được thành những tài nguyên phân chia được, theo một cách nào đó, để cấp phát cho các tiến trình khi nó có yêu cầu, đặc biệt là các tiến trình hoạt động đồng thời với nhau. Các hệ điều hành đa nhiệm đã cài đặt thành công mục tiêu này. Như chúng ta đã thấy trong môi trường Windows 9x/ NT có thể có nhiều tiến trình/ nhiều người sử dụng khác nhau đồng thời sử dụng một máy in.

Ngoài ra hệ điều hành còn phải giải quyết vấn đề tranh chấp tài nguyên giữa các tiến trình đồng thời khi yêu cầu phục vụ của các tiến trình này vượt quá khả năng cấp phát của một tài nguyên kể cả đó là tài nguyên phân chia được.

#### **I.2.10. Lời gọi hệ thống (System Calls)**

Để tạo môi trường giao tiếp giữa chương trình của người sử dụng và hệ điều hành, hệ điều hành đưa ra các lời gọi hệ thống. Chương trình của người sử dụng dùng các lời gọi hệ thống để liên lạc với hệ điều hành và yêu cầu các dịch vụ từ hệ điều hành.

Mỗi lời gọi hệ thống tương ứng với một thủ tục trong thư viện của hệ điều hành, do đó chương trình của người sử dụng có thể gọi thủ tục để thực hiện một lời gọi hệ thống. Lời gọi hệ thống còn được thiết dưới dạng các câu lệnh trong các ngôn ngữ lập trình cấp thấp. Lệnh gọi ngắt trong hợp ngữ (Int), và thủ tục gọi hàm API trong windows được xem là một lời gọi hệ thống.

Lời gọi hệ thống có thể được chia thành các loại: quản lý tiến trình, thao tác trên tập tin, thao tác trên thiết bị vào/ ra, thông tin liên tiến trình, ...

Sau đây là một số lời gọi hệ thống của hệ điều hành MS\_DOS:

- S = Load\_and\_exec(processname): tạo tiến trình con và thực hiện nó.
- Fd = Open(filename, mode): mở file để đọc hoặc/và ghi.
- N = Write(Fd, buffer, nbyte): ghi dữ liệu từ đệm vào file.
- Addr = alloc\_memory(nbyte): cấp phát một khối nhớ
- Keep\_pro(mem\_size, status): kết thúc và thường trú chương trình.

**Chú ý:** Cần phải phân biệt sự khác nhau giữa Shell và System Call. Shell tạo môi trường giao tiếp giữa người sử dụng và hệ điều hành, System Call tạo môi trường giao tiếp giữa chương trình người sử dụng và hệ điều hành.

## I.10. Hệ điều hành và phân loại hệ điều hành

### I.3.3. Hệ điều hành là gì?

Khó có một khái niệm hay định nghĩa chính xác về hệ điều hành, vì hệ điều hành là một bộ phận được nhiều đối tượng khai thác nhất, họ có thể là người sử dụng thông thường, có thể là lập trình viên, có thể là người quản lý hệ thống và tùy theo mức độ khai thác hệ điều hành mà họ có thể đưa ra những khái niệm khác nhau về nó. Ở đây ta xem xét 3 khái niệm về hệ điều hành dựa trên quan điểm của người khai thác hệ thống máy tính:

- **Khái niệm 1:** Hệ điều hành là một hệ thống mô hình hoá, mô phỏng hoạt động của máy tính, của người sử dụng và của lập trình viên, hoạt động trong chế độ đối thoại nhằm tạo môi trường khai thác thuận lợi hệ thống máy tính và quản lý tối ưu tài nguyên của hệ thống.

- **Khái niệm 2:** Hệ điều hành là hệ thống chương trình với các chức năng giám sát, điều khiển việc thực hiện các chương trình của người sử dụng, quản lý và phân chia tài nguyên cho nhiều chương trình người sử dụng đồng thời sao cho việc khai thác chức năng của hệ thống máy tính của người sử dụng là thuận lợi và hiệu quả nhất.

- **Khái niệm 3:** Hệ điều hành là một chương trình đóng vai trò như là giao diện giữa người sử dụng và phần cứng máy tính, nó điều khiển việc thực hiện của tất cả các loại chương trình. Khái niệm này rất gần với các hệ điều hành đang sử dụng trên các máy tính hiện nay.

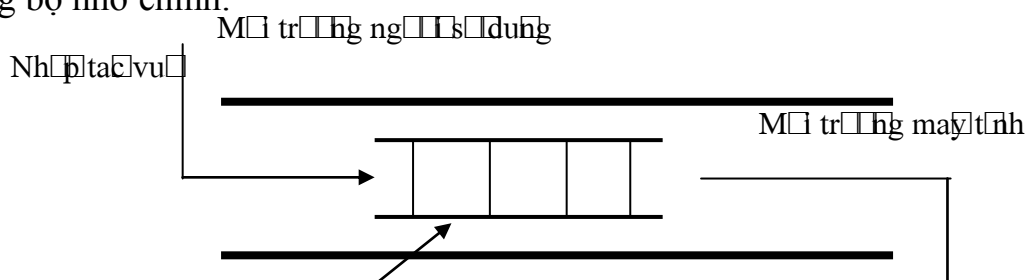
Từ các khái niệm trên chúng ta có thể thấy rằng: Hệ điều hành ra đời, tồn tại và phát triển là để giải quyết vấn đề sử dụng máy tính của người sử dụng, nhằm giúp người sử dụng khai thác hết các chức năng của phần cứng máy tính mà cụ thể là giúp người sử dụng thực hiện được các chương trình của họ trên máy tính.

### I.3.4. Phân loại hệ điều hành

Có nhiều cách khác nhau để phân loại hệ điều hành, ở đây chúng tôi dựa vào cách mà hệ điều hành thực hiện các công việc, các tác vụ, các tiến trình của người sử dụng để phân loại hệ điều hành.

#### I.3.2.a. Hệ điều hành xử lý theo lô đơn giản

Hệ điều hành loại này thực hiện các tác vụ lần lượt theo những chỉ thị đã được xác định trước. Khi một tác vụ chấm dứt thì hệ thống sẽ tự động thực hiện tác vụ tiếp theo mà không cần sự can thiệp từ bên ngoài, do đó hệ thống đạt tốc độ thực hiện cao. Để thực hiện được điều này hệ điều hành phải có bộ phận giám sát thường trực để giám sát việc thực hiện của các tác vụ trong hệ thống, bộ phận này thường trú trong bộ nhớ chính.



Trong hệ điều hành này khi hệ thống cần thực hiện một tác vụ thì nó phải lưu chương trình và dữ liệu của các tác vụ vào hàng đợi các công việc, sau đó sẽ thực hiện lần lượt từng bộ chương trình và dữ liệu của tác vụ tương ứng trong hàng đợi và cho ra lần lượt các kết quả. Hình 1.2 ở trên minh họa cho sự hoạt động của hệ thống theo lô đa chương.

➤ Với cách tổ chức *hàng đợi tác vụ*, thì hệ thống không thể thay đổi chương trình và dữ liệu của các tác vụ ngay cả khi chúng còn nằm trong hàng đợi, đây là một hạn chế. Mặt khác trong quá trình thực hiện tác vụ nếu tác vụ chuyển sang truy xuất trên thiết bị vào/ra thì processor rơi vào trạng thái chờ điều này gây lãng phí thời gian xử lý của processor.

#### **I.3.2.b. Hệ điều hành xử lý theo lô đa chương**

Một trong những hạn chế của hệ điều hành xử lý theo lô đơn giản là lãng phí thời gian xử lý của processor khi tác vụ hiện tại truy xuất đến thiết bị vào/ra. Hệ điều hành xử lý theo lô đa chương sẽ khắc phục hạn chế này.

Hệ điều hành loại này có khả năng thực hiện nhiều tác vụ, nhiều chương trình đồng thời. Khi cần thực hiện nhiều tác vụ đồng thời hệ điều hành sẽ nạp một phần code và data của các tác vụ vào bộ nhớ (các phần còn lại sẽ được nạp sau tại thời điểm thích hợp) và tất cả đều ở trạng thái sẵn sàng thực hiện, sau đó hệ điều hành bắt đầu thực hiện một tác vụ nào đó, nhưng khi tác vụ đang thực hiện cần truy xuất thiết bị vào/ra thì processor sẽ được chuyển sang thực hiện các tác vụ khác, và cứ như thế hệ điều hành tổ chức chuyển hướng processor để thực hiện hết các phần tác vụ trong bộ nhớ cũng như các tác vụ mà hệ thống yêu cầu.

➤ Hệ điều hành loại này mang lại hai ưu điểm đó là tiết kiệm được bộ nhớ, vì không nạp hết code và data của các tác vụ vào bộ nhớ, và hạn chế thời gian rỗi của processor. Tuy nhiên nó phải chi phí cao cho việc lập lịch processor, tức là khi có được processor hệ điều hành phải xem xét nên chuyển nó cho tác vụ nào trong số các tác vụ đang ở trạng thái sẵn sàng. Ngoài ra hệ điều hành còn phải giải quyết việc chia sẻ bộ nhớ chính cho các tác vụ khác nhau. Hệ điều hành MS\_DOS là hệ điều hành đơn nhiệm, đa chương.



### **I.3.2.c. Hệ điều hành chia sẻ thời gian**

Khái niệm chia sẻ thời gian ra đời đã đánh dấu một bước phát triển mới của hệ điều hành trong việc điều khiển các hệ thống đa người dùng. Chia sẻ thời gian ở đây chính là chia sẻ thời gian xử lý của processor cho các tác vụ, các tiến trình đang ở trong trạng thái sẵn sàng thực hiện.

Nguyên tắc của hệ điều hành chia sẻ thời gian tương tự như trong hệ điều hành xử lý theo lô đa chương nhưng việc chuyển processor từ tác vụ, tiến trình này sang tác vụ, tiến trình khác không phụ thuộc vào việc tác vụ, tiến trình hiện tại có truy xuất đến thiết bị vào/ra hay không mà chỉ phụ thuộc vào sự điều phối processor của hệ điều hành. Công việc điều phối processor của hệ điều hành rất phức tạp phụ thuộc vào nhiều yếu tố khác nhau, chúng ta sẽ đề cập đến vấn đề này trong chương sau của tài liệu này.

Trong hệ điều hành này thời gian chuyển đổi processor giữa các tác vụ là rất nhỏ nên ta có cảm giác các tác vụ thực hiện song song với nhau. Với hệ điều hành này người sử dụng có thể yêu cầu hệ điều hành thực hiện nhiều chương trình, tiến trình, tác vụ đồng thời với nhau.

➤ Hệ điều hành chia sẻ thời gian là mở rộng logic của hệ điều hành đa chương và nó thường được gọi là hệ điều hành đa nhiệm (Multitasking). Hệ điều hành Windows 9x/NT là các hệ điều hành đa nhiệm.

### **I.3.2.d. Hệ điều hành đa vi xử lý**

Là các hệ điều hành dùng để điều khiển sự hoạt động của các hệ thống máy tính có nhiều vi xử lý. Các hệ điều hành đa vi xử lý (multiprocessor) gồm có 2 loại:

□ **Đa xử lý đối xứng (SMP: symmetric):** Trong hệ thống này vi xử lý nào cũng có thể chạy một loại tiểu trình bất kỳ, các vi xử lý giao tiếp với nhau thông qua một bộ nhớ dùng chung. Hệ SMP cung cấp một cơ chế chịu lỗi và khả năng cân bằng tải tối ưu hơn, vì các tiểu trình của hệ điều hành có thể chạy trên bất kỳ vi xử lý nào nên nguy cơ xảy ra tình trạng tắc nghẽn ở CPU giảm đi đáng kể. Vấn đề đồng bộ giữa các vi xử lý được đặt lên hàng đầu khi thiết kế hệ điều hành cho hệ thống SMP. Hệ điều hành Windows NT, hệ điều hành Windows 2000 là các hệ điều hành đa xử lý đối xứng.

□ **Đa xử lý bất đối xứng (ASMP: asymmetric):** Trong hệ thống này hệ điều hành dành ra một hoặc hai vi xử lý để sử dụng riêng, các vi xử lý còn lại dùng để điều khiển các chương trình của người sử dụng. Hệ ASMP đơn giản hơn nhiều so với hệ SMP, nhưng trong hệ này nếu có một vi xử lý trong các vi xử lý dành riêng cho hệ điều hành bị hỏng thì hệ thống có thể ngừng hoạt động.

### **I.3.2.e. Hệ điều hành xử lý thời gian thực**

Hệ điều hành này khắc phục nhược điểm của hệ điều hành xử lý theo lô, tức là nó có khả năng cho kết quả tức thời, chính xác sau mỗi tác vụ.

Trong hệ điều hành này các tác vụ cần thực hiện không được đưa vào hàng đợi mà được xử lý tức thời và trả lại ngay kết quả hoặc thông báo lỗi cho người sử dụng có yêu cầu. Hệ điều hành này hoạt động đòi hỏi sự phối hợp cao giữa phần mềm và phần cứng.

### **I.3.2.f. Hệ điều hành mạng**

Là các hệ điều hành dùng để điều khiển sự hoạt động của mạng máy tính. Ngoài các chức năng cơ bản của một hệ điều hành, các hệ điều hành mạng còn phải thực hiện việc chia sẻ và bảo vệ tài nguyên của mạng. Hệ điều hành Windows 9x/NT, Windows 2000, Linux, là các hệ điều hành mạng máy tính.

➤ **Tóm lại:** Qua sự phân loại hệ điều hành ở trên ta có thể thấy được quá trình phát triển (evolution) của hệ điều hành. Để khắc phục hạn chế về lãng phí thời gian xử lý của processor trong hệ điều hành theo lô thì hệ điều hành theo lô đa chương ra đời. Để khai thác tối đa thời gian xử lý của processor và tiết kiệm hơn nữa không gian bộ nhớ chính hệ điều hành chia sẻ thời gian ra đời. Chia sẻ thời gian xử lý của processor kết hợp với chia sẻ không gian bộ nhớ chính đã giúp cho hệ điều hành có thể đưa vào bộ nhớ chính nhiều chương trình, tiến trình hơn và các chương trình, tiến trình này có thể hoạt động đồng thời với nhau, nhờ đó mà hiệu suất của hệ thống tăng lên, và cũng từ đây khái niệm hệ điều hành đa chương ra đời. Hệ điều hành đa xử lý và hệ điều hành mạng được phát triển dựa trên hệ điều hành đa nhiệm. Hệ điều hành thời gian thực ra đời là để khắc phục hạn chế của hệ điều hành theo lô và điều khiển các hệ thống thời gian thực. Từ đây chúng ta rút ra một điều rằng: *các hệ điều hành ra đời sau luôn tìm cách khắc phục các hạn chế của hệ điều hành trước đó và phát triển nhiều hơn nữa để đáp ứng yêu cầu ngày càng cao của người sử dụng và chương trình người sử dụng, cũng như khai thác tối đa các chức năng của phần cứng máy tính để nâng cao hiệu suất của hệ thống. Nhưng chức năng của hệ điều hành càng cao thì chi phí cho nó cũng tăng theo và cấu trúc của hệ điều hành cũng sẽ phức tạp hơn.*

Hệ điều hành Windows NT và hệ điều hành Windows 2000 là các hệ điều hành mạnh, nó có đầy đủ các chức năng của các loại hệ điều hành, do đó WindowsNT/2000 chứa rất nhiều thành phần với một cấu trúc khá phức tạp.

## **I.11. Thành phần và cấu trúc của hệ điều hành**

Hệ điều hành là một hệ thống chương trình lớn, thực hiện nhiều nhiệm vụ khác nhau, do đó các nhà thiết kế thường chia hệ điều hành thành nhiều thành phần, mỗi thành phần đảm nhận một nhóm các nhiệm vụ nào đó, các nhiệm vụ này có liên quan với nhau. Cách phân chia nhiệm vụ cho mỗi thành phần, cách kết nối các thành phần lại với nhau để nó thực hiện được một nhiệm vụ lớn hơn khi cần và cách gọi các thành phần này khi cần nó thực hiện một nhiệm vụ nào đó, ... , tất cả các phương thức trên tạo nên cấu trúc của hệ điều hành.

### **I.4.3. Các thành phần của hệ điều hành**

#### **I.4.1.a. Thành phần quản lý tiến trình**

Hệ điều hành phải có nhiệm vụ tạo lập tiến trình và đưa nó vào danh sách quản lý tiến trình của hệ thống. Khi tiến trình kết thúc hệ điều hành phải loại bỏ tiến trình ra khỏi danh sách quản lý tiến trình của hệ thống.

Hệ điều hành phải cung cấp đầy đủ tài nguyên để tiến trình đi vào hoạt động và phải đảm bảo đủ tài nguyên để duy trì sự hoạt động của tiến trình cho đến khi tiến trình kết thúc. Khi tiến trình kết thúc hệ điều hành phải thu hồi những tài nguyên mà hệ điều hành đã cấp cho tiến trình.

Trong quá trình hoạt động nếu vì một lý do nào đó tiến trình không thể tiếp tục hoạt động được thì hệ điều hành phải tạm dừng tiến trình, thu hồi tài nguyên mà tiến trình đang chiếm giữ, sau đó nếu điều kiện thuận lợi thì hệ điều hành phải tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động cho đến khi kết thúc.

Trong các hệ thống có nhiều tiến trình hoạt động song song hệ điều hành phải giải quyết vấn đề tranh chấp tài nguyên giữa các tiến trình, điều phối processor cho các tiến trình, giúp các tiến trình trao đổi thông tin và hoạt động đồng bộ với nhau, đảm bảo nguyên tắc tất cả các tiến trình đã được khởi tạo phải được thực hiện và kết thúc được.

➤ Tóm lại, bộ phận quản lý tiến trình của hệ điều hành phải thực hiện những nhiệm vụ sau đây:

- Tạo lập, hủy bỏ tiến trình.
- Tạm dừng, tái kích hoạt tiến trình.
- Tạo cơ chế thông tin liên lạc giữa các tiến trình.
- Tạo cơ chế đồng bộ hóa giữa các tiến trình.

#### **I.4.1.b. Thành phần quản lý bộ nhớ chính**

Bộ nhớ chính là một trong những tài nguyên quan trọng của hệ thống, đây là thiết bị lưu trữ duy nhất mà CPU có thể truy xuất trực tiếp được.

Các chương trình của người sử dụng muốn thực hiện được bởi CPU thì trước hết nó phải được hệ điều hành nạp vào bộ nhớ chính, chuyển đổi các địa chỉ sử dụng trong chương trình thành những địa chỉ mà CPU có thể truy xuất được.

Khi chương trình, tiến trình có yêu cầu được nạp vào bộ nhớ thì hệ điều hành phải cấp phát không gian nhớ cho nó. Khi chương trình, tiến trình kết thúc thì hệ điều hành phải thu hồi lại không gian nhớ đã cấp phát cho chương trình, tiến trình trước đó.

Trong các hệ thống đa chương hay đa tiến trình, trong bộ nhớ tồn tại nhiều chương trình/ nhiều tiến trình, hệ điều hành phải thực hiện nhiệm vụ bảo vệ các

vùng nhớ đã cấp phát cho các chương trình/ tiến trình, tránh sự vi phạm trên các vùng nhớ của nhau.

➤ Tóm lại, bộ phận quản lý bộ nhớ chính của hệ điều hành thực hiện những nhiệm vụ sau:

- Cấp phát, thu hồi vùng nhớ.
- Ghi nhận trạng thái bộ nhớ chính.
- Bảo vệ bộ nhớ.
- Quyết định tiến trình nào được nạp vào bộ nhớ.

#### **I.4.1.c. Thành phần quản lý xuất/ nhập**

Một trong những mục tiêu của hệ điều hành là giúp người sử dụng khai thác hệ thống máy tính dễ dàng và hiệu quả, do đó các thao tác trao đổi thông tin trên thiết bị xuất/ nhập phải *trong suốt* đối với người sử dụng.

Để thực hiện được điều này hệ điều hành phải tồn tại một bộ phận điều khiển thiết bị, bộ phận này phối hợp cùng CPU để quản lý sự hoạt động và trao đổi thông tin giữa hệ thống, chương trình người sử dụng và người sử dụng với các thiết bị xuất/ nhập.

Bộ phận điều khiển thiết bị thực hiện những nhiệm vụ sau:

- Gởi mã lệnh điều khiển đến thiết bị: Hệ điều hành điều khiển các thiết bị bằng các mã điều khiển, do đó trước khi bắt đầu một quá trình trao đổi dữ liệu với thiết bị thì hệ điều hành phải gởi mã điều khiển đến thiết bị.
- Tiếp nhận yêu cầu ngắt (Interrupt) từ các thiết bị: Các thiết bị khi cần trao đổi với hệ thống thì nó phát ra một tín hiệu yêu cầu ngắt, hệ điều hành tiếp nhận yêu cầu ngắt từ các thiết bị, xem xét và thực hiện một thủ tục để đáp ứng yêu cầu từ các thiết bị.
- Phát hiện và xử lý lỗi: quá trình trao đổi dữ liệu thường xảy ra các lỗi như: thiết bị vào ra chưa sẵn sàng, đường truyền hỏng, ... do đó hệ điều hành phải tạo ra các cơ chế thích hợp để phát hiện lỗi sớm nhất và khắc phục các lỗi vừa xảy ra nếu có thể.

#### **I.4.1.d. Thành phần quản lý bộ nhớ phụ (đĩa)**

Không gian lưu trữ của đĩa được chia thành các phần có kích thước bằng nhau được gọi là các block, khi cần lưu trữ một tập tin trên đĩa hệ điều hành sẽ cấp cho tập tin một lượng vừa đủ các block để chứa hết nội dung của tập tin. Block cấp cho tập tin phải là các block còn tự do, chưa cấp cho các tập tin trước đó, do đó sau khi thực hiện một thao tác cấp phát block hệ điều hành phải ghi nhận trạng thái của các block trên đĩa, đặc biệt là các block còn tự do để chuẩn bị cho các quá trình cấp block sau này.

Trong quá trình sử dụng tập tin nội dung của tập tin có thể thay đổi (tăng, giảm), do đó hệ điều hành phải tổ chức cấp phát động các block cho tập tin.

Để ghi/đọc nội dung của một block thì trước hết phải định vị đầu đọc/ ghi đến block đó. Khi chương trình của người sử dụng cần đọc nội dung của một dãy các block không liên tiếp nhau, thì hệ điều hành phải chọn lựa nên đọc block nào trước, nên đọc theo thứ tự nào,..., dựa vào đó mà hệ điều hành di chuyển đầu đọc đến các block thích hợp, nhằm nâng cao tốc độ đọc dữ liệu trên đĩa. Thao tác trên được gọi là lập lịch cho đĩa.

➤ Tóm lại, bộ phận quản lý bộ nhớ phụ thực hiện những nhiệm vụ sau:

- Quản lý không gian trống trên đĩa.
- Định vị lưu trữ thông tin trên đĩa.
- Lập lịch cho vấn đề ghi/ đọc thông tin trên đĩa của đầu từ.

#### **I.4.1.e. Thành phần quản lý tập tin**

Máy tính có thể lưu trữ thông tin trên nhiều loại thiết bị lưu trữ khác nhau, mỗi thiết bị lại có tính chất và cơ chế tổ chức lưu trữ thông tin khác nhau, điều này gây khó khăn cho người sử dụng. Để khắc phục điều này hệ điều hành đưa ra khái niệm đồng nhất cho tất cả các thiết bị lưu trữ vật lý, đó là *tập tin* (file).

Tập tin là đơn vị lưu trữ cơ bản nhất, mỗi tập tin có một tên riêng. Hệ điều hành phải thiết lập mối quan hệ tương ứng giữa tên tập tin và thiết bị lưu trữ chứa tập tin. Theo đó khi cần truy xuất đến thông tin đang lưu trữ trên bất kỳ thiết bị lưu trữ nào người sử dụng chỉ cần truy xuất đến tập tin tương ứng thông qua tên của nó, tất cả mọi việc còn lại đều do hệ điều hành thực hiện.

Trong hệ thống có nhiều tiến trình đồng thời truy xuất tập tin hệ điều hành phải tạo ra những cơ chế thích hợp để bảo vệ tập tin tránh việc ghi/ đọc bất hợp lệ trên tập tin.

➤ Tóm lại: Như vậy bộ phận quản lý tập tin của hệ điều hành thực hiện những nhiệm vụ sau:

- Tạo/ xoá một tập tin/ thư mục.
- Bảo vệ tập tin khi có hiện tượng truy xuất đồng thời.
- Cung cấp các thao tác xử lý và bảo vệ tập tin/ thư mục.
- Tạo mối quan hệ giữa tập tin và bộ nhớ phụ chứa tập tin.
- Tạo cơ chế truy xuất tập tin thông qua tên tập tin.

#### **I.4.1.f. Thành phần thông dịch lệnh**

Đây là bộ phận quan trọng của hệ điều hành, nó đóng vai trò giao tiếp giữa hệ điều hành và người sử dụng. Thành phần này chính là shell mà chúng ta đã biết ở trên. Một số hệ điều hành chứa shell trong nhân (kernel) của nó, một số hệ điều hành

khác thì shell được thiết kế dưới dạng một chương trình đặc biệt.

#### **I.4.1.g. Thành phần bảo vệ hệ thống**

Trong môi trường hệ điều hành đa nhiệm có thể có nhiều tiến trình hoạt động đồng thời, thì mỗi tiến trình phải được bảo vệ để không bị tác động, có chủ ý hay không chủ ý, của các tiến trình khác. Trong trường hợp này hệ điều hành cần phải có các cơ chế để luôn đảm bảo rằng các File, Memory, CPU và các tài nguyên khác mà hệ điều hành đã cấp cho một chương trình, tiến trình thì chỉ có chương trình tiến trình đó được quyền tác động đến các thành phần này.

Nhiệm vụ trên thuộc thành phần bảo vệ hệ thống của hệ điều hành. Thành phần này điều khiển việc sử dụng tài nguyên, đặc biệt là các tài nguyên dùng chung, của các tiến trình, đặc biệt là các tiến trình hoạt động đồng thời với nhau, sao cho không xảy ra sự tranh chấp tài nguyên giữa các tiến trình hoạt động đồng thời và không cho phép các tiến trình truy xuất bất hợp lệ lên các vùng nhớ của nhau.

➤ Ngoài ra các hệ điều hành mạng, các hệ điều hành phân tán hiện nay còn có thêm thành phần kết nối mạng và truyền thông..

➤ Để đáp ứng yêu cầu của người sử dụng và chương trình người sử dụng các nhiệm vụ của hệ điều hành được thiết kế dưới dạng các dịch vụ:

- Thi hành chương trình: hệ điều hành phải có nhiệm vụ nạp chương trình của người sử dụng vào bộ nhớ, chuẩn bị đầy đủ các điều kiện về tài nguyên để chương trình có thể chạy được và kết thúc được, có thể kết thúc bình thường hoặc kết thúc do bị lỗi. Khi chương trình kết thúc hệ điều hành phải thu hồi tài nguyên đã cấp cho chương trình và ghi lại các thông tin mà chương trình đã thay đổi trong quá trình chạy (nếu có).
- Thực hiện các thao tác xuất nhập dữ liệu: Khi chương trình chạy nó có thể yêu cầu xuất nhập dữ liệu từ một tập tin hoặc từ một thiết bị xuất nhập nào đó, trong trường hợp này hệ điều hành phải hỗ trợ việc xuất nhập dữ liệu cho chương trình, phải nạp được dữ liệu mà chương trình cần vào bộ nhớ.
- Thực hiện các thao tác trên hệ thống tập tin: Hệ điều hành cần cung cấp các công cụ để chương trình dễ dàng thực hiện các thao tác đọc ghi trên các tập tin, các thao tác này phải thực sự an toàn, đặc biệt là trong môi trường đa nhiệm.
- Trao đổi thông tin giữa các tiến trình: Trong môi trường hệ điều hành đa nhiệm, với nhiều tiến trình hoạt động đồng thời với nhau, một tiến trình có thể trao đổi thông tin với nhiều tiến trình khác, hệ điều hành phải cung cấp các dịch vụ cần thiết để các tiến trình có thể trao đổi thông tin với nhau và phối hợp cùng nhau để hoàn thành một tác vụ nào đó.
- Phát hiện và xử lý lỗi: Hệ điều hành phải có các công cụ để chính hệ

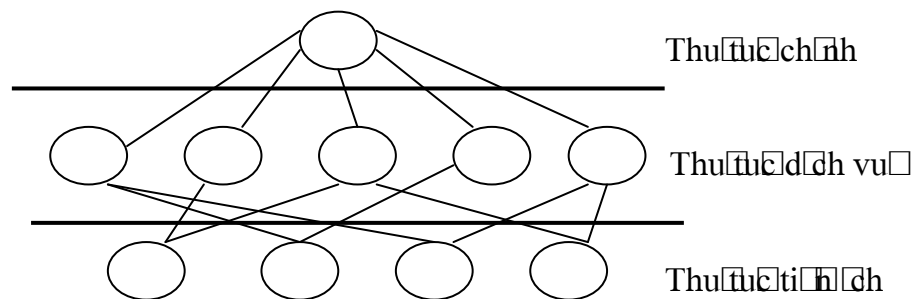
- Hình vẽ sau đây minh họa cho việc đáp ứng một lời gọi dịch vụ từ chương trình của người sử dụng dựa vào bảng chỉ mục.

3. Hệ điều hành xác định (vị trí) và gọi thủ tục dịch vụ tương ứng.
4. Hệ điều hành trả điều khiển lại cho chương trình người sử dụng.

Sau đây là một cấu trúc đơn giản của hệ thống đơn khối, trong cấu trúc này các thủ tục được chia thành 3 lớp:

1. Một chương trình chính (chương trình của người sử dụng) gọi đến một thủ tục dịch vụ của hệ điều hành. Lời gọi này được gọi là lời gọi hệ thống.
2. Một tập các thủ tục dịch vụ (service) để đáp ứng những lời gọi hệ thống từ các chương trình người sử dụng.
3. Một tập các thủ tục tiện ích (utility) hỗ trợ cho các thủ tục dịch vụ trong việc thực hiện cho các lời gọi hệ thống.

Trong cấu trúc này mỗi lời gọi hệ thống sẽ gọi một thủ tục dịch vụ tương ứng. Thủ tục tiện ích thực hiện một vài điều gì đó mà thủ tục dịch vụ cần, chẳng hạn như nhận dữ liệu từ chương trình người sử dụng. Các thủ tục của hệ điều hành được chia vào 3 lớp theo như hình vẽ dưới đây.



**Hình 1.4:** Cấu trúc đơn giản của mô hình monolithic system

#### Nhận xét:

- Với cấu trúc này chương trình của người sử dụng có thể truy xuất trực tiếp đến các chi tiết phần cứng bằng cách gọi một thủ tục cấp thấp, điều này gây khó khăn cho hệ điều hành trong việc kiểm soát và bảo vệ hệ thống.
- Các thủ tục dịch vụ mang tính chất tĩnh, nó chỉ hoạt động khi được gọi bởi chương trình của người sử dụng, điều này làm cho hệ điều hành thiếu chủ động trong việc quản lý môi trường.

#### I.4.2.b. Các hệ thống phân lớp (Layered Systems)

Hệ thống được chia thành một số lớp, mỗi lớp được xây dựng dựa vào lớp bên trong. Lớp trong cùng thường là phần cứng, lớp ngoài cùng là giao diện với người sử dụng.

Mỗi lớp là một đối tượng trừu tượng, chứa đựng bên trong nó các dữ liệu và thao tác xử lý dữ liệu đó. Lớp n chứa đựng một cấu trúc dữ liệu và các thủ tục có thể được gọi bởi lớp n+1 hoặc ngược lại có thể gọi các thủ tục ở lớp n-1.



Ví dụ về một hệ điều hành phân lớp:

Lớp 5: Chương trình ứng dụng

Lớp 4: Quản lý bộ đệm cho các thiết bị xuất nhập

Lớp 3: Trình điều khiển thao tác console

Lớp 2: Quản lý bộ nhớ

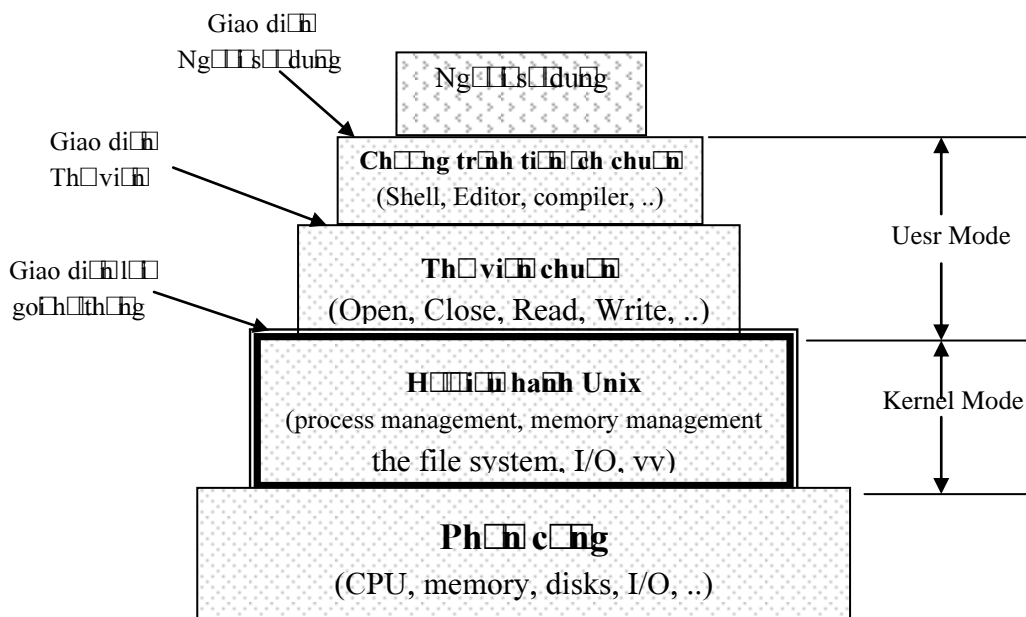
Lớp 1: Điều phối processor

Lớp 0: Phần cứng hệ thống

Hình vẽ 1.5 sau đây cho ta thấy cấu trúc phân lớp trong hệ điều hành Unix.

### Nhận xét:

- Khi xây dựng hệ điều hành theo hệ thống này các nhà thiết kế gặp khó khăn trong việc xác định số lượng lớp, thứ tự và chức năng của mỗi lớp.



**Hình 1.5:** Hệ thống phân lớp của UNIX

- Hệ thống này mang tính đơn thể, nên dễ cài đặt, tìm lỗi và kiểm chứng hệ thống.
- Trong một số trường hợp lời gọi thủ tục có thể lan truyền đến các thủ tục khác ở các lớp bên trong nên chi phí cho vấn đề truyền tham số và chuyển đổi ngữ cảnh tăng lên, dẫn đến lời gọi hệ thống trong cấu trúc này thực hiện chậm hơn so với các cấu trúc khác.

### I.4.2.c. Máy ảo (Virtual Machine)

Thông thường một hệ thống máy tính bao gồm nhiều lớp: phần cứng ở lớp thấp

nhất, hạt nhân ở lớp kế trên. Hạt nhân dùng các chỉ thị (lệnh máy) của phần cứng để tạo ra một tập các lời gọi hệ thống. Các hệ điều hành hiện đại thiết kế một lớp các chương trình hệ thống nằm giữa hệ điều hành và chương trình của người sử dụng.

Các chương trình hệ thống có thể sử dụng các lời gọi hệ thống hoặc sử dụng trực tiếp các chỉ thị phần cứng để thực hiện một chức năng hoặc một thao tác nào đó, do đó các chương trình hệ thống thường xem các lời gọi hệ thống và các chỉ thị phần cứng như ở trên cùng một lớp.

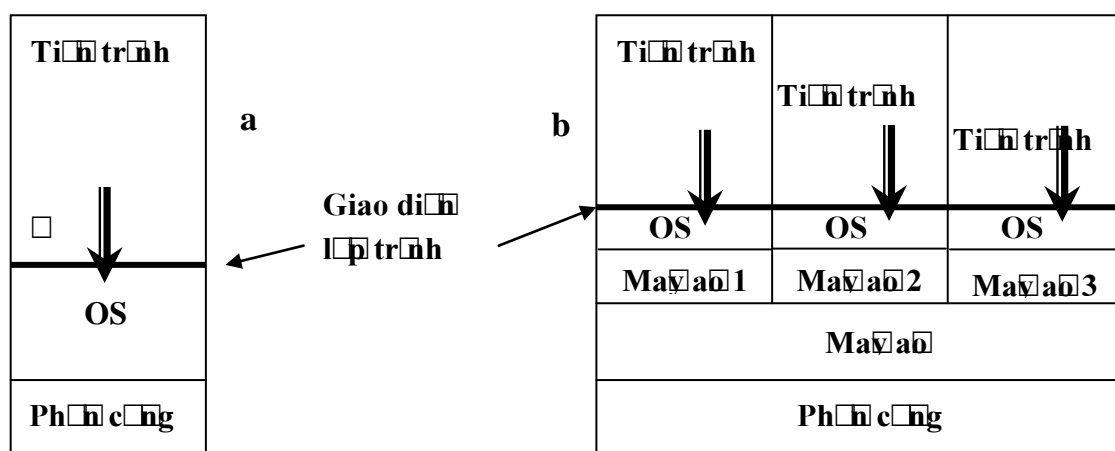
Một số hệ điều hành tổ cho phép các chương trình của người sử dụng có thể gọi dễ dàng các chương trình hệ thống và xem mọi thành phần dưới chương trình hệ thống đều là phần cứng máy tính. Lớp các ứng dụng này sử dụng khái niệm máy ảo.

Mục đích của việc sử dụng máy ảo là xây dựng các hệ thống đa chương với nhiều tiến trình thực hiện đồng thời, mỗi tiến trình được cung cấp một máy ảo với đầy đủ tài nguyên, tất nhiên là tài nguyên ảo, để nó thực hiện được.

Trong cấu trúc này phần nhân của hệ thống trở thành bộ phận tổ chức giám sát máy ảo, phần này chịu trách nhiệm giao tiếp với phần cứng, chia sẻ tài nguyên hệ thống để tạo ra nhiều máy ảo, hoạt động độc lập với nhau, để cung cấp cho lớp trên.

Ở đây cần phân biệt sự khác nhau giữa máy ảo và máy tính mở rộng, máy ảo là bản sao chính xác các đặc tính phần cứng của máy tính thực sự và cho phép hệ điều hành hoạt động trên nó, sau đó hệ điều hành xây dựng máy tính mở rộng để cung cấp cho người sử dụng.

Với cấu trúc này mỗi tiến trình hoạt động trên một máy ảo độc lập và nó có cảm giác như đang sở hữu một máy tính thực sự.



**Hình 1.6:** Mô hình hệ thống (a) Hệ có máy ảo (b) Máy ảo

Hình vẽ trên đây cho chúng ta thấy sự khác nhau trong hệ thống không có máy ảo và hệ thống có máy ảo:

**Nhận xét:**

- Việc cài đặt các phần mềm giả lập phần cứng để tạo ra máy ảo thường rất khó khăn và phức tạp.
- Trong hệ thống này vấn đề bảo vệ tài nguyên hệ thống và tài nguyên đã cấp phát cho các tiến trình, sẽ trở nên đơn giản hơn vì mỗi tiến trình thực hiện trên một máy tính (ảo) độc lập với nhau nên việc tranh chấp tài nguyên là không thể xảy ra.
- Nhờ hệ thống máy ảo mà một ứng dụng được xây dựng trên hệ điều hành có thể hoạt động được trên hệ điều hành khác. Trong môi trường hệ điều hành Windows 9x người sử dụng có thể thực hiện được các ứng dụng được thiết kế để thực hiện trên môi trường MS\_DOS, sở dĩ như vậy là vì Windows đã cung cấp cho các ứng dụng này một máy ảo DOS (VMD: Virtual Machine DOS) để nó hoạt động như đang hoạt động trong hệ điều hành DOS. Tương tự trong môi trường hệ điều hành Windows NT người sử dụng có thể thực hiện được các ứng dụng được thiết kế trên tất cả các hệ điều hành khác nhau, có được điều này là nhờ trong cấu trúc của Windows NT có chứa các hệ thống con (subsystems) môi trường tương thích với các môi trường hệ điều hành khác nhau như: Win32, OS/2,..., các ứng dụng khi cần thực hiện trên Windows NT sẽ thực hiện trong các hệ thống con môi trường tương ứng, đúng với môi trường mà ứng dụng đó được tạo ra.

**I.4.2.d. Mô hình Client/ Server (client/ server model)**

Các hệ điều hành hiện đại thường chuyển dần các tác vụ của hệ điều hành ra các lớp bên ngoài nhằm thu nhỏ phần cốt lõi của hệ điều hành thành hạt nhân cực tiểu (kernel) sao cho chỉ phần hạt nhân này phụ thuộc vào phần cứng. Để thực hiện được điều này hệ điều hành xây dựng theo mô hình Client/ Server, theo mô hình này hệ điều hành bao gồm nhiều tiến trình đóng vai trò Server có các chức năng chuyên biệt như quản lý tiến trình, quản lý bộ nhớ, ..., phần hạt nhân của hệ điều hành chỉ thực hiện nhiệm vụ tạo cơ chế thông tin liên lạc giữa các tiến trình Client và Server.

Như vậy các tiến trình trong hệ thống được chia thành 2 loại:

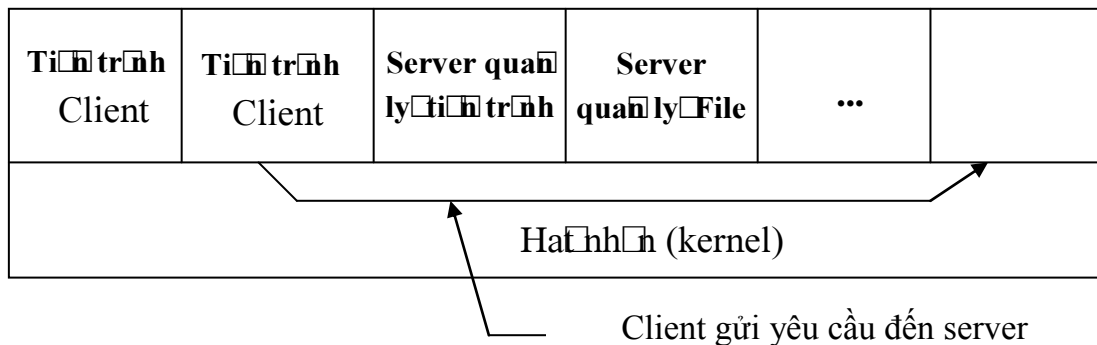
- Tiến trình bên ngoài hay tiến trình của chương trình người sử dụng được gọi là các tiến trình Client.
- Tiến trình của hệ điều hành được gọi là tiến trình Server.

Khi cần thực hiện một chức năng hệ thống các tiến trình Client sẽ gửi yêu

cầu tới tiến trình server tương ứng, tiến trình server sẽ xử lý và trả lời kết quả cho tiến trình Client.

### Nhận xét:

- Hệ thống này dễ thay đổi và dễ mở rộng hệ điều hành. Để thay đổi các chức năng của hệ điều hành chỉ cần thay đổi ở server tương ứng, để mở rộng hệ điều hành chỉ cần thêm các server mới vào hệ thống.
- Các tiến trình Server của hệ điều hành hoạt động trong chế độ không đặc quyền nên không thể truy cập trực tiếp đến phần cứng, điều này giúp hệ thống được bảo vệ tốt hơn.

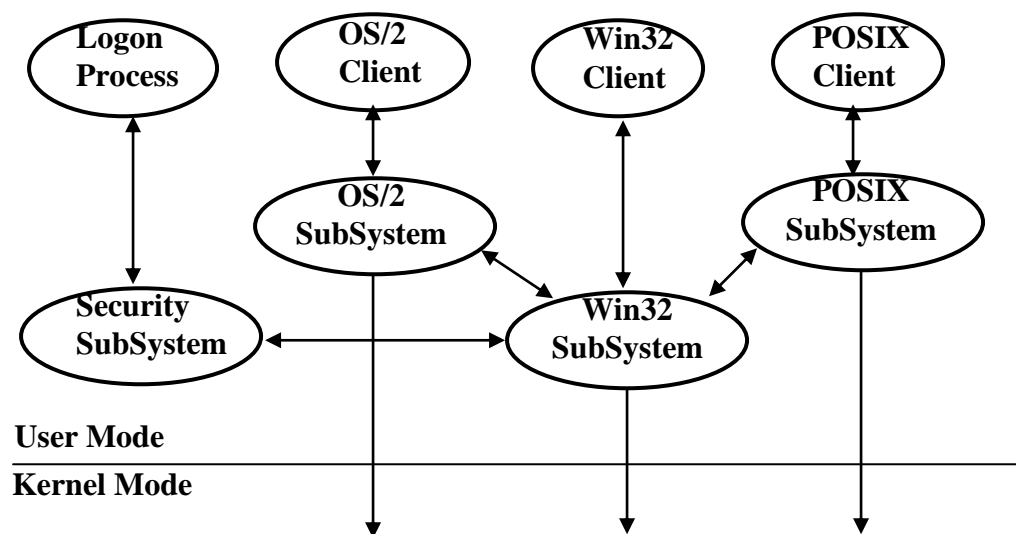


**Hình 1.7:** Mô hình client- server

➤ Hình vẽ sau đây cho thấy cấu trúc của hệ điều hành Windows NT. Đây là một cấu trúc phức tạp với nhiều thành phần khác nhau và nó được xây dựng dựa trên mô hình hệ điều hành Client/ Server.

Trong cấu trúc này chúng ta thấy nổi rõ hai điểm sau đây:

- Cấu trúc của windows NT được chia thành 2 mode: Kernel mode và User mode. Các chương trình ứng dụng của người sử dụng chỉ chạy trong User mode, các dịch vụ của hệ điều hành chỉ chạy trong Kernel mode. Nhờ vậy mà việc bảo vệ các chương trình của người sử dụng cũng như các thành phần của hệ điều hành, trên bộ nhớ, được thực hiện dễ dàng hơn.



- Trong User mode của Windows NT có chứa các hệ thống con môi trường như: OS/2 subsystem và POSIX subsystem, nhờ có các hệ thống con môi trường này mà các ứng dụng được thiết kế trên các hệ điều hành khác vẫn chạy được trên hệ điều hành Windows NT. Đây là điểm mạnh của các hệ điều hành Microsoft của từ Windows NT.

Chúng tôi sẽ giải thích rõ hơn về hai khái niệm **Kernel mode** và **User mode**, và các thành phần trong cấu trúc của hệ điều hành Windows NT ở phần sau, thông qua việc giới thiệu về hệ điều hành Windows 2000.

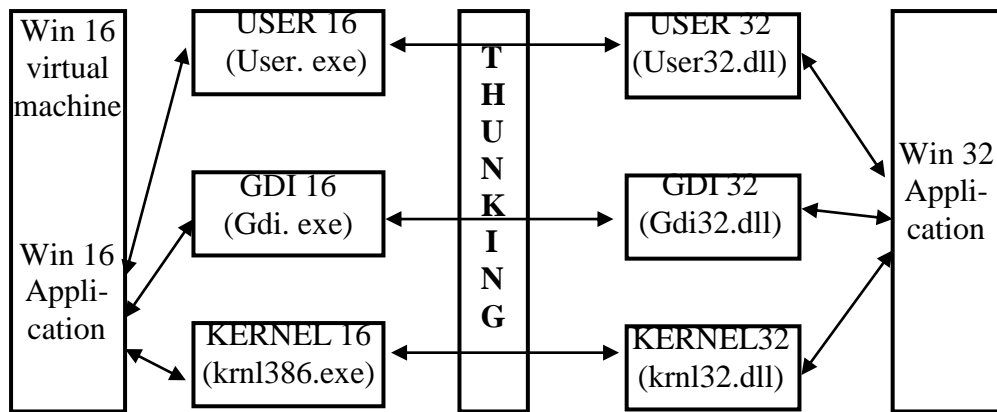
## **I.12. Hệ điều hành Windows95**

### **I.5.4. Giới thiệu về hệ điều hành Windows95**

Windows95 là kết quả của một sự phát triển lớn từ windows31. Microsoft không chọn giải pháp nâng cấp windows31 mà nó thực hiện việc kiến trúc lại windows để nó đủ mạnh để nó có thể thực hiện được các ứng dụng 32 bit trong một môi trường ổn định. Kết quả là Microsoft có được một phiên bản hệ điều hành windows95 đủ mạnh, có độ tin cậy và độ ổn định cao, và đặc biệt là cho phép các ứng dụng 16 bit và DOS chạy trên môi trường của nó.

Windows95 giữ lại các thành phần hệ thống của windows31 để đảm bảo tương thích với Win16, USER16, GDI và các thành phần Kernel 16 bit.

Một trong những thành phần quan trọng của windows95 là thành phần Thunking. Nhờ có Thunking mà các modul 16 bit có thể giao tiếp với các bản sao 32 bit của chúng và ngược lại. Thunking là một tập các thường trình, mà nó ánh xạ các địa chỉ để cho phép các ứng dụng phân đoạn 16 bit chia sẻ hoàn toàn bộ nhớ phẳng (flat) với các ứng dụng 32 bit. Hình vẽ sau đây cho thấy vai trò và vị trí của lớp Thunking trong windows95.



**Hình 1.9:** Lớp Thunking trong Windows95

### ➤ Kiến trúc 32 bit của Intel

Hãng Intel đưa ra vi xử lý 32 bit (80386) đầu tiên cách đây 10 năm, nhưng đến khi hệ điều hành windows95 ra đời thì những điểm mạnh trong kiến trúc của nó mới được phát huy, vì windows95 đã tận dụng được các điểm mạnh trong kiến trúc của Intel 32 bit để xây dựng thành một hệ điều hành 32 bit đủ mạnh. Các hệ điều hành 32 bit có thể truy xuất bộ nhớ theo mô hình bộ nhớ phẳng, trong mô hình này hệ điều hành có thể đánh địa chỉ bộ nhớ theo kiểu tuyến tính lên đến 4Gb, tức là nó loại trừ được sự phân đoạn bộ nhớ mà chúng ta đã thấy trong các hệ điều hành 16 bit. Khi chạy trên vi xử lý 80386 hệ điều hành windows95 khai thác tối đa các điểm mạnh trong chế độ ảo của vi xử lý này, vi xử lý 80386 có thể hoạt động ở các chế độ: thực (real mode), bảo vệ (protected mode) và ảo (virtual mode). Chế độ ảo của 80386 còn được gọi là chế độ 8086 ảo, trong chế độ 8086 ảo ngoài việc cung cấp không gian bộ nhớ ảo cho các ứng dụng, 80386 còn cho phép các ứng dụng chế độ 8086 ảo thực thi trong chế độ 8086 ảo, thực tế thực thi trong chế độ bảo vệ. Các ứng dụng chạy trong chế độ bảo vệ được hệ điều hành bảo vệ trên bộ nhớ và được truy xuất một không gian bộ nhớ lớn hơn (đến 4Gb bộ nhớ RAM). Nhờ có chế độ 8086 ảo mà windows95 có thể cho chạy nhiều ứng dụng đồng thời, kể cả các ứng dụng 16 bit của DOS và các ứng dụng 32 bit của windows, trên bộ nhớ và các ứng dụng này được hệ điều hành bảo vệ để các ứng dụng không truy xuất bất hợp lệ lên các vùng nhớ của nhau, nếu có một ứng dụng bị hỏng thì các ứng dụng còn lại vẫn

hoạt động bình thường. Windows95 xây dựng các máy ảo DOS để chạy các ứng dụng 16 bit của DOS.

Intel 80386 là một vi xử lý 32 bit, nhưng nếu sử dụng với hệ điều hành 16 bit thì các hệ điều hành này xem nó như là các vi xử lý 80286 16 bit, nên khả năng quản lý bộ nhớ của nó sẽ bị giới hạn. Việc xử lý dữ liệu trong môi trường 32 bit cũng có nhiều điểm lợi hơn trong môi trường 16 bit. Cùng một ứng dụng đó nhưng nếu chạy trong môi trường 16 bit thì nó phải chia thành các phân đoạn 16 bit và chỉ có thể truy xuất dữ liệu trong không gian 64Kb, nhưng khi chạy trong môi trường 32 bit thì nó không cần chia nhỏ và có thể truy xuất dữ liệu trong không gian bộ nhớ 4Gb, trong trường hợp này dữ liệu được tham chiếu theo kiểu tuyến tính nên tốc độ truy xuất được cải thiện hơn.

### ➤ **Kiến trúc vòng bảo vệ của Intel**

Kiến trúc vòng của Intel là cơ sở để hệ điều hành windows95 xây dựng các cơ chế bảo vệ các vùng nhớ đã cấp phát cho các ứng dụng trong môi trường có nhiều ứng dụng hoạt động đồng thời, cũng như bảo vệ vùng nhớ của hệ điều hành, không cho các ứng dụng truy xuất lên vùng nhớ của nhau và không cho các ứng dụng truy xuất lên vùng nhớ chứa chính hệ điều hành.

Tất cả các vi xử lý Intel từ 80386 trở về sau đều duy trì kiến trúc 4 vòng (Ring), các ring cũng được hiểu như là các cấp độ ưu tiên của hệ thống, tuy nhiên windows95 chỉ sử dụng hai ring: ring 0 và ring 3. Trong windows95 tất cả các ứng dụng đều chạy tại ring 3 (được xem như chế độ người sử dụng), mà nó được ngăn cản truy xuất đến các vùng nhớ khác. Điều này đảm bảo rằng một ứng dụng không thể làm hỏng toàn bộ hệ thống. Các thành phần của hệ điều hành chạy tại ring 0 (được xem như chế độ kernel), các tiến trình chạy tại ring 0 không bị giới hạn truy xuất đến hệ thống (ring 0 có độ ưu tiên cao nhất, ring 3 có độ ưu tiên thấp nhất) nên code của nó phải thực sự tin cậy. Các tiến trình ở ring 3 phải thông qua các tiến trình ở ring 0 để truy xuất vào hệ thống.

### ➤ **Mô hình đa nhiệm trong Windows95**

Windows95 là hệ điều hành đa nhiệm, nhờ có khả năng đa nhiệm mà windows95 có thể cho phép nhiều ứng dụng hoạt động đồng thời, nếu có một ứng dụng trong số này bị hỏng không thể tiếp tục thì các ứng dụng còn lại vẫn hoạt động bình thường. Windows95 có hai hình thức đa nhiệm: Đa nhiệm hợp tác (Cooperative Multitasking) và đa nhiệm ưu tiên (Preemptive Multitasking).

Trong mô hình đa nhiệm hợp tác, chỉ có ứng dụng đang sở hữu processor mới quyết định khi nào trở lại processor cho tiến trình khác hoạt động. Trong mô hình đa nhiệm ưu tiên thì việc chuyển processor từ ứng dụng hiện tại cho tiến trình khác được thực hiện bởi bộ phận lập lịch của hệ điều hành. Bộ phận lập lịch quyết định thời gian mà mỗi tiến trình được sở hữu processor, khi nào thì dừng tiến trình hiện tại để thu hồi processor, khi có được processor thì chuyển nó cho tiến trình

nào trong số các tiến trình đang chờ được cấp processor. Bộ phận lập lịch thường dựa vào độ ưu tiên của tiến trình để quyết định việc cấp processor cho nó. Các ứng dụng win32 đều hoạt động trong môi trường đa nhiệm ưu tiên, trong khi đó các ứng dụng win16 hoạt động trong môi trường đa nhiệm hợp tác.

### **1.5.5. Cấu trúc của windows95**

Có nhiều thành phần tạo nên cấu trúc của windows95, mỗi thành phần thực hiện một chức năng nào đó của môi trường windows. Windows95 có 4 thành phần chính:

#### ➤ **Máy ảo hệ thống (VM: virtual machine):**

Một trong những thành phần chính của windows95 là trình quản lý máy ảo. Trình quản lý máy ảo điều khiển các ứng dụng MS\_DOS, các ứng dụng windows, các trình điều khiển thiết bị ảo (VxD), và các thành phần cơ sở chính của windows. Các máy ảo có thể là máy ảo hệ thống hoặc các máy ảo DOS.

Máy ảo hệ thống cung cấp đầy đủ các chức năng dành riêng cho người sử dụng windows95, nhờ có nó mà các chương trình của người sử dụng có thể chạy trên windows. Nó gồm 3 yếu tố chính: Các ứng dụng windows 32bit, shell, và các ứng dụng windows 16 bit:

- Các ứng dụng windows 32 bit: là các ứng dụng dành riêng cho win32, nó cung cấp khả năng đa nhiệm tốt hơn so với các ứng dụng 16 bit. Tất cả các ứng dụng 32 bit đều sử dụng một không gian địa chỉ duy nhất. Windows sử dụng chế độ đa nhiệm ưu tiên (preemptive multitasking) để đảm bảo mỗi tác vụ đều được chia sẻ công bằng tài nguyên của hệ thống.
- Môi trường shell: đó là windows explorer, explorer cung cấp đầy đủ các khả năng 32 bit. Hay nói cách khác Shell là một ứng dụng 32 bit.
- Các ứng dụng windows 16 bit: đó là các ứng dụng được xây dựng trên các hệ điều hành trước windows95. Windows95 cho chạy tất cả các ứng dụng này trong một không gian địa chỉ dùng chung và các ứng dụng này được đối xử như một tác vụ duy nhất. Windows sử dụng chế độ đa nhiệm hợp tác (cooperative multitasking) cho các ứng dụng ở đây.

➤ **Máy ảo DOS (VMD: virtual machine DOS):** Là thành phần dành riêng cho các ứng dụng MS\_DOS. Nhờ có các máy ảo DOS mà các ứng dụng được xây dựng trên nền hệ điều hành MS\_DOS vẫn có thể chạy trên môi trường hệ điều hành windows95. Có thể có nhiều máy ảo đồng thời chạy trên windows, nhờ đó mà ta có thể cho phép nhiều ứng dụng DOS chạy trên môi trường windows. Mỗi máy ảo có một vùng nhớ riêng của nó và nó đều truy xuất đến các thiết bị trên hệ thống. Các máy ảo DOS chạy trong chế độ 8086 ảo của các vi xử lý, nhờ đó mà nó được bảo vệ và nếu có một ứng dụng DOS bị hỏng khi đang chạy (Crash) thì các ứng dụng khác vẫn hoạt động bình thường.



➤ **Giao diện lập trình ứng dụng (API: application Programming Interface):** Có 2 loại API 16 bit và 32 bit. API 32 bit của windows95 cung cấp một tập các dịch vụ mà tất cả các ứng dụng 32 bit có thể truy xuất được, các ứng dụng Win 32 bit được hưởng các lợi ích mà giao diện API này cung cấp. API 32 bit bao gồm các thành phần cơ bản: KERNEL32.DLL, USER32.DLL, GDI32.DLL, các thành phần này được gọi là hệ thống con windows (windows subsystem):

- Kernel32.DLL: Phần hạt nhân của windows, nó cung cấp một sự hỗ trợ cho những chức năng ở mức thấp mà một ứng dụng cần để chạy, nếu ứng dụng cần bộ nhớ thì nó sẽ nhận từ Kernel.
- GDI32.DLL: Giao diện thiết bị đồ họa của windows, nó thực hiện các chức năng về Font chữ, máy in, màn hình, ...
- User32.DLL: Giao tiếp người sử dụng.

➤ **Hệ thống cơ sở (Base System):** Thành phần này chứa tất cả các dịch vụ đặc trưng của hệ điều hành. Đây là phần lõi (core) của windows95, nó bao gồm:

- Hệ thống con quản lý tập tin (File Management): thành phần này cung cấp một khả năng giao tiếp với tất cả các thiết bị khối có trên máy tính, nối trực tiếp hoặc thông qua mạng, nó giúp máy tính truy xuất được đến các thiết bị này.
- Hệ thống con quản mạng (Network Management Subsystem)
- Các dịch vụ hệ điều hành (Operating System Services)
- Bộ quản lý máy ảo (Virtual Machine Manager): Bộ phận này thực hiện các nhiệm vụ sau: Lập lịch cho các tác vụ; Khởi động cũng như kết thúc mọi ứng dụng có trên hệ thống, kể cả các ứng dụng DOS; Cấp phát bộ nhớ và quản lý cả bộ nhớ ảo của hệ thống; Giúp các tiến trình trao đổi thông tin với nhau.
- Các trình điều khiển thiết bị: Các trình điều khiển thiết bị tiếp nhận các yêu cầu của windows và trao chúng cho các thiết bị dưới khuôn dạng mà thiết bị đó có thể hiểu được. Windows95 hỗ trợ hai loại trình điều khiển thiết bị. Thứ nhất, là trình điều khiển thiết bị chế độ thực, hoạt động trong chế độ thực, mà ta đã dùng trong windows3.1. Thứ hai, là các trình điều khiển thiết bị ảo, hoạt động trong chế độ bảo vệ, đó là các VxD: Virtual Anything Drivers, các VxD cho phép windows trao đổi với các thiết bị mà không cần chuyển qua chế độ thực. Với các VxD hệ thống sẽ chạy ổn định hơn, nhanh hơn, và khả năng phục hồi lỗi tốt hơn so với các trình điều khiển thiết bị trong chế độ thực. Tuy nhiên các VxD có thể làm hỏng hệ thống, vì code của nó hoạt động tại ring 0.

➤ **Một thành phần không thể không nhắc đến trong môi trường windows đó là các DLL (Dynamic Link Library: Thư viện liên kết động):** Trong môi

trường hệ điều hành Windows, tại một thời điểm có thể có nhiều chương trình đồng thời hoạt động, và các chương trình này có thể cùng sử dụng một đoạn mã giống nhau nào đó. Như vậy trong bộ nhớ sẽ tồn tại nhiều đoạn mã giống nhau để đáp ứng cho các chương trình khác nhau, điều này gây lãng phí bộ nhớ. Để khắc phục Windows 9x đưa ra các tập tin DLL, DLL chứa các đoạn mã mà các ứng dụng thường sử dụng. DLL được nạp vào bộ nhớ ngay sau khi khởi động hệ điều hành để sẵn sàng phục vụ các ứng dụng hoặc được nạp vào bộ nhớ khi nó được gọi lần đầu tiên. Hệ điều hành luôn giám sát việc sử dụng DLL của các ứng dụng, khi không còn một ứng dụng nào sử dụng DLL thì nó được giải phóng ra khỏi bộ nhớ. Các mã trong DLL sẽ được liên kết vào các ứng dụng khi các ứng dụng được nạp vào bộ nhớ, các ứng dụng truy cập vào hệ thống thông qua các DLL. Như vậy nhờ có DLL mà windows linh động hơn và tiết kiệm được nhiều bộ nhớ hơn.

#### **I.5.6. Bộ nhớ ảo (Virtual Memory) trong windows95**

Mặc dù các tiến trình win32 có thể sử dụng đến 4GB bộ nhớ RAM, nhưng các giới hạn phần cứng hiện nay ngăn cản hầu hết các máy tính chứa nhiều bộ nhớ. Để mở rộng giới hạn bộ nhớ này các vi xử lý đã đưa ra các mô hình quản lý bộ nhớ khác nhau nhằm mở rộng khả năng quản lý bộ nhớ của vi xử lý cũng như cung cấp nhiều hơn không gian bộ nhớ cho các tiến trình. Vi xử lý 80386 đã sử dụng mô hình bộ nhớ ảo.

Với vi xử lý 80386 không gian bộ nhớ được chia thành các phân đoạn (segmentation), mỗi phân đoạn lại được chia thành các phân trang (paging), các phân trang đều có kích thước bằng nhau và bằng 4Kb. CPU cũng như hệ điều hành sử dụng các trang bộ nhớ để chứa code và data của các tiến trình, trong trường hợp này các tiến trình cũng được chia thành các trang có kích thước bằng các trang bộ nhớ.

Trong mô hình bộ nhớ ảo CPU không nạp tất cả các trang của tiến trình vào bộ nhớ RAM mà chỉ nạp các trang cần thiết ban đầu, các trang còn lại sẽ được nạp sau đó nếu cần. CPU dùng các bảng trang (PCT: Page Control Table) để theo dõi một trang của tiến trình là đã được nạp vào bộ nhớ RAM hay chưa. Khi có một trang mới của tiến trình được nạp vào bộ nhớ hoặc khi có một trang của tiến trình bị đưa ra lại đĩa thì hệ thống phải thực hiện việc cập nhật lại PCT.

Khi có yêu cầu nạp một trang tiến trình mới vào bộ nhớ nhưng trên bộ nhớ không còn trang trống thì CPU cùng với hệ điều hành sẽ tìm một trang tiến trình nào đó không thực sự cần thiết tại thời điểm hiện tại, thường là trang ít được sử dụng gần đây nhất, để đưa ra đĩa (swap out), để lấy khung trang trống đó nạp trang tiến trình vừa yêu cầu, trang tiến trình bị đưa ra đĩa này sẽ được CPU và hệ điều hành nạp vào lại bộ nhớ (swap in) tại một thời điểm thích hợp sau này. Các trang bị swap out thường được chứa trong một tập tin nào đó trên đĩa cứng, và được gọi là các tập tin swap. Trong windows95 các tập tin swap không bị giới hạn kích thước.

Khi người sử dụng khởi động một ứng dụng thì windows95 sẽ khởi tạo một tập tin swap có kích thước ban đầu bằng kích thước của ứng dụng để sẵn sàng chứa các trang của ứng dụng khi các trang này bị CPU swap out ra đĩa.

Windows95 thiết kế các tập tin swap theo kiểu động, tức là kích thước của nó có thể thay đổi tùy theo số trang mà nó chứa. Nếu có nhiều trang bị swap out thì kích thước của nó tăng lên, nếu các trang trong nó được swap in vào lại bộ nhớ RAM thì kích thước của nó sẽ tự động giảm xuống.

## **I.13. Hệ điều hành Windows 2000**

### **I.6.5. Giới thiệu về hệ điều hành Windows 2000**

Windows 2000 được thiết kế để chạy trên các kiến trúc phần cứng khác nhau như: Các hệ thống dựa trên nền Intel CISC và RISC, Alpha AXP, Motorola PowerPC, .... Nó được viết bởi C và C++, ngôn ngữ assembly chỉ được sử dụng để viết các thành phần giao tiếp trực tiếp với phần cứng, mã ngôn ngữ assembly không chỉ tồn tại trong kernel và HAL mà nó còn tồn tại trong phần kernel mode của hệ thống con Win32, và trong một vài thư viện của user mode.

Windows 2000 là hệ điều hành đa xử lý (multiprocess) 32 bit, được xây dựng để quản lý các hệ thống mạng máy tính, nó hỗ trợ cả 2 mô hình mạng: client/server (server-based) và peer-to-peer.

Windows 2000 được xây dựng dựa trên Windows NT 4.0, nó cung cấp nhiều công cụ tốt hơn để quản lý Internet và các dịch vụ trên Internet.

Windows 2000 là một họ gồm có 4 sản phẩm, một cho client và ba cho server: Client: Windows 2000 Professional; Server: Windows 2000 Server, Windows 2000 Advanced Server, Windows 2000 datacenter Server

Các sản phẩm trên khác nhau ở các điểm sau:

- Số các processor được hỗ trợ.
- Số lượng bộ nhớ vật lý được hỗ trợ.
- Số các kết nối mạng hiện tại được hỗ trợ.
- Các dịch vụ có trong các sản phẩm server không có trong sản phẩm client.

Các file chính của Windows 2000 bao gồm:

- **Ntoskrnl.exe**: Thành phần Executive và Kernel của hệ điều hành.
- **Ntkrnlpa.exe**: Thành phần Executive và Kernel với sự hỗ trợ để mở rộng bộ nhớ vật lý, nó cho phép địa chỉ hoá bộ nhớ vật lý lên đến 64GB.
- **Hal.dll**: Lớp phần cứng trừu tượng.

- **Win32k.sys**: Bộ phận kernel mode của hệ thống con Win32.
- **Ntdll.dll**: Hỗ trợ sự điều phối để thực hiện các hàm.
- **Kernel32.dll, Advapi32.dll, User32.dll, Gdi32.dll**: Các file chính của hệ thống con Win32 DLLs.

#### **I.6.6. Một số đặc tính của Windows 2000**

##### ➤ **Windows 2000 so với các Windows khác:**

- Windows 2000 hỗ trợ các hệ thống multiprocessor các windows khác không hỗ trợ điều này.
- Windows 2000 hỗ trợ hệ thống file an toàn các windows khác không có hệ thống file an toàn.
- Windows 2000 là hệ điều hành 32 bit đầy đủ, nó không chứa các mã 16 bit, nó hỗ trợ các mã khác để chạy các ứng dụng windows 16 bit. Các windows khác chứa một lượng lớn các mã 16 bit cũ từ các phiên bản trước. Đây là điểm khác biệt lớn của windows 2000 so với windows 3.1 và MS\_DOS.
- Windows 2000 cung cấp một tùy chọn để chạy các ứng dụng windows 16 bit, mà mỗi ứng dụng sở hữu một không gian địa chỉ riêng. Trong các hệ điều hành windows khác các ứng dụng windows 16 bit luôn chạy trong không gian địa chỉ bộ nhớ được chia sẻ, mà ở đó các ứng dụng có thể bị làm hỏng bởi các ứng dụng khác.
- Bộ nhớ được chia sẻ trong Windows 2000 là chỉ nhìn thấy khi các tiến trình ánh xạ đến cùng một vùng nhớ được chia sẻ, trong Win32 API một vùng bộ nhớ được chia sẻ được gọi là file ánh xạ. Trong các hệ điều hành windows khác tất cả bộ nhớ được chia sẻ là được nhìn thấy và được ghi bởi tất cả các tiến trình. Do đó bất kỳ một tiến trình nào cũng có thể ghi đến bất kỳ file ánh xạ.
- Trong các hệ điều hành windows khác một vài trang (page) quan trọng của hệ điều hành trên bộ nhớ là có thể được ghi từ user mode, vì thế một ứng dụng của người sử dụng có thể là hỏng hệ thống (ghi đè lên hệ điều hành). Điều này không xảy ra đối với hệ điều hành Windows 2000.

##### ➤ **Đặc tính của Windows 2000 Server:**

- Windows 2000 server cung cấp chế độ Safe mode, cho phép khởi tạo windows 2000 server với tập các thiết bị và dịch vụ tối thiểu nhất. Nó cũng hỗ trợ chức năng Plug-n-Play.
- Windows 2000 server hỗ trợ các ứng dụng trên nền MS\_DOS, các ứng dụng 16/32 bit trên nền Windows, ... và nó cũng đưa ra các cơ chế để bảo vệ các ứng dụng khi các ứng dụng này hoạt động trên bộ nhớ.

- Windows 2000 server cung cấp tiện ích Backup, chức năng Automated System Recorder, để backup dữ liệu, khôi phục dữ liệu khi dữ liệu tình cờ bị mất, recover và restore hệ thống trong trường hợp lỗi nghiêm trọng xảy ra trên đĩa cứng.
- Windows 2000 server cung cấp các công cụ cấp cao cho các hệ thống cần có độ ổn định và khả năng chịu đựng lỗi cao.
- Windows 2000 server cho phép quản lý 2 hệ thống đĩa: Basic disk và Dynamic Disk. Nó cũng hỗ trợ các hệ thống tập tin: FAT, NTFS, CDFS.
- Windows 2000 server hỗ trợ các hệ thống Symmetric Multiprocessing, có từ 4 đến 8 processor. Nó cũng có thể quản lý được từ 4GB đến 8GB bộ nhớ vật lý.
- Windows 2000 advanced server hỗ trợ các hệ thống Clusters. Cluster là một nhóm các server được kết nối để cùng làm việc với nhau, nếu một server trong cluster bị hỏng thì một server khác trong cùng cluster sẽ được thay thế để hoàn thành tác vụ mà server đó (server bị hỏng) đang thực hiện.
- Windows 2000 Datacenter Server hỗ trợ 32-way SMP nhờ OEM (original equipment manufacturer) và có khả năng quản lý đến 64 GB bộ nhớ vật lý. Đặc tính này giúp các server Windows 2000 trở thành các trung tâm dữ liệu (database centric), các kho dữ liệu lớn (data warehouses) phục vụ cho các Internet Service Provider và host Web site.

## **I.6.7. Một số khái niệm trong Windows 2000**

### **I.5.3.a. Tiến trình (Process) và tiểu trình (Thread)**

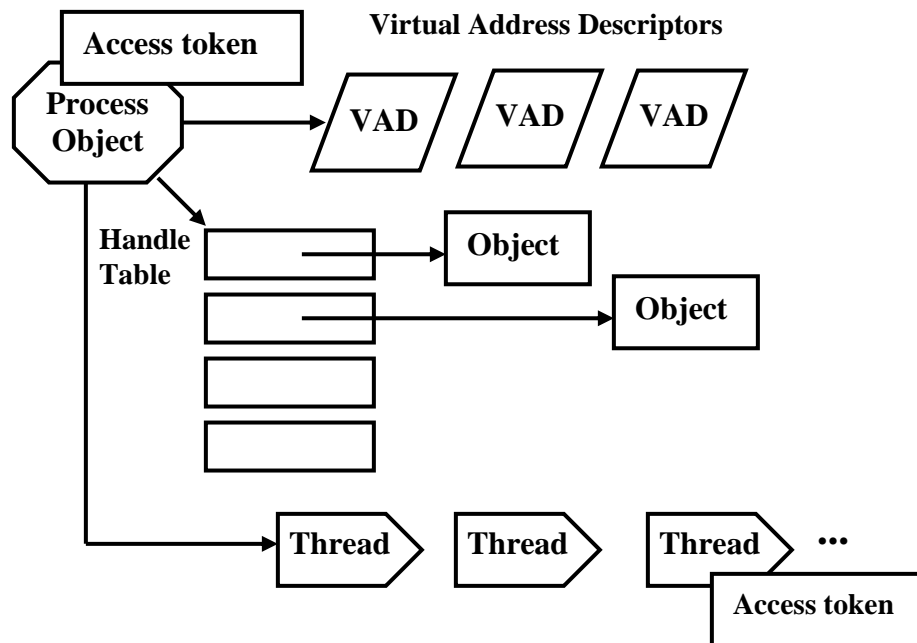
➤ **Tiến trình:** Người sử dụng khó có thể phân biệt sự khác nhau giữa chương trình và tiến trình, mặc dù nó có các sự khác nhau rất cơ bản. Một chương trình là một dãy tĩnh các chỉ thị, trong khi đó tiến trình là nơi chứa một tập các tài nguyên được sử dụng bởi các tiểu trình mà các tiểu trình này thực hiện một đoạn mã đặc biệt nào đó của chương trình. Các tiến trình của Windows 2000 bao gồm:

- **Một không gian địa chỉ ảo riêng, đó là một tập các địa chỉ bộ nhớ ảo mà các tiến trình có thể sử dụng.**
- **Một chương trình có thể thực hiện được, mà nó định rõ bất kỳ một code và data ban đầu nào và nó được ánh xạ vào không gian địa chỉ của tiến trình.**
- **Một danh sách mở các tài nguyên hệ thống khác nhau mà tiến trình sử dụng như các semaphore (sự đánh tín hiệu bằng cờ), các cổng giao tiếp tiến trình, các file, ... , các tài nguyên này được truy cập bởi tất cả các tiểu trình trong tiến trình.**
- **Một ngữ cảnh an toàn (security context), được gọi là thẻ truy cập**

(access token), nó định danh người sử dụng, các nhóm an toàn, và các cấp đặc quyền có liên quan với tiến trình.

- Một định danh duy nhất, được gọi là Process ID.
- Có ít nhất một tiểu trình.

➤ **Tiểu trình:** Một tiểu trình là một thực thể trong tiến trình mà hệ điều hành có thể lập lịch để nó thực hiện, không có nó thì các tiến trình của nó không thể thực hiện được. Một tiểu trình bao gồm các thành phần cơ bản sau:



Hình 1.10: Các tiến trình và tài nguyên của nó

- Nội dung của các thanh ghi trong CPU miêu tả trạng thái của processor.
- Hai Stack, một cho tiểu trình sử dụng khi thực hiện trong kernel mode và một cho tiểu trình sử dụng trong user mode.
- Một vùng lưu trữ riêng được gọi là TLS (thread local storage) để sử dụng bởi các hệ thống Connection, các thư viện run-time, và các DLL.
- Một định danh duy nhất, được gọi là Thread ID.
- Đôi khi các tiểu trình cũng sở hữu một ngữ cảnh an toàn riêng, nó thường được sử dụng bởi các ứng dụng server đa tiểu trình.

Các thanh ghi, các stack và các vùng lưu trữ riêng được gọi là ngữ cảnh của tiểu trình. Bởi vì các thông tin này là khác nhau cho mỗi kiến trúc máy khác nhau mà Windows 2000 chạy trên nó. Cấu trúc

ngữ cảnh này được trả về bởi hàm Win32 API *GetThreadContext*.

Mặc dù các tiểu trình có một ngữ cảnh thực hiện riêng, nhưng mỗi tiểu trình trong phạm vi một tiến trình đều chia sẻ không gian địa chỉ ảo của tiến trình, điều này có nghĩa rằng tất cả các tiểu trình trong một tiến trình có thể ghi đến hoặc đọc từ bộ nhớ của tiểu trình khác. Các tiểu trình không thể tham chiếu đến không gian địa chỉ của các tiến trình khác, trừ khi tiến trình khác đó đưa ra một phần không gian địa chỉ riêng của nó như là một phần bộ nhớ được chia sẻ.

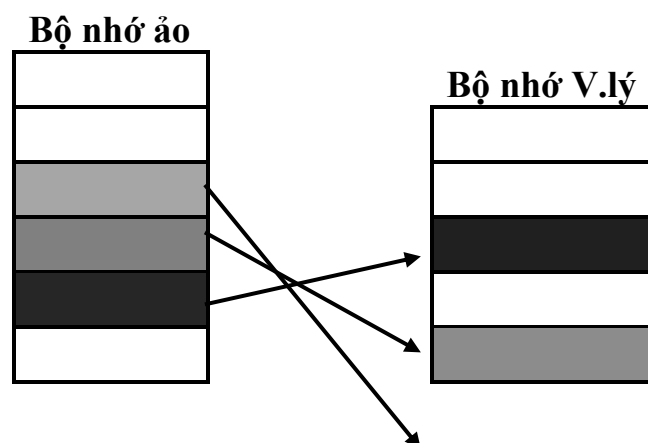
Ngoài không gian địa chỉ riêng và một hoặc nhiều tiểu trình, mỗi tiến trình còn có một định danh an toàn và một danh sách điều khiển các đối tượng như là các file, các section bộ nhớ được chia sẻ hoặc một hoặc nhiều đối tượng đồng bộ như là: các mutex, các event, các semaphore (sự đánh tín hiệu bằng cờ). Điều này được minh họa ở hình trên.

#### **I.5.3.b. Bộ nhớ ảo (Virtual Memory) trong windows 2000**

Windows 2000 cài đặt một hệ thống bộ nhớ ảo dựa trên một không gian địa chỉ 32 bit. Ba hai bit của địa chỉ ảo này chuyển thành 4GB bộ nhớ ảo. Windows 2000 dùng nửa thấp của 4GB này cấp cho các tiến trình, nửa còn lại dành riêng cho hệ điều hành, phần này được bảo vệ bởi chính hệ điều hành. Sự ánh xạ của nửa thấp thay đổi để tương ứng với tiến trình đang thực hiện, nhưng sự thay đổi của nửa cao luôn phù hợp với bộ nhớ ảo của hệ điều hành.

Nhớ lại rằng, không gian địa chỉ ảo của tiến trình là một tập các địa chỉ có sẵn cho các tiểu trình của tiến trình sử dụng. Bộ nhớ ảo cung cấp một cái nhìn logic của bộ nhớ, nhờ đó nó mở rộng được sức mạnh lưu trữ tiểu trình của bộ nhớ vật lý. Trong quá trình hoạt động của hệ thống, với sự giúp đỡ của phần cứng, trình biên dịch hoặc các ánh xạ, của trình quản lý bộ nhớ sẽ chuyển địa chỉ ảo thành địa chỉ vật lý, nơi dữ liệu được lưu trữ thực tế. Bằng cách điều khiển sự bảo vệ và sự ánh xạ, hệ điều hành có thể đảm bảo rằng một tiến trình riêng lẻ không làm hỏng các tiểu trình và không ghi đè lên dữ liệu của hệ điều hành.

Hình vẽ 1.11 sau đây cho thấy có 3 trang ảo liên kế được ánh xạ thành 3 trang không liên kế trong bộ nhớ vật lý.





**Hình 1.11:** Bộ nhớ ảo và bộ nhớ Vật lý

Đa số các hệ thống đều có bộ nhớ vật lý nhỏ hơn tổng số bộ nhớ ảo mà các tiến trình cần sử dụng khi thực hiện, 2 GB hoặc 3 GB cho mỗi tiến trình. Khi điều này xảy ra thì trình quản lý bộ nhớ sẽ di chuyển một nội dung của một vài trang bộ nhớ ra đĩa, để lấy không gian trang trống này sử dụng cho các tiến trình khác hoặc cho chính hệ điều hành. Khi một tiến trình truy cập đến một trang địa chỉ ảo mà nội dung của trang này đã bị đưa ra đĩa thì bộ phận quản lý bộ nhớ ảo sẽ nạp thông tin này trở lại bộ nhớ từ đĩa. Các ứng dụng không cần thay đổi bất kỳ một điều gì để phù hợp với sự phân trang, bởi vì phần cứng đã hỗ trợ để cho phép trình quản lý bộ nhớ thực hiện sự phân trang mà không cần hiểu biết hoặc sự trợ giúp của các tiến trình hoặc các tiểu trình.

### **I.5.3.c. Đa xử lý đối xứng (SMP: Symmetric Multiprocessing)**

Đa tác vụ (multitasking) là một kỹ thuật của hệ điều hành dùng để chia sẻ một processor đơn cho nhiều tiểu trình đang thực hiện. Khi máy tính có nhiều hơn một processor thì nó có thể thực hiện hai tiểu trình đồng thời. Nhưng ngược lại hệ điều hành đa tác vụ chỉ có vẻ như thực hiện đa tiểu trình tại cùng một thời điểm, hệ điều hành đa xử lý thực tế làm được điều đó, thực hiện một tiểu trình trên mỗi processor của nó.

Một trong những mục tiêu thiết kế của hệ điều hành Windows NT là làm cho NT chạy tốt trên các hệ thống máy tính multiprocessor. Windows 2000 cũng là hệ điều hành SMP. Nó không có processor master, hệ điều hành cũng như các tiểu trình của người sử dụng đều có thể được chia sẻ trên bất kỳ một processor nào. Ngoài ra, tất cả các processor cũng chỉ chia sẻ một không gian bộ nhớ riêng. Đây là mô hình tương phản với mô hình đa xử lý bất đối xứng (ASMP: asymmetric multiprocessor), trong mô hình này hệ điều hành chạy trên một processor riêng, các processor còn lại chỉ dùng để chạy các tiểu trình của người sử dụng.

Số processor sử dụng phụ thuộc vào phiên bản Windows 2000 được sử dụng. Con số này được lưu trữ trong Registry tại khoá:

HKLM\SYSTEM\CurrentControlSet\Control\Session\Manager\LicensedProcessor

Để chạy tốt trên một hệ thống SMP thì hệ điều hành Windows 2000 phải được thiết kế sao cho nó phải tuân thủ một cách nghiêm ngặt các nguyên tắc sau đây, đó là các nguyên tắc của một hệ điều hành Multiprocessor:

- Có khả năng chạy mã của hệ điều hành trên bất kỳ một processor có sẵn nào và chạy được trên multiprocessor tại cùng một thời điểm.



- Đa tiểu trình được thực hiện trong một tiến trình đơn, mỗi tiểu trình có thể thực hiện đồng thời trên các processor khác nhau.
- Cho phép các thành phần khác nhau của hệ thống như device driver, server process chạy tốt trên hệ thống multiprocessor.

### **I.6.8. Kiến trúc của Windows 2000**

#### ➤ **Kernel Mode & User Mode**

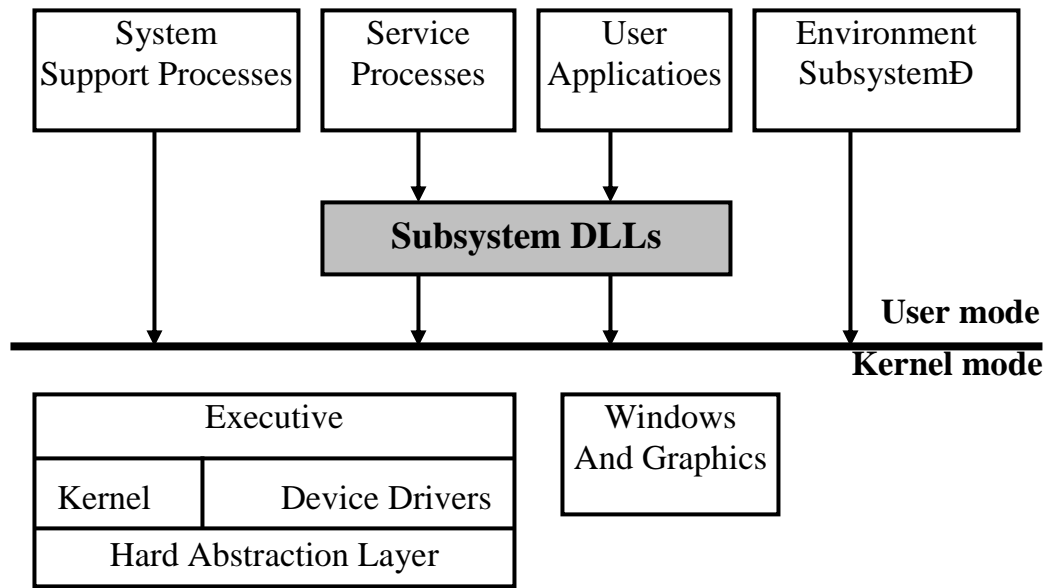
Để bảo vệ hệ điều hành tránh sự truy cập và/hoặc thay đổi bất hợp lệ của các chương trình ứng dụng của người sử dụng, Windows 2000 sử dụng hai chế độ xử lý truy cập: **Kernel mode** và **User mode**. Các chương trình ứng dụng của người sử dụng chạy trong user mode, trong khi đó các dịch vụ hệ thống và các chương trình điều khiển thiết bị của hệ điều hành chạy trong kernel mode. Kernel mode chỉ đến một chế độ của việc thực hiện trong processor mà ở đó nó có toàn quyền truy cập đến tất cả hệ thống bộ nhớ và tất cả các chỉ thị của CPU. Trong cấu trúc này phần mềm hệ điều hành được cung cấp một mức đặc quyền cao hơn so với mức đặc quyền của các chương trình ứng dụng của người sử dụng. Processor cung cấp các cơ sở cần thiết để người thiết kế hệ điều hành đảm bảo rằng các ứng dụng không thể phá vỡ trạng thái ổn định của hệ thống và làm hỏng nó.

Các tiểu trình trong user mode thực hiện trong không gian địa chỉ bộ nhớ được bảo vệ, mỗi thành phần trong user mode sở hữu một không gian địa chỉ tiến trình riêng. Trong khi đó Windows 2000 không cung cấp bất kỳ một sự bảo vệ nào trên các không gian bộ nhớ riêng được sử dụng bởi các thành phần chạy trong kernel mode. Trong một tuyên bố khác, trong kernel mode, mã hệ điều hành và các chương trình điều khiển thiết bị hoàn toàn có thể truy cập đến không gian bộ nhớ hệ thống và có thể vượt qua sự giám sát an toàn của Windows 2000 để truy cập đến các đối tượng. Bởi vì phần lớn mã của hệ điều hành Windows 2000 chạy trong kernel mode, các thành phần quan trọng nhất của hệ điều hành chạy trong kernel mode được thiết kế và được kiểm tra rất cẩn thận để đảm bảo rằng nó không vi phạm đến sự an toàn của hệ thống.

**Chú ý:** Kiến trúc của processor Intel x86 định nghĩa 4 cấp/ vòng đặc quyền truy cập (Privilege levels/ Rings), để bảo vệ code và data của hệ thống, tránh sự ghi đè (overwrite) có chủ ý (maliciously) hoặc không chủ ý (inadvertently) bởi các code có cấp đặc quyền truy cập thấp hơn. Windows 2000 sử dụng cấp 0/ vòng 0 cho **Kernl mode** và cấp 3/ vòng 3 cho **Uer mode**. Nguyên nhân mà Windows 2000 chỉ sử dụng có 2 cấp là do một vài kiến trúc phần cứng trước đó, chẳng hạn như Compaq Alpha và Silicon Graphics, chỉ được cài đặt 2 cấp đặc quyền truy cập.

#### ➤ **Kiến trúc của Windows 2000**

Hình vẽ 1.12 sau đây cho ta thấy kiến trúc đã được đơn giản hoá của Windows 2000.



**Hình 1.12:** Kiến trúc được đơn giản của Windows 2000

Hình vẽ cho ta thấy kiến trúc của hệ điều hành Windows 2000 được chia thành hai phần: User mode và Kernel mode. User mode bao gồm các thành phần: System support processes, Service Processes, User applications, và Environment subsystems, mỗi thành phần này sở hữu một không gian địa chỉ tiến trình riêng.

□ **Các thành phần trong User mode:**

- *System support processes* (các tiến trình hỗ trợ hệ thống): Như là tiến trình login, quản lý các Session, các thành phần này không phải là các dịch vụ của Windows 2000, do đó nó không được khởi động bởi thành phần *Service Control Manager*.

*Service processes* (các tiến trình dịch vụ): Đó là các dịch vụ chủ Win32, như là dịch Task Scheduler và Spooler, và cũng có thể là các ứng dụng server Windows 2000 như là Microsoft SQL Server, Exchange Server và các thành phần chạy như là các dịch vụ.

- *User applications* (các ứng dụng người sử dụng): Nó có thể là một trong năm loại sau: Win32, Windows 3.1, MS\_DOS, POSIX, hoặc OS/2 1.2.

- *Environment subsystems* (các hệ thống con môi trường): nó đưa ra các dịch vụ nguyên thủy của hệ điều hành, các ứng dụng của người sử dụng thông qua một tập các hàm có thể gọi được, do đó nó cung cấp một môi trường hệ điều hành cho các ứng dụng. Windows 2000 đưa ra ba hệ thống con môi trường: Win32, POSIX và OS/2, trong đó Win32 là hệ thống con đặc biệt nhất, Windows 2000 không thể chạy nếu không có nó, do đó nó phải

luôn ở trạng thái chạy ngay sau khi hệ thống được khởi động. POSIX và OS/2 được cấu hình là chỉ khởi tạo khi cần. Các ứng dụng được viết trên các hệ điều hành khác nhau có thể chạy trên Windows 2000 nhờ sử dụng các environment subsystem.

- *Subsystem DLLs* (hệ thống con các thư viện liên kết động): Hình trên cho thấy trong Windows 2000 các ứng dụng của người sử dụng không thể gọi trực tiếp các dịch vụ nguyên thủy của hệ điều hành, mà chúng phải thông qua một hoặc nhiều các DLL. Vai trò của các Subsystem DLL là chuyển các yêu cầu gọi hàm vào bên trong các dịch vụ hệ thống của Windows 2000.

□ **Các thành phần trong Kernel mode:**

- *Windows 2000 Executive*: Chứa các dịch vụ cơ sở của hệ điều hành, như là: quản lý bộ nhớ, quản lý các tiến trình và tiểu trình, quản lý sự an toàn hệ thống, quản lý I/O, và thực hiện việc truyền thông liên tiến trình.

- *Windows 2000 Kernel*: Bao gồm các chức năng cấp thấp của hệ điều hành như là: lập lịch tiểu trình, đồng bộ cho các hệ thống multiprocessor. Nó cũng cung cấp một tập các thường trình và các đối tượng cơ sở mà Executive sử dụng để cài đặt các chức năng cấp cao.

- *Device drivers* (các trình điều khiển thiết bị): Bao gồm cả hai: điều khiển thiết bị phần cứng và điều khiển hệ thống file và mạng. Điều khiển thiết bị phần cứng có nhiệm vụ chuyển các lời gọi hàm I/O từ phía người sử dụng thành các yêu cầu I/O thiết bị phần cứng cụ thể.

- *HAL: Hardware Abstraction Layer* (lớp phần cứng trừu tượng): Lớp này làm trừu tượng hoá các chi tiết phần cứng bên trong của PC, làm cho Windows 2000 Server tương thích với nhiều kiến trúc phần cứng khác nhau. Nó cho phép Windows 2000 chạy trên các nền vi xử lý khác nhau như Intel và Alpha, mà không cần duy trì 2 version khác của Windows 2000 Execute. HAL bảo vệ tất cả phần cứng và hỗ trợ các nền cụ thể cần cho mỗi thành phần trong hệ thống đối với tất cả phần cứng và hỗ trợ nền cụ thể. HAL được cài đặt như là một DLL và đóng vai trò như là giao diện giữa các thành phần phần cứng và phần mềm.

- *Window Manager and Graphical Device Interface (GDI)*: Window Manager và GDI được sử dụng để quản lý hệ thống hiển thị. Window Manager bảo vệ màn hình và nhận các lệnh từ các thiết bị nhập như là Mouse hoặc bàn phím. GDI điều khiển việc vẽ và thực hiện các thao tác đồ hoạ với sự giúp đỡ của các chức năng khác nhau được định nghĩa trước.

Sau đây chúng ta sẽ tìm hiểu rõ hơn về một số thành phần trong kiến trúc của hệ điều hành Windows 2000:

□ **Environment Subsystem và Subsystem DLL:**

Vai trò của hệ thống con môi trường là đưa ra một vài tập con cơ sở các dịch vụ hệ thống trong Windows 2000 executive cho các ứng dụng. Mỗi hệ thống con có thể cung cấp truy cập đến các tập con khác nhau của các dịch vụ nguyên thủy của Windows 2000. Từ một ứng dụng được xây dựng trên một hệ thống con này không thể gọi đến một ứng dụng được xây dựng trên một hệ thống con khác.

Các lời gọi hàm không thể lẫn lộn giữa các hệ thống con. Tức là một ứng dụng trên POSIX chỉ có thể gọi các dịch vụ được đưa ra bởi hệ thống con POSIX, và một ứng dụng Win32 chỉ có thể gọi các dịch vụ được đưa ra bởi hệ thống con Win32.

Như đã biết các ứng dụng người sử dụng không thể gọi trực tiếp các dịch vụ hệ thống của Windows 2000 mà phải thông qua một hoặc nhiều các hệ thống con DLL. Các hệ thống con DLL Win32 như kernel32.dll, Advapi32.dll, User32.dll và Gdi32.dll, cài đặt các hàm Win32 API, để các ứng dụng của người sử dụng gọi nó thông qua tập hàm này. Khi một ứng dụng gọi một hàm trong hệ thống con DLL, thì một trong ba trường hợp sau sẽ xảy ra:

- Hàm được cài đặt hoàn toàn trong hệ thống con DLL. Nói cách khác là không có thông điệp gửi tới tiến trình Vai trò của hệ thống con môi trường, và không có một dịch vụ hệ thống nào trong Windows 2000 executive nào được gọi. Hàm được thực hiện trong user mode và kết quả được trả về cho chương trình gọi.
- Hàm yêu cầu một hoặc nhiều lời gọi đến Windows 2000 executive. Ví dụ khi các hàm Win32 ReadFile và WriteFile được gọi thì nó phải gọi đến các dịch vụ hệ thống I/O NtReadFile và NtWriteFile trong Windows 2000.
- Hàm yêu cầu một vài công việc để thực hiện trong tiến trình, của hệ thống con môi trường. Các tiến trình của hệ thống con môi trường chạy trong user mode, chịu trách nhiệm duy trì trạng thái của các ứng dụng client chạy dưới sự điều khiển của nó. Trong trường hợp này một client/server yêu cầu tạo một hệ thống con môi trường qua một thông điệp gửi tới một hệ thống con để thực hiện một vài thao tác. Hệ thống con DLL thì đợi trả lời trước khi trả về cho ứng dụng gọi.

Một vài hàm có thể kết hợp 2 trong 3 trường trên, như các hàm Win32: CreateProcess và CreateThread.

Tập tin Ntdll.Dll là một hệ thống đặc biệt, nó hỗ trợ thư viện chính cho việc sử dụng các hệ thống con DLL. Nó chứa hai loại hàm sau:

- Dịch vụ hệ thống gửi đến các dịch vụ hệ thống Windows 2000 executive.
- Các hàm hỗ trợ bên trong được sử dụng bởi các hệ thống con, các hệ

thông con DLL và các hàm nguyên thủy điển hình khác.

Các hàm ở nhóm đầu tiên cung cấp một giao diện để Windows 2000 executive có thể được gọi từ user mode. Có hơn 200 hàm như thế và các hàm này có thể truy cập thông qua Win32 API.

Ntdll cũng chứa các hàm hỗ trợ như là image loader, heap manager và các hàm truyền thông tiến trình Win32.

#### □ **Executive:**

Windows 2000 executive là lớp trên của Ntoskrnl.exe (kernel là lớp thấp). Executive bao gồm các hàm sau:

- Các hàm được đưa ra và có thể gọi từ user mode. Đây là các hàm được gọi và được đưa ra qua Ntdll. Hầu hết các dịch vụ là được truy cập thông qua các hàm Win32 API hoặc các API của các Vai trò của hệ thống con môi trường khác.
- Các hàm chỉ có thể được gọi từ kernel mode, nó được đưa ra và được cung cấp in Windows 2000 DDK Windows 2000 Installable File System (IFS) Kit.
- Các hàm được đưa ra và có thể gọi từ kernel mode nhưng không được giới thiệu trong Windows 2000 DDK và IFS Kit.
- Các hàm được định nghĩa nhưng không được đưa ra. Đây là các hàm hỗ trợ bên trong, nó được gọi trong phạm vi Ntoskrnl.

Windows 2000 Executive chứa các thành phần quan trọng sau đây:

- Configuration Manager (quản lý cấu hình): chịu trách nhiệm cài đặt và quản lý Registry hệ thống.
- I/O Manager (quản lý I/O): Thành phần này chuyển các lệnh đọc/ ghi trong user mode đến việc đọc/ghi của IRP (I/O Request Packets). Nó gồm có: các hệ thống file, các điều khiển thiết bị, quản lý bộ nhớ cache, quản lý bộ nhớ ảo.
- InterProcess Communication - IPC Manager (quản lý truyền thông liên tiến trình): Quản lý IPC là tạo liên kết giữa client và server. Environment subsystem đóng vai trò như là một client và Executive đóng vai trò như là một server. Nó được tạo ra từ 2 thành phần: Remote Procedure Call - RPC: giữ thông tin về kết nối giữa các client và các server trên các máy tính khác nhau. Local Procedure Call - RPC: giữ thông tin về kết nối giữa các client và các server trên cùng một máy tính.
- Security Manager (quản lý sự an toàn): Đây là thành phần tạo nên sự an toàn hệ thống bằng cách bắt buộc các chính sách an toàn trên các máy tính cục bộ.

- Plug and Play Manager (quản lý plug and play): Plug and play theo dõi các hoạt động tại thời điểm Boot của các thiết bị plug and play và nó tương tác với HAL, các điều khiển thiết bị và Executive. Nó xác định các điều khiển bus thực hiện việc cấu hình và đếm như thế nào. Nó cũng xác định khi nào thì các điều khiển thiết bị được thêm vào hoặc khi nào thì khởi tạo một thiết bị.
- Process and Thread Manager (quản lý tiến trình và tiểu trình): Tạo và kết thúc các tiến trình và tiểu trình. Hỗ trợ các tiến trình và tiểu trình được cài đặt bên trong Windows 2000 kernel.
- Và một số thành phần khác như: Power manager (quản lý nguồn); Cache manager (quản lý cache); Virtual memory manager (quản lý bộ nhớ ảo), ...
- Ngoài ra executive còn chứa bốn nhóm chính các hàm hỗ trợ mà nó được sử dụng bởi chỉ các thành phần executive được liệt kê. Sau đây là bốn loại hàm hỗ trợ:
  - Object Manager (quản lý đối tượng): Tạo, quản lý, xoá các đối tượng Windows 2000 executive và các loại dữ liệu trừu tượng mà nó được sử dụng để chỉ đến các tài nguyên của Windows 2000 như: các tiến trình, các tiểu trình, và các đối tượng đồng bộ khác.
  - LPC facility: Chuyển thông điệp giữa các tiến trình client và các tiến trình server trên cùng máy tính. LPC có tính mềm dẻo, và là version được tối ưu của Remote Function Call (RPC).
  - Một tập các hàm thư viện run-time như là: xử lý string, thực hiện các phép tính, chuyển đổi các kiểu dữ liệu và xử lý các cấu trúc an toàn.
  - Executive support routine: như là cấp phát bộ nhớ hệ thống, khoá truy cập bộ nhớ và các đối tượng đồng bộ.

#### □ **Kernel:**

Kernel bao gồm một tập các hàm trong Ntoskrnl.exe mà nó cung cấp các kỹ thuật cơ bản, như điều phối tiểu trình và đồng bộ các dịch vụ, được sử dụng bởi các thành phần executive, cũng như hỗ trợ cho các kiến trúc phần cứng cấp thấp trên các kiến trúc processor khác nhau. Đa số các mã của kernel được viết bằng C, một số ít thành phần quan trọng can thiệp sâu vào phần cứng được viết bằng assembly. Một số các hàm của kernel được đưa ra trong DDK, đây là các thành phần cần thiết cho việc cài đặt các trình điều khiển thiết bị.

#### □ **Hardware Abstraction Layer (HAL):**

Như đã biết một trong những mục tiêu thiết kế của Windows 2000 là làm cho nó dễ

dường tương thích trên các nền phần cứng khác nhau. HAL là thành phần chủ chốt có thể tạo nên sự tương thích này. HAL là một modul kernel mode có thể được nạp (Hal.dll) mà nó có thể cung cấp một giao diện cấp thấp để Windows 2000 có thể chạy trên các nền phần cứng khác nhau. HAL làm ẩn các chi tiết phần cứng, như: các giao diện I/O, các điều khiển ngắt và các cơ chế truyền thông giữa các processor trong hệ thống multiprocessor, với bất kỳ một hàm nào trong cả các kiến trúc cụ thể và các máy phụ thuộc.

Trong Windows 2000 có nhiều tập tin Hal\*.dll, mỗi tập tin hỗ trợ cho một hệ thống máy tính nào đó. Hal.dll hỗ trợ cho các PC chuẩn, Halmps.dll hỗ trợ cho các PC Multiprocessor, ...

#### □ **Device Drivers:**

Các Device Driver (\*.sys) là các modul kernel, nó là giao diện giữa thành phần quản lý I/O và các phần cứng có liên quan. Các device driver không thao tác trực tiếp trên phần cứng, nó chỉ gọi các hàm trong HAL để giao tiếp với phần cứng. Windows 2000 có các loại Device Driver sau đây:

- Các hardware device driver thao tác phần cứng, sử dụng HAL, để ghi/đọc trên các thiết bị vật lý hoặc mạng. Loại này bao gồm: các điều khiển bus, các điều khiển thiết bị giao tiếp với người sử dụng, các điều khiển thiết bị lưu trữ khối, ...
- Các file system driver là các điều khiển mà Windows 2000 dùng nó để truy cập các file trong hệ thống.
- ...

#### □ **Kernel Mode Drivers:**

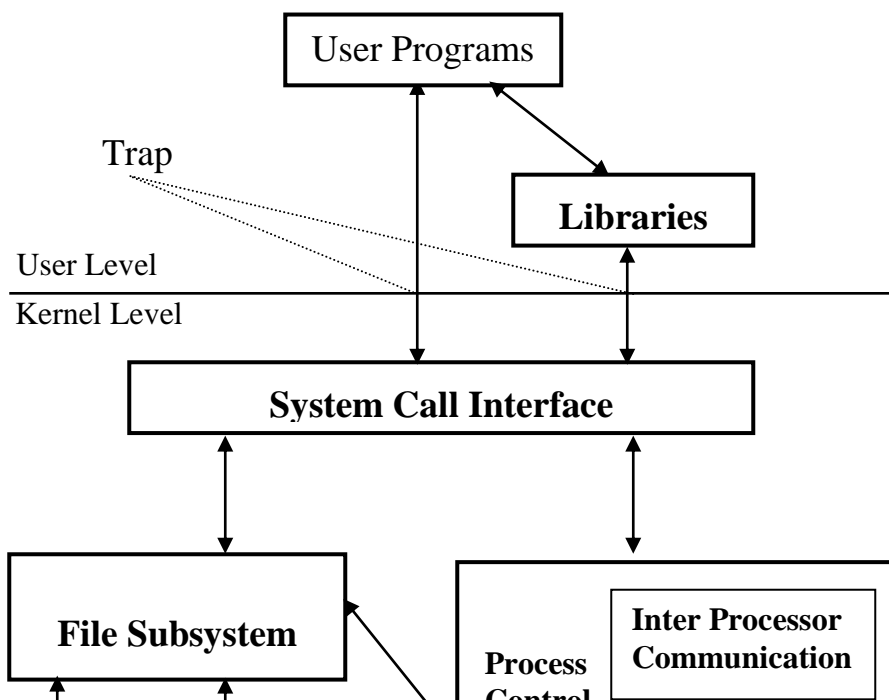
Kernel Mode Drivers cũng được cài đặt như là một thành phần, nó chứa tất cả các chức năng cần thiết. Nó gồm có: WMD (Windows Drive Model) driver, cùng với nhiều driver để hỗ trợ cho các yêu cầu của các thiết bị cụ thể.

## **I.14. Hệ điều hành Linux**

Linux là hệ điều hành miễn phí được xây dựng từ hệ điều hành Unix. Nó được phát triển bởi Linus Torvalds, một sinh viên của trường Đại học Hensinki. Linus chỉ chịu trách nhiệm tạo ra hệ thống kernel. Kernel là phần lõi của hệ điều hành, nó chịu trách nhiệm thực hiện các tác vụ của hệ thống. Linux bây giờ như một tập các phần mềm mà trong đó bao gồm kernel và các thành phần khác để nó trở thành một hệ điều hành hoàn chỉnh. Một trong những nguyên nhân làm cho Linux được nhiều người biết đến là nó được cung cấp miễn phí với mã nguồn mở.

Hình vẽ 1.13 dưới đây cho thấy cấu trúc của hệ điều hành Unix. Hình vẽ cho thấy hệ điều hành Linux được chia thành 2 cấp: User Level (cấp người sử dụng) và Kernel Level (cấp lõi).

- Kernel là cấp đặc quyền, ở đây không có giới hạn nào đối với kernel của hệ thống. Kernel có thể sử dụng tất cả các lệnh của vi xử lý, điều khiển toàn bộ bộ nhớ và truyền thông trực tiếp đến tất cả các thiết bị ngoại vi.
- User là cấp không có đặc quyền, tất cả các chương trình của người sử dụng phải hoạt động ở cấp này. Ở đây các tiến trình không thể thực hiện tất cả các lệnh của vi xử lý, không thể truy cập trực tiếp vào hệ thống phần cứng và nó chỉ được quyền sử dụng không gian nhớ đã được cấp phát. Các tiến trình ở đây chỉ có thể thực hiện các thao tác trong môi trường của riêng nó mà không làm ảnh hưởng đến các tiến trình khác và nó có thể bị ngắt bất cứ lúc nào. Các tiến trình hoạt động trong User Level không thể truy cập trực tiếp tài nguyên của hệ thống mà nó phải thông qua giao diện lời gọi hệ thống (System call Interface). Một lời gọi hệ thống là một yêu cầu được gửi từ tiến trình của chương trình người sử dụng đến Kernel, Kernel sẽ xử lý yêu cầu trong chế độ kernel sau đó trả kết quả về lại cho tiến trình để tiến trình tiếp tục thực hiện.





➤ **Sau đây là một vài đặc điểm của Linux:**

- **Miễn phí (Free):** Linux là một hệ điều hành được cung cấp miễn phí trên Internet, chúng ta không phải trả bất kỳ một chi phí nào cho việc download nó. Linux được cung cấp cùng với các phần mềm chạy trên nó.
- **Mã nguồn mở (Open Source):** Điều này có nghĩa người sử dụng không chỉ sử dụng hệ điều hành và thực hiện các chương trình mà còn có thể xem và sửa đổi mã nguồn của nó, để phát triển nó theo từng mục đích cụ thể của người sử dụng.
- **Yêu cầu phần cứng (Hardware):** Linux có thể chạy trên hầu hết các phần cứng hiện có, nó có thể hoạt động trên các vi xử lý: 386, 486, Pentium MMX, Pentium II, Sparc, Dec Alpha hoặc Motorola 68000.
- **Đa tác vụ (Multi-Tasking):** Linux là hệ điều hành đa tác vụ, tức là một người sử dụng có thể chạy nhiều chương trình tại cùng một thời điểm. Mỗi tác vụ là một tiến trình. Theo cách này người sử dụng không cần phải đợi cho

một tiến trình kế thúc hợp lệ để khởi động một tiến trình khác.

- Đa người sử dụng (Multi-User): Điều này có nghĩa có nhiều hơn một người sử dụng có thể sử dụng hệ thống tại cùng một thời điểm. Khái niệm multi user xuất phát trực tiếp từ khía cạnh multi-tasking. Hệ thống có thể điều khiển nhiều hơn một người sử dụng tại cùng một thời điểm giống như cách mà nó điều khiển nhiều hơn một công việc.
- Hỗ trợ đa vi xử lý (Multi Processor Support): Linux có thể điều hành các hệ thống máy tính có nhiều hơn một vi xử lý.
- Máy chủ web (Web Server): Linux có thể được sử dụng để chạy như là một web server, và đáp ứng các giao thức ứng dụng như là HTTP hoặc FTP.
- Hỗ trợ mạng TCP/IP (TCP/IP Networking Support): Hỗ trợ mạng TCP/IP được xây dựng trong chính kernel của Linux. Linux một trong các hệ điều hành mạng tốt nhất. Nó bao gồm các chương trình như là: Telnet, Ftp, Rlogin, Rsh và nhiều chương trình khác.
- Hỗ trợ lập trình (Programming Support): Linux cung cấp hỗ trợ lập trình cho Fortran, C, C++, Tcl/Tk, Perl và nhiều ngôn ngữ lập trình khác.
- Độ an toàn cao (High Level Security): Một trong những thuận lợi chính của Linux đó là nó cung cấp một sự an toàn cao cấp bằng cách sử dụng sự xác thực người sử dụng. Nó cũng lưu trữ password trong dạng thức được mã hoá, password một khi đã được mã hoá thì không thể giải mã. Linux cũng bao gồm hệ thống file an toàn, nó được mở rộng từ hệ thống file đang tồn tại.

## Chương II

# QUẢN LÝ TIẾN TRÌNH

---

*Tất cả các hệ điều hành đa chương, từ các hệ điều hành đơn người sử dụng đến các hệ điều hành có thể hỗ trợ đến hàng ngàn người sử dụng, đều phải xây dựng dựa trên khái niệm tiến trình. Vì thế, một yêu cầu quan trọng trong thiết kế hệ điều hành là thành phần quản lý tiến trình của hệ điều hành phải đáp ứng tất cả những gì liên quan đến tiến trình:*

- *Hệ điều hành phải cho phép thực hiện nhiều tiến trình đồng thời để khai thác tối đa thời gian xử lý của processor nhưng cũng cung cấp được thời gian hồi đáp hợp lý.*

- *Hệ điều hành phải cấp phát tài nguyên để tiến trình hoạt động một cách hiệu quả với một chính sách hợp lý nhưng không xảy ra tình trạng tắc nghẽn trong hệ thống.*

- *Hệ điều hành có thể được yêu cầu để hỗ trợ truyền thông liên tiến trình và người sử dụng tạo ra tiến trình.*

*Hệ điều hành phải có nhiệm vụ tạo ra tiến trình, điều khiển sự hoạt động của tiến trình và kết thúc tiến trình.*

*Một số hệ điều hành phân biệt hai khái niệm tiến trình và tiểu trình. Tiến trình liên quan đến quyền sở hữu tài nguyên, tiểu trình liên quan đến sự thực hiện chương trình.*

*Trong các hệ điều hành đa chương, có nhiều tiến trình tồn tại trên bộ nhớ chính, các tiến trình này luân phiên giữa hai trạng thái: sử dụng processor và đợi thực hiện vào/ra hay một vài sự kiện nào đó xảy ra.*

*Tất cả những vấn đề trên sẽ được làm sáng tỏ trong chương này.*

## **I.15. Tổng quan về tiến trình**

### **I.1.9. Tiến trình và các loại tiến trình**

➤ **Tiến trình (process):** Trong chương I chúng ta đã có khái niệm về tiến trình: *Tiến trình là một bộ phận của một chương trình đang thực hiện, đơn vị thực hiện tiến trình là processor.* Ở đây chúng tôi nhấn mạnh thêm rằng: Vì tiến trình là một bộ phận của chương trình nên tương tự như chương trình tiến trình cũng sở hữu một con trỏ lệnh, một con trỏ stack, một tập các thanh ghi, một không gian địa chỉ trong bộ nhớ chính và tất cả các thông tin cần thiết khác để tiến trình có thể hoạt động được.

Khái niệm trên đây mang tính trực quan, để thấy được bản chất của tiến trình các chuyên gia về hệ điều hành đã đưa ra nhiều định nghĩa khác nhau về tiến trình, ở đây chúng tôi nêu ra hai định nghĩa để các bạn tham khảo. Định nghĩa của Saltzer: *Tiến trình là một chương trình do một processor logic thực hiện.* Định nghĩa của Horning & Rendell: *Tiến trình là một quá trình chuyển từ trạng thái này sang trạng thái khác dưới tác động của hàm hành động, xuất phát từ một trạng thái ban đầu nào đó.*

Định nghĩa của Saltzer cho thấy, trên góc độ thực hiện thì tiến trình hoàn toàn tương tự chương trình, chỉ khác ở chỗ: tiến trình do processor logic chứ không phải processor vật lý thực hiện. Điều này sẽ được làm sáng tỏ trong phần mô tả về tiến trình sau đây. Định nghĩa của Horning & Rendell cho thấy trong quá trình hoạt

động của tiến trình là quá trình chuyển từ trạng thái này sang trạng thái khác nhưng sự chuyển đổi này không phải do chính bản thân tiến trình mà là do sự tác động từ bên ngoài, cụ thể ở đây là bộ phận điều phối tiến trình của hệ điều hành. Điều này sẽ được làm sáng tỏ trong phần mô tả về các trạng thái tiến trình sau đây.

➤ **Các loại tiến trình:** Các tiến trình trong hệ thống có thể chia thành hai loại: tiến trình tuần tự và tiến trình song song. Tiến trình tuần tự là các tiến trình mà điểm khởi tạo của nó là điểm kết thúc của tiến trình trước đó. Tiến trình song song là các tiến trình mà điểm khởi tạo của tiến trình này nằm ở thân của các tiến trình khác, tức là có thể khởi tạo một tiến trình mới khi các tiến trình trước đó chưa kết thúc. Tiến trình song song được chia thành nhiều loại:

- Tiến trình song song độc lập: là các tiến trình hoạt động song song nhưng không có quan hệ thông tin với nhau, trong trường hợp này hệ điều hành phải thiết lập cơ chế bảo vệ dữ liệu của các tiến trình, và cấp phát tài nguyên cho các tiến trình một cách hợp lý.

- Tiến trình song song có quan hệ thông tin: trong quá trình hoạt động các tiến trình thường trao đổi thông tin với nhau, trong một số trường hợp tiến trình gửi thông báo cần phải nhận được tín hiệu từ tiến trình nhận để tiếp tục, điều này dễ dẫn đến bế tắc khi tiến trình nhận tín hiệu không ở trong trạng thái nhận hay tiến trình gửi không ở trong trạng thái nhận thông báo trả lời.

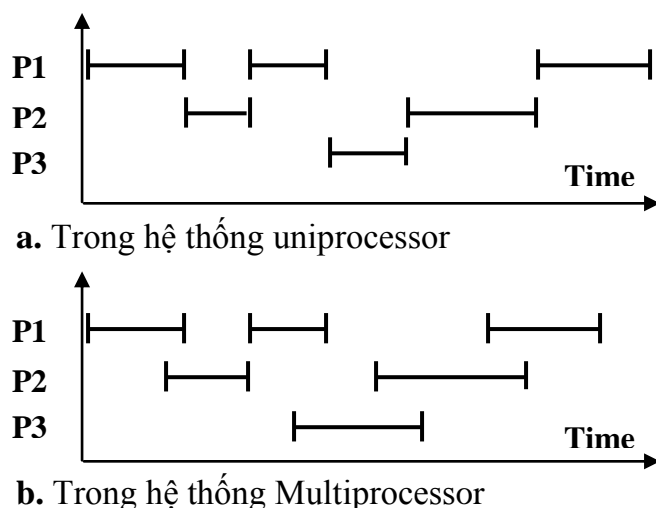
- Tiến trình song song phân cấp: Trong quá trình hoạt động một tiến trình có thể khởi tạo các tiến trình khác hoạt động song song với nó, tiến trình khởi tạo được gọi là tiến trình cha, tiến trình được tạo gọi là tiến trình con. Trong mô hình này hệ điều hành phải giải quyết vấn đề cấp phát tài nguyên cho các tiến trình con. Tiến trình con nhận tài nguyên ở đâu, từ tiến trình cha hay từ hệ thống. Để giải quyết vấn đề này hệ điều hành đưa ra 2 mô hình quản lý tài nguyên: Thứ nhất, mô hình tập trung, trong mô hình này hệ điều hành chịu trách nhiệm phân phối tài nguyên cho tất cả các tiến trình trong hệ thống. Thứ hai, mô hình phân tán, trong mô hình này hệ điều hành cho phép tiến trình con nhận tài nguyên từ tiến trình cha, tức là tiến trình khởi tạo có nhiệm vụ nhận tài nguyên từ hệ điều hành để cấp phát cho các tiến trình mà nó tạo ra, và nó có nhiệm vụ thu hồi lại tài nguyên đã cấp phát trả về cho hệ điều hành trước khi kết thúc.

- Tiến trình song song đồng mức: là các tiến trình hoạt động song song sử dụng chung tài nguyên theo nguyên tắc luân lượt, mỗi tiến trình sau một khoảng thời gian chiếm giữ tài nguyên phải tự động trả lại tài nguyên cho tiến trình kia.

*Các tiến trình tuần tự chỉ xuất hiện trong các hệ điều hành đơn nhiệm đa chương, như hệ điều hành MS-DOS, loại tiến trình này tồn tại nhiều hạn chế, điển hình nhất là không khai thác tối đa thời gian xử lý của processor. Các tiến trình song song xuất hiện trong các hệ điều hành đa nhiệm đa chương, trên cả hệ thống uniprocessor và multiprocessor. Nhưng sự song song thực, chỉ có ở các hệ thống*

*multiprocessor, trong hệ thống này mỗi processor chịu trách nhiệm thực hiện một tiến trình. Sự song song trên các hệ thống uniprocessor là sự song song giả, các tiến trình song song trên hệ thống này thực chất là các tiến trình thay nhau sử dụng processor, tiến trình này đang chạy thì có thể dừng lại để nhường processor cho tiến trình khác chạy và sẽ tiếp tục lại sau đó khi có được processor. Đây là trường hợp mà ở trên ta cho rằng: điểm khởi tạo của tiến trình này nằm ở thân của tiến trình khác.*

Hình vẽ sau đây minh họa sự khác nhau, về mặt thực hiện, giữa các tiến trình song song/ đồng thời trong hệ thống uniprocessor với các tiến trình song song/ đồng thời trong hệ thống multiprocessor.



**Hình 2.1:** Sự thực hiện đồng thời của các tiến trình trong hệ thống uniprocessor (a) và hệ thống multiprocessor (b).

Trong tài liệu này chúng ta chỉ khảo sát sự hoạt động của các tiến trình song song (hay đồng thời) trên các hệ thống uniprocessor.

Đối với người sử dụng thì trong hệ thống chỉ có hai nhóm tiến trình. Thứ nhất, là các tiến trình của hệ điều hành. Thứ hai, là các tiến trình của chương trình người sử dụng. Các tiến trình của hệ điều hành hoạt động trong chế độ đặc quyền, nhờ đó mà nó có thể truy xuất vào các vùng dữ liệu được bảo vệ của hệ thống. Trong khi đó các tiến trình của chương trình người sử dụng hoạt động trong chế độ không đặc quyền, nên nó không thể truy xuất vào hệ thống, nhờ đó mà hệ điều hành được bảo vệ. Các tiến trình của chương trình người sử dụng có thể truy xuất vào hệ thống thông qua các tiến trình của hệ điều hành bằng cách thực hiện một lời gọi hệ thống.

### **I.1.10. Mô hình tiến trình**

Đa số các hệ điều hành đều muốn đưa sự đa chương, đa nhiệm vào hệ thống. Tức

là, trong hệ thống có thể có nhiều chương trình hoạt động đồng thời (concurrency) với nhau. Về nguyên tắc, để thực hiện được điều này thì hệ thống phải có nhiều processor, mỗi processor có nhiệm vụ thực hiện một chương trình, nhưng mong muốn của hệ điều hành cũng như người sử dụng là *thực hiện sự đa chương trên các hệ thống chỉ có một processor*, và trên thực tế đã xuất hiện nhiều hệ điều hành thực hiện được điều này, hệ điều hành windows9x, windowsNT/2000 chạy trên máy tính cá nhân là một ví dụ. Để thực hiện được điều này hệ điều hành đã sử dụng mô hình tiến trình để tạo ra sự song song giả hay tạo ra các processor logic từ processor vật lý. Các processor logic có thể hoạt động song song với nhau, mỗi processor logic chịu trách nhiệm thực hiện một tiến trình.

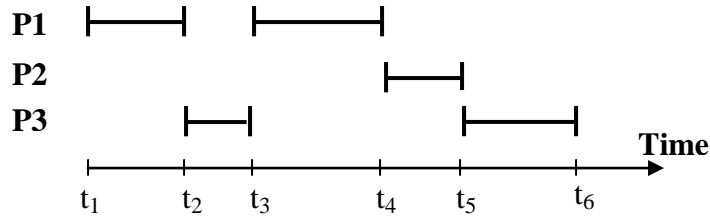
Trong mô hình tiến trình hệ điều hành chia chương trình thành nhiều tiến trình, khởi tạo và đưa vào hệ thống nhiều tiến trình của một chương trình hoặc của nhiều chương trình khác nhau, cấp phát đầy đủ tài nguyên (trừ processor) cho tiến trình và đưa các tiến trình sang trạng thái sẵn sàng. Hệ điều hành bắt đầu cấp processor cho một tiến trình trong số các tiến trình ở trạng thái sẵn sàng để tiến trình này hoạt động, sau một khoảng thời gian nào đó hệ điều hành thu hồi processor của tiến trình này để cấp cho một tiến trình sẵn sàng khác, sau đó hệ điều hành lại thu hồi processor từ tiến trình mà nó vừa cấp để cấp cho tiến trình khác, có thể là tiến trình mà trước đây bị hệ điều hành thu hồi processor khi nó chưa kết thúc, và cứ như thế cho đến khi tất cả các tiến trình mà hệ điều hành khởi tạo đều hoạt động và kết thúc được. Điều đáng chú ý trong mô hình tiến trình này là khoảng thời gian chuyển processor từ tiến trình này sang tiến trình khác hay khoảng thời gian giữa hai lần được cấp phát processor của một tiến trình là rất nhỏ nên các tiến trình có cảm giác luôn được sở hữu processor (logic) hay hệ thống có cảm giác các tiến trình/ chương trình hoạt động song song nhau. Hiện tượng này được gọi là sự song song giả.

Giả sử trong hệ thống có 3 tiến trình sẵn sàng  $P_1$ ,  $P_2$ ,  $P_3$  thì quá trình chuyển processor giữa 3 tiến trình này có thể minh họa như sau:

Thời điểm	Trạng thái các tiến trình
$t_1$	$P_1$ : được cấp processor
$t_2$	$P_1$ : bị thu hồi processor (khi chưa kết thúc) $P_3$ : được cấp processor
$t_3$	$P_3$ : bị thu hồi processor (khi chưa kết thúc) $P_1$ : được cấp processor
$t_4$	$P_1$ : kết thúc và trả lại processor $P_2$ : được cấp processor
$t_5$	$P_2$ : kết thúc và trả lại processor $P_3$ : được cấp processor

$t_6$  P3: kết thúc và trả lại processor

Hình sau đây minh họa quá trình thực hiện của 3 tiến trình  $P_1$ ,  $P_2$ ,  $P_3$  ở trên:



**Hình 2.2:** Sự hoạt động “song song” của các tiến trình  $P_1$ ,  $P_2$ ,  $P_3$

Chúng ta đều biết, chức năng cơ bản của processor là thực hiện các chỉ thị máy (machine instruction) thường trú trong bộ nhớ chính, các chỉ thị này được cung cấp từ một chương trình, chương trình bao gồm một dãy tuần tự các chỉ thị. Và theo trên, tiến trình là một bộ phận của chương trình, nó cũng sở hữu một tập lệnh trong bộ nhớ chính, một con trỏ lệnh,... Nên xét về bản chất, thì việc chuyển processor từ tiến trình này sang tiến trình khác thực chất là việc điều khiển processor để nó thực hiện xen kẽ các chỉ thị bên trong tiến trình. Điều này có thể thực hiện dễ dàng bằng cách thay đổi hợp lý giá trị của con trỏ lệnh, đó chính là cặp thanh ghi CS:IP trong các processor thuộc kiến trúc Intel, để con trỏ lệnh chỉ đến các chỉ thị cần thực hiện trong các tiến trình. Để thấy rõ hơn điều này ta hãy xem ví dụ sau đây:

Giả sử hệ thống cần thực hiện đồng thời 3 tiến trình  $P_1$ ,  $P_2$ ,  $P_3$ , bắt đầu từ tiến trình  $P_1$ . Các chỉ thị của các tiến trình này được nạp vào bộ nhớ tại các địa chỉ như sau:

Tiến trình $P_1$ :	Tiến trình $P_2$ :	Tiến trình $P_3$ :
$a + 0$	$b + 0$	$c + 0$
$a + 1$	$b + 2$	$c + 1$
$a + 3$	$b + 3$	$c + 4$
$a + 5$		$c + 6$

Trong đó:  $a$ : là địa chỉ bắt đầu của chương trình của tiến trình  $P_1$   
 $b$ : là địa chỉ bắt đầu của chương trình của tiến trình  $P_2$   
 $c$ : là địa chỉ bắt đầu của chương trình của tiến trình  $P_3$

Thì giá trị của con trỏ lệnh, chính xác là giá trị cặp thanh ghi CS:IP, lần lượt là:  $a + 0$ ,  $b + 0$ ,  $c + 0$ ,  $a + 1$ ,  $b + 2$ ,  $c + 1$ ,  $a + 3$ ,  $b + 3$ ,  $c + 4$ ,  $a + 5$ ,  $c + 6$ . Tức là, processor thực hiện xen kẽ các chỉ thị của 3 tiến trình  $P_1$ ,  $P_2$ ,  $P_3$  từ lệnh đầu tiên đến lệnh cuối cùng, cho đến khi tất cả các chỉ thị của 3 tiến trình đều được thực hiện. Nhưng khoảng thời gian từ khi con trỏ lệnh =  $a + 0$  đến khi =  $a + 1$ , hay từ khi =  $b + 0$  đến khi =  $b + 2$ , ... là rất nhỏ, nên hệ thống có “cảm giác” 3 tiến trình  $P_1$ ,  $P_2$ ,  $P_3$  hoạt động đồng thời với nhau.

Ví dụ trên đây cho ta thấy bản chất của việc thực hiện song song (hay đồng thời) các tiến trình trên các hệ thống uniprocessor.

Rõ ràng với mô hình tiến trình hệ thống có được 2 điều lợi:

- Tiết kiệm được bộ nhớ: vì không phải nạp tất cả chương trình vào bộ nhớ mà chỉ nạp các tiến trình cần thiết nhất, sau đó tùy theo yêu cầu mà có thể nạp tiếp các tiến trình khác.
- Cho phép các chương trình hoạt động song song nên tốc độ xử lý của toàn hệ thống tăng lên và khai thác tối đa thời gian xử lý của processor.

*Việc chọn thời điểm dừng của tiến trình đang hoạt động (đang chiếm giữ processor) để thu hồi processor chuyển cho tiến trình khác hay việc chọn tiến trình tiếp theo nào trong số các tiến trình đang ở trạng thái sẵn sàng để cấp processor là những vấn đề khá phức tạp đòi hỏi hệ điều hành phải có một cơ chế điều phối thích hợp thì mới có thể tạo ra được hiệu ứng song song giả và sử dụng tối ưu thời gian xử lý của processor. Bộ phận thực hiện chức năng này của hệ điều hành được gọi là bộ điều phối (dispatcher) tiến trình.*

#### **I.1.11. Tiểu trình và tiến trình**

➤ **Tiểu trình:** Thông thường mỗi tiến trình có một không gian địa chỉ và một dòng xử lý. Nhưng trong thực tế có một số ứng dụng cần nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ tiến trình, các dòng xử lý này có thể hoạt động song song với nhau như các tiến trình độc lập trên hệ thống. Để thực hiện được điều này các hệ điều hành hiện nay đưa ra một cơ chế thực thi (các chỉ thị trong chương trình) mới, được gọi là tiểu trình.

Tiểu trình là một đơn vị xử lý cơ bản trong hệ thống, nó hoàn toàn tương tự như tiến trình. Tức là nó cũng phải xử lý tuần tự các chỉ thị máy của nó, nó cũng sở hữu con trỏ lệnh, một tập các thanh ghi, và một không gian stack riêng.

Một tiến trình đơn có thể bao gồm nhiều tiểu trình. Các tiểu trình trong một tiến trình chia sẻ một không gian địa chỉ chung, nhờ đó mà các tiểu trình có thể chia sẻ các biến toàn cục của tiến trình và có thể truy xuất lên các vùng nhớ stack của nhau.

Các tiểu trình chia sẻ thời gian xử lý của processor giống như cách của tiến trình, nhờ đó mà các tiểu trình có thể hoạt động song song (giả) với nhau. Trong quá trình thực thi của tiểu trình nó cũng có thể tạo ra các tiến trình con của nó.

➤ **Đa tiểu trình trong đơn tiến trình:** Điểm đáng chú ý nhất của mô hình tiểu trình là: có nhiều tiểu trình trong phạm vi một tiến trình đơn. Các tiến trình đơn này có thể hoạt động trên các hệ thống multiprocessor hoặc uniprocessor. Các hệ điều hành khác nhau có cách tiếp cận mô hình tiểu trình khác nhau. Ở đây chúng ta tiếp cận mô hình tiểu trình từ mô hình tác vụ (Task), đây là các tiếp cận của windows NT và các hệ điều hành đa nhiệm khác. Trong các hệ điều hành này tác vụ được



định nghĩa như là một đơn vị của sự bảo vệ hay đơn vị cấp phát tài nguyên. Trong hệ thống tồn tại một không gian địa chỉ ảo để lưu giữ tác vụ và một cơ chế bảo vệ sự truy cập đến các file, các tài nguyên Vào/Ra và các tiến trình khác (trong các thao tác truyền thông liên tiến trình).

Trong phạm vi một tác vụ, có thể có một hoặc nhiều tiểu trình, mỗi tiểu trình bao gồm: Một trạng thái thực thi tiểu trình (running, ready,...). Một lưu trữ về ngữ cảnh của processor khi tiểu trình ở trạng thái not running (một cách để xem tiểu trình như một bộ đếm chương trình độc lập hoạt động trong phạm vi tác vụ). Các thông tin thống kê về việc sử dụng các biên cục bộ của tiểu trình. Một stack thực thi. Truy xuất đến bộ nhớ và tài nguyên của tác vụ, được chia sẻ với tất cả các tiểu trình khác trong tác vụ.

Trong các ứng dụng server, chẳng hạn như ứng dụng file server trên mạng cục bộ, khi có một yêu cầu hình thành một file mới, thì một tiểu trình mới được hình thành từ chương trình quản lý file. Vì một server sẽ phải điều khiển nhiều yêu cầu, có thể đồng thời, nên phải có nhiều tiểu trình được tạo ra và được giải phóng trong, có thể đồng thời, một khoảng thời gian ngắn. Nếu server là một hệ thống multiprocessor thì các tiểu trình trong cùng một tác vụ có thể thực hiện đồng thời trên các processor khác nhau, do đó hiệu suất của hệ thống tăng lên. Sự hình thành các tiểu trình này cũng thật sự hữu ích trên các hệ thống uniprocessor, trong trường hợp một chương trình phải thực hiện nhiều chức năng khác nhau. Hiệu quả của việc sử dụng tiểu trình được thấy rõ trong các ứng dụng cần có sự truyền thông giữa các tiến trình hoặc các chương trình khác nhau.

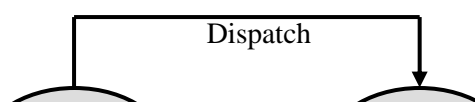
Các thao tác lập lịch và điều phối tiến trình của hệ điều hành thực hiện trên cơ sở tiểu trình. Nhưng nếu có một thao tác nào đó ảnh hưởng đến tất cả các tiểu trình trong tác vụ thì hệ điều hành phải tác động vào tác vụ.

Vì tất cả các tiểu trình trong một tác vụ chia sẻ cùng một không gian địa chỉ, nên tất cả các tiểu trình phải được đưa vào trạng thái suspend tại cùng thời điểm. Tương tự, khi một tác vụ kết thúc thì sẽ kết thúc tất cả các tiểu trình trong tác vụ đó. Trạng thái suspend sẽ được giải thích ngay sau đây.

### I.1.12. Các trạng thái tiến trình

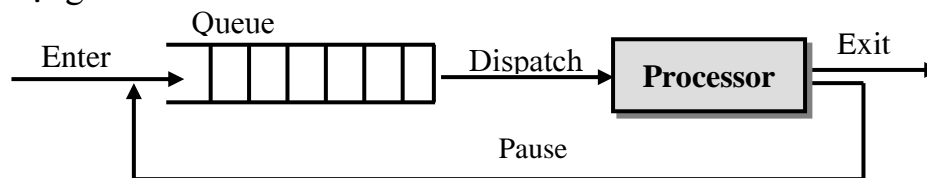
Từ khi được đưa vào hệ thống cho đến khi kết thúc tiến trình tồn tại ở các trạng thái khác nhau. Trạng thái của tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điểm đó.

➤ **Tiến trình hai trạng thái:** Một số ít hệ điều hành chỉ cho phép tiến trình tồn tại ở một trong hai trạng thái: Not Running và Running. Khi hệ điều hành tạo ra một tiến trình mới, hệ điều hành đưa tiến trình đó vào hệ thống ở trạng thái Not Running, tiến trình ở trạng thái này để chờ được chuyển sang trạng thái Running. Vì một lý do nào đó, tiến trình đang thực hiện bị ngắt thì bộ điều phối tiến trình của



hệ điều hành sẽ thu hồi lại processor của tiến trình này và chọn một tiến trình ở trạng thái Not running để cấp processor cho nó và chuyển nó sang trạng thái Running. Tiến trình bị thu hồi processor sẽ được chuyển về lại trạng thái Not running.

Tại một thời điểm xác định chỉ có duy nhất một tiến trình ở trạng thái Running, nhưng có thể có nhiều tiến trình ở trạng thái Not running, các tiến trình ở trạng thái Not running được chứa trong một hàng đợi (Queue). Tiến trình đang ở trạng thái Running bị chuyển sang trạng thái Not running sẽ được đưa vào hàng đợi. Hình vẽ sau đây mô tả việc chuyển trạng thái tiến trình trong các hệ điều hành sử dụng 2 trạng thái tiến trình



**Hình 2.3.b:** Sơ đồ chuyển tiến trình vào hàng đợi

➤ **Tiến trình ba trạng thái:** Đa số hệ điều hành đều cho phép tiến trình tồn tại ở một trong ba trạng thái, đó là: ready, running, blocked:

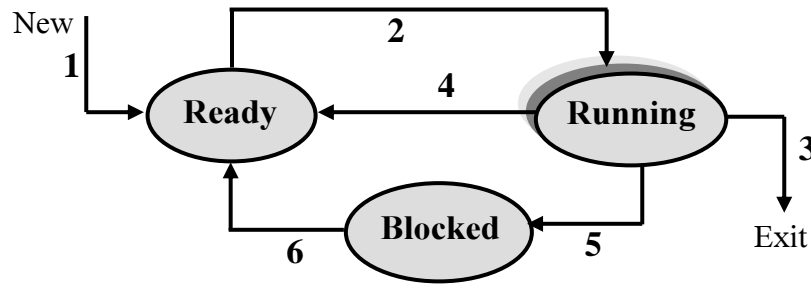
- Trạng thái Ready (sẵn sàng): Ngay sau khi khởi tạo tiến trình, đưa tiến trình vào hệ thống và cấp phát đầy đủ tài nguyên (trừ processor) cho tiến trình, hệ điều hành đưa tiến trình vào trạng thái ready. Hay nói cách khác, trạng thái ready là trạng thái của một tiến trình trong hệ thống đang chờ được cấp processor để bắt đầu thực hiện.

- Trạng thái Running (thực hiện): Là trạng thái mà tiến trình đang được sở hữu processor để hoạt động, hay nói cách khác là các chỉ thị của tiến trình đang được thực hiện/ xử lý bởi processor.

- Trạng thái Blocked (khóa): Là trạng thái mà tiến trình đang chờ để được cấp phát thêm tài nguyên, để một sự kiện nào đó xảy ra, hay một quá trình vào/ra kết thúc.

Quá trình chuyển trạng thái của các tiến trình trong được mô tả bởi sơ đồ

sau:



**Hình 2.4.a:** Sơ đồ chuyển trạng thái tiến trình

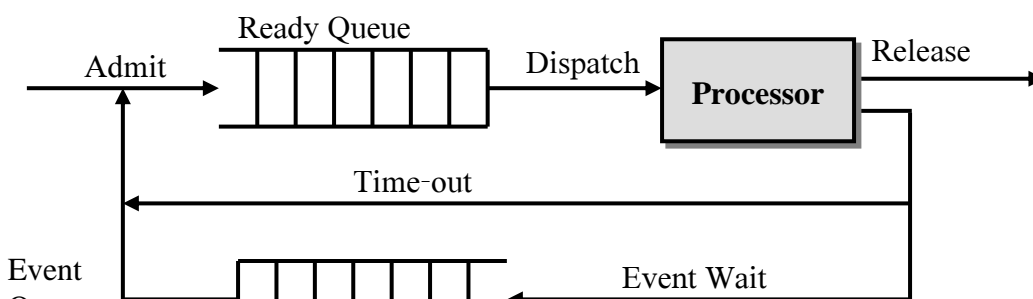
*Trong đó:*

1. (Admit) Tiến trình được khởi tạo, được đưa vào hệ thống, được cấp phát đầy đủ tài nguyên chỉ thiếu processor.
2. (Dispatch) Tiến trình được cấp processor để bắt đầu thực hiện/ xử lý.
3. (Release) Tiến trình hoàn thành xử lý và kết thúc.
4. (Time\_out) Tiến trình bị bộ điều phối tiến trình thu hồi processor, do hết thời gian được quyền sử dụng processor, để cấp phát cho tiến trình khác.
5. (Event wait) Tiến trình đang chờ một sự kiện nào đó xảy ra hay đang chờ một thao vào/ra kết thúc hay tài nguyên mà tiến trình yêu cầu chưa được hệ điều hành đáp ứng.
6. (Event Occurs) Sự kiện mà tiến trình chờ đã xảy ra, thao tác vào/ra mà tiến trình đợi đã kết thúc, hay tài nguyên mà tiến trình yêu cầu đã được hệ điều hành đáp ứng,

Bộ phận điều phối tiến trình thu hồi processor từ một tiến trình đang thực hiện trong các trường hợp sau:

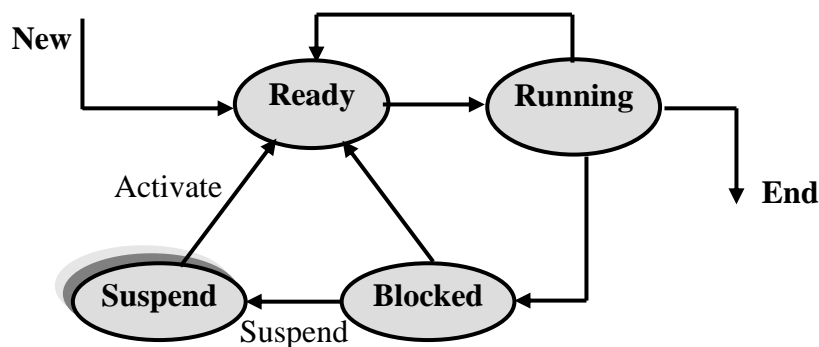
- Tiến trình đang thực hiện hết thời gian (time-out) được quyền sử dụng processor mà bộ phận điều phối dành cho nó.
- Có một tiến trình mới phát sinh và tiến trình mới này có độ ưu tiên cao hơn tiến trình hiện tại.
- Có một tiến trình mới phát sinh và tiến trình này mới cần một khoảng thời gian của processor nhỏ hơn nhiều so với khoảng thời gian còn lại mà tiến trình hiện tại cần processor.

Tại một thời điểm xác định trong hệ thống có thể có nhiều tiến trình đang ở trạng thái Ready hoặc Blocked nhưng chỉ có một tiến trình ở trạng thái Running. Các tiến trình ở trạng thái Ready và Blocked được chứa trong các hàng đợi (Queue) riêng.



Có nhiều lý do để một tiến trình đang ở trạng thái running chuyển sang trạng thái blocked, do đó đa số các hệ điều hành đều thiết kế một hệ thống hàng đợi gồm nhiều hàng đợi, mỗi hàng đợi dùng để chứa những tiến trình đang đợi cùng một sự kiện nào đó.

➤ **Tiến trình 4 trạng thái:** Trong môi trường hệ điều hành đa nhiệm thì việc tổ chức các Queue để lưu các tiến trình chưa thể hoạt động là cần thiết, nhưng nếu tồn tại quá nhiều tiến trình trong Queue, hay chính xác hơn trong bộ nhớ chính, sẽ dẫn đến tình trạng lãng phí bộ nhớ, không còn đủ bộ nhớ để nạp các tiến trình khác khi cần thiết. Mặt khác nếu các tiến trình trong Queue đang chiếm giữ tài nguyên của hệ thống, mà những tài nguyên này lại là những tài nguyên các tiến trình khác đang cần, điều này dẫn đến tình trạng sử dụng tài nguyên không hợp lý, làm cho hệ thống thiếu tài nguyên (thực chất là thừa) trầm trọng và có thể làm cho hệ thống tắc nghẽn. Với những lý do trên các hệ điều hành đa nhiệm thiết kế thêm một trạng thái tiến trình mới, đó là trạng thái Suspend (tạm dừng). Trạng thái này rất cần thiết cho các hệ thống sử dụng kỹ thuật Swap trong việc cấp phát bộ nhớ cho các tiến trình. Khái niệm Swap sẽ được đề cập đến trong chương Quản lý bộ nhớ của tài liệu này.



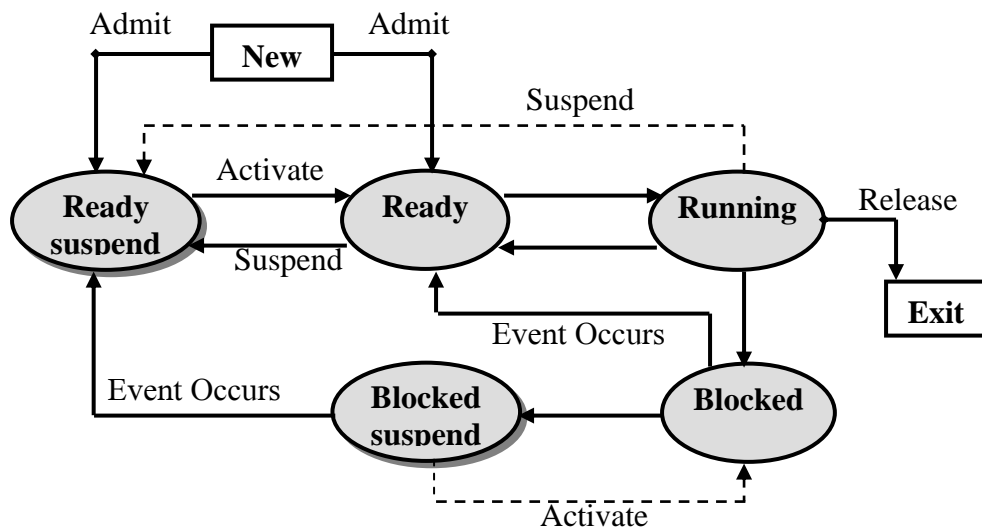
Hình 2.5.a: Sơ đồ chuyển trạng thái tiến trình có

Trạng thái Suspend là trạng thái của một tiến trình khi nó đang được lưu trữ trên bộ nhớ phụ, hay chính xác hơn đây là các tiến trình đang ở trong trạng thái blocked và/hoặc ready bị hệ điều hành chuyển ra đĩa để thu hồi lại không gian nhớ đã cấp cho tiến trình hoặc thu hồi lại tài nguyên đã cấp cho tiến trình để cấp cho một tiến trình khác đang rất cần được nạp vào bộ nhớ tại thời điểm hiện tại.

➤ **Tiến trình 5 trạng thái:** Trong thực tế hệ điều hành thiết kế 2 trạng thái suspend, một trạng thái suspend dành cho các tiến trình từ blocked chuyển đến, trạng thái này được gọi là blocked-suspend và một trạng thái suspend dành cho các tiến trình từ ready chuyển đến, trạng thái này được gọi là ready-suspend.

Tới đây ta có thể hiểu các trạng thái tiến trình như sau:

- Ở trạng thái Ready tiến trình được định vị trong bộ nhớ chính và đang chờ được cấp processor để thực hiện.
- Ở trạng thái Blocked tiến trình được định vị trong bộ nhớ chính và đang đợi một sự kiện hay một quá trình I/O nào đó.
- Ở trạng thái Blocked-suspend tiến trình đang bị chứa trên bộ nhớ phụ (đĩa) và đang đợi một sự kiện nào đó.
- Ở trạng thái Ready-suspend tiến trình đang bị chứa trên bộ nhớ phụ nhưng sẵn sàng thực hiện ngay sau khi được nạp vào bộ nhớ chính.



Hình 2.5.b: Sơ đồ chuyển trạng thái tiến trình với 2

Sau đây chúng ta xem xét sự chuyển trạng thái tiến trình trong sơ đồ trên:

**1.** Blocked sang Blocked-suspend: nếu không còn tiến trình ready trong bộ nhớ chính và bộ nhớ chính không còn không gian nhớ trống thì phải có ít nhất một tiến trình blocked bị chuyển ra ngoài, blocked-suspend, để dành bộ nhớ cho một tiến trình không bị khoá (not blocked) khác.

**2.** Blocked-suspend sang Ready-suspend: một tiến trình đang ở trạng thái blocked-suspend được chuyển sang trạng thái ready-suspend khi sự kiện mà nó đợi đã xảy ra.

**3.** Ready-suspend sang Ready: có 2 lý do để hệ điều hành chọn khi chuyển một tiến trình ở trạng thái ready-suspend sang trạng thái ready:

- Không còn tiến trình ready trong bộ nhớ chính, hệ điều hành phải nạp một tiến trình mới vào để nó tiếp tục thực hiện
- Nếu có tiến trình ready-suspend có độ ưu tiên cao hơn so với các tiến trình ready hiện tại thì hệ điều hành có thể chuyển nó sang trạng thái ready để nó nhiều cơ hội để được thực hiện hơn.

**4.** Ready sang Ready suspend: Hệ điều hành thường chuyển các tiến trình blocked sang suspend hơn là các tiến trình ready, vì các tiến trình ở trạng thái blocked không thể thực hiện ngay lập tức nhưng lại chiếm nhiều không gian bộ nhớ chính hơn so với các tiến trình ở trạng thái ready. Tuy nhiên, nếu việc chọn tiến trình để chuyển sang suspend dựa vào 2 điều kiện: chiếm ít không gian bộ nhớ hơn và có độ ưu tiên thấp hơn thì hệ điều hành có thể chuyển một tiến trình ready sang trạng thái suspend.

*Như vậy với việc chuyển tiến trình sang trạng thái suspend hệ điều hành sẽ chủ động hơn trong việc cấp phát bộ nhớ và ngăn chặn các tình huống tắc nghẽn có thể xảy ra do sự tranh chấp về tài nguyên, nhờ vậy mà hệ điều hành tiết kiệm được bộ nhớ, chia sẻ được tài nguyên cho nhiều tiến trình và tăng được mức độ đa chương của hệ thống. Tuy nhiên, để có được những lợi ích trên hệ điều hành đã phải chi phí rất nhiều cho việc tạm dừng tiến trình. Hệ điều hành phải xem xét tiến trình nào được chọn để suspend, khi suspend một tiến trình hệ điều hành phải lưu lại tất cả các thông tin liên quan đến tiến trình đó (con trỏ lệnh, tài nguyên mà tiến trình đã được cấp, ...), hệ điều hành phải lựa chọn thời điểm thích hợp để đưa tiến trình ra bộ nhớ ngoài, ... những thao tác đó sẽ làm chậm tốc độ thực hiện của toàn bộ hệ thống. Nhưng dầu sao đi nữa thì hệ điều hành vẫn phải sử dụng trạng thái*

*suspend vì tăng mức độ đa chương của hệ thống là một trong những mục tiêu lớn của hệ điều hành.*

### **I.1.13. Cấu trúc dữ liệu của khối quản lý tiến trình**

Để quản lý các tiến trình và tài nguyên trong hệ thống, hệ điều hành phải có các thông tin về trạng thái hiện thời của mỗi tiến trình và tài nguyên. Trong trường hợp này hệ điều hành xây dựng và duy trì các bảng thông tin về mỗi đối tượng (memory, devices, file, process) mà nó quản lý, đó là các bảng: memory table cho đối tượng bộ nhớ, I/O table cho đối tượng thiết bị vào/ra, file table cho đối tượng tập tin, process table cho đối tượng tiến trình. Memory table được sử dụng để theo dõi cả bộ nhớ thực lẫn bộ nhớ ảo, nó phải bao gồm các thông tin sau: Không gian bộ nhớ chính dành cho tiến trình. Không gian bộ nhớ phụ dành cho tiến trình. Các thuộc tính bảo vệ bộ nhớ chính và bộ nhớ ảo. Các thông tin cần thiết để quản lý bộ nhớ ảo. Ở đây chúng tôi điếm qua một vài thông tin về memory table, là để lưu ý với các bạn rằng: nhiệm vụ quản lý tiến trình và quản lý bộ nhớ của hệ điều hành có quan hệ chéo với nhau, bộ phận quản lý tiến trình cần phải có các thông tin về bộ nhớ để điều khiển sự hoạt động của tiến trình, ngược lại bộ phận quản lý bộ nhớ phải có các thông tin về tiến trình để tổ chức nạp tiến trình vào bộ nhớ, ... Điều này cũng đúng với các bộ phận quản lý Vào/ ra và quản lý tập tin. Trong phần trình bày sau đây chúng tôi chỉ đề cập đến Process Table của hệ điều hành.

Để quản lý và điều khiển được một tiến trình, thì hệ điều hành phải biết được vị trí nạp tiến trình trong bộ nhớ chính, phải biết được các thuộc tính của tiến trình cần thiết cho việc quản lý tiến trình của nó:

➤ **Định vị của tiến trình (process location):** định vị của tiến trình phụ thuộc vào chiến lược quản lý bộ nhớ đang sử dụng. Trong trường hợp đơn giản nhất, tiến trình, hay chính xác hơn là hình ảnh tiến trình, được lưu giữa tại các khối nhớ liên tục trên bộ nhớ phụ (thường là đĩa), để tiến trình thực hiện được thì tiến trình phải được nạp vào bộ nhớ chính. Do đó, hệ điều hành cần phải biết định vị của mỗi tiến trình trên đĩa và cho mỗi tiến trình đó trên bộ nhớ chính. Trong một số chiến lược quản lý bộ nhớ, hệ điều hành chỉ cần nạp một phần tiến trình vào bộ nhớ chính, phần còn lại vẫn nằm trên đĩa. Hay tiến trình đang ở trên bộ nhớ chính thì có một phần bị swap-out ra lại đĩa, phần còn lại vẫn còn nằm ở bộ nhớ chính. Trong các trường hợp này hệ điều hành phải theo dõi tiến trình để biết phần nào của tiến trình là đang ở trong bộ nhớ chính, phần nào của tiến trình là còn ở trên đĩa.

Đa số các hệ điều hành hiện nay đều sử dụng chiến lược quản lý bộ nhớ mà trong đó không gian địa chỉ của tiến trình là một tập các block, các block này có thể không liên tiếp nhau. Tùy theo chiến lược bộ nhớ sử dụng mà các block này có thể có chiều dài cố định (chiến lược phân phân trang bộ nhớ) hay thay đổi (chiến lược phân đoạn bộ nhớ) hay kết hợp cả hai. Hệ điều hành cho phép không nạp tất cả các trang (page) và/hoặc các đoạn (segment) của tiến trình vào bộ nhớ. Do đó, process

table phải được duy trì bởi hệ điều hành và phải cho biết vị trí của mỗi trang/ đoạn tiến trình trên hệ thống. Những điều trên đây sẽ được làm rõ ở phần chiến lược cấp phát bộ nhớ trong chương *Quản lý bộ nhớ* của tài liệu này.

➤ **Các thuộc tính của tiến trình:** Trong các hệ thống đa chương, thông tin về mỗi tiến trình là rất cần cho công tác quản lý tiến trình của hệ điều hành, các thông tin này có thể thường trú trong khối quản lý tiến trình (PCB: process control block). Các hệ điều hành khác nhau sẽ có cách tổ chức PCB khác nhau, ở đây chúng ta khảo sát một trường hợp chung nhất. Các thông tin trong PCB có thể được chia thành ba nhóm chính:

- Định danh tiến trình (PID: process identification): mỗi tiến trình được gán một định danh duy nhất để phân biệt với các tiến trình khác trong hệ thống. Định danh của tiến trình có thể xuất hiện trong memory table, I/O table. Khi tiến trình này truyền thông với tiến trình khác thì định danh tiến trình được sử dụng để hệ điều hành xác định tiến trình đích. Khi tiến trình cho phép tạo ra tiến trình khác thì định danh được sử dụng để chỉ đến tiến trình cha và tiến trình con của mỗi tiến trình. Tóm lại, các định danh có thể lưu trữ trong PCB bao gồm: định danh của tiến trình này, định danh của tiến trình tạo ra tiến trình này, định danh của người sử dụng.

- Thông tin trạng thái processor (processor state information): bao gồm các thanh ghi User-visible, các thanh ghi trạng thái và điều khiển, các con trỏ stack.

- Thông tin điều khiển tiến trình (process control information): bao gồm thông tin trạng thái và lập lịch, cấu trúc dữ liệu, truyền thông liên tiến trình, quyền truy cập tiến trình, quản lý bộ nhớ, tài nguyên khởi tạo và tài nguyên sinh ra.

PCB là một trong những cấu trúc dữ liệu trung tâm và quan trọng của hệ điều hành. Mỗi PCB chứa tất cả các thông tin về tiến trình mà nó rất cần cho hệ điều hành. Có nhiều modun thành phần trong hệ điều hành có thể read và/hoặc modified PCB như: lập lịch tiến trình, cấp phát tài nguyên cho tiến trình, ngắt tiến trình, vv. Có thể nói các thiết lập trong PCB định nghĩa trạng thái của hệ điều hành.

#### **I.1.14. Các thao tác điều khiển tiến trình**

➤ **Khi khởi tạo tiến trình hệ điều hành thực hiện các thao tác sau:**

- Hệ điều hành gán PID cho tiến trình mới và đưa tiến trình vào danh sách quản lý của hệ thống, tức là, dùng một entry trong PCB để chứa các thông tin liên quan đến tiến trình mới tạo ra này.

- Cấp phát không gian bộ nhớ cho tiến trình. Ở đây hệ điều hành cần phải xác định được kích thước của tiến trình, bao gồm code, data và stack. Giá trị kích thước này có thể được gán mặt định dựa theo loại của tiến trình hoặc được gán theo yêu cầu của người sử dụng khi có một công việc (job) được tạo. Nếu một tiến trình được sinh ra bởi một tiến trình khác, thì tiến trình cha có thể chuyển kích thước của



nó đến hệ điều hành trong yêu cầu tạo tiến trình.

- Khởi tạo các thông tin cần thiết cho khối điều khiển tiến trình như các PID của tiến trình cha (nếu có), thông tin trạng thái tiến trình, độ ưu tiên của tiến trình, thông tin ngữ cảnh của processor (bộ đếm chương trình và các thanh ghi khác), vv.

- Cung cấp đầy đủ các tài nguyên cần thiết nhất, trừ processor, để tiến trình có thể vào trạng thái ready được hoặc bắt đầu hoạt động được.

- Đưa tiến trình vào một danh sách tiến trình nào đó: ready list, suspend list, waiting list, vv, sao cho phù hợp với chiến lược điều phối tiến trình hiện tại của bộ phận điều phối tiến trình của hệ điều hành.

Khi một tiến trình tạo lập một tiến trình con, tiến trình con có thể được cấp phát tài nguyên bởi chính hệ điều hành, hoặc được tiến trình cha cho thừa hưởng một số tài nguyên ban đầu của nó.

➤ **Khi kết thúc tiến trình hệ điều hành thực hiện các thao tác sau:** Khi tiến trình kết thúc xử lý, hoàn thành chỉ thị cuối cùng, hệ điều hành sẽ thực hiện các thao tác sau đây:

- Thu hồi tài nguyên đã cấp phát cho tiến trình.
- Loại bỏ tiến trình ra khỏi danh sách quản lý của hệ thống.
- Huỷ bỏ khối điều khiển tiến trình.

Hầu hết các hệ điều hành đều không cho phép tiến trình con hoạt động khi tiến trình cha đã kết thúc. Trong những trường hợp như thế hệ điều hành sẽ chủ động việc kết thúc tiến trình con khi tiến trình cha vừa kết thúc.

➤ **Khi thay đổi trạng thái tiến trình hệ điều hành thực hiện các bước sau:** Khi một tiến trình đang ở trạng thái running bị chuyển sang trạng thái khác (ready, blocked, ...) thì hệ điều hành phải tạo ra sự thay đổi trong môi trường làm việc của nó. Sau đây là các bước mà hệ điều hành phải thực hiện đầy đủ khi thay đổi trạng thái tiến trình:

- Lưu (save) ngữ cảnh của processor, bao gồm thanh ghi bộ đếm chương trình (PC: program counter) và các thanh ghi khác.

- Cập nhật PCB của tiến trình, sao cho phù hợp với trạng thái mới của tiến trình, bao gồm trạng thái mới của tiến trình, các thông tin tính toán, vv.

- Di chuyển PCB của tiến trình đến một hàng đợi thích hợp, để đáp ứng được các yêu cầu của công tác điều phối tiến trình.

- Chọn một tiến trình khác để cho phép nó thực hiện.

- Cập nhật PCB của tiến trình vừa được chọn thực hiện ở trên, chủ yếu là thay đổi trạng thái của tiến trình đến trạng thái running.

- Cập nhật các thông tin liên quan đến quản lý bộ nhớ. Bước này phụ thuộc vào các yêu cầu chuyển đổi địa chỉ bộ nhớ đang được sử dụng.
- Khôi phục (Restore) lại ngữ cảnh của processor và thay đổi giá trị của bộ đếm chương trình và các thanh ghi khác sao cho phù hợp với tiến trình được chọn ở trên, để tiến trình này có thể bắt đầu hoạt động được.

Như vậy, khi hệ điều hành chuyển một tiến trình từ trạng thái running (đang chạy) sang một trạng thái nào đó (tạm dừng) thì hệ điều hành phải lưu trữ các thông tin cần thiết, nhất là Program Count, để sau này hệ điều hành có thể cho tiến trình tiếp tục hoạt động trở (tái kích hoạt) lại được. Đồng thời hệ điều hành phải chọn một tiến trình nào đó đang ở trạng thái ready để cho tiến trình này chạy (chuyển tiến trình sang trạng thái running). Tại đây, trong các thao tác phải thực hiện, hệ điều hành phải thực hiện việc thay đổi giá trị của PC, thay đổi ngữ cảnh processor, để PC chỉ đến địa chỉ của chỉ thị đầu tiên của tiến trình running mới này trong bộ nhớ. Đây cũng chính là bản chất của việc thực hiện các tiến trình trong các hệ thống uniprocessor.

## **I.16. Tài nguyên căng và đoạn căng**

### **II.2.4. Tài nguyên căng (Critical Resource)**

Trong môi trường hệ điều hành đa nhiệm - đa chương – đa người sử dụng, việc chia sẻ tài nguyên cho các tiến trình của người sử dụng dùng chung là cần thiết, nhưng nếu hệ điều hành không tổ chức tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời, thì không những không mang lại hiệu quả khai thác tài nguyên của hệ thống mà còn làm hỏng dữ liệu của các ứng dụng. Và nguy hiểm hơn là việc hỏng dữ liệu này có thể hệ điều hành và ứng dụng không thể phát hiện được. Việc hỏng dữ liệu của ứng dụng có thể làm sai lệch ý nghĩa thiết kế của nó. Đây là điều mà cả hệ điều hành và người lập trình đều không mong muốn.

Các tiến trình hoạt động đồng thời thường cạnh tranh với nhau trong việc sử dụng tài nguyên dùng chung. Hai tiến trình hoạt động đồng thời cùng ghi vào một không gian nhớ chung (một biến chung) trên bộ nhớ hay hai tiến trình đồng thời cùng ghi dữ liệu vào một file chia sẻ, đó là những biểu hiện của sự cạnh tranh về việc sử dụng tài nguyên dùng chung của các tiến trình. Để các tiến trình hoạt động đồng thời không cạnh tranh hay xung đột với nhau khi sử dụng tài nguyên dùng chung hệ điều hành phải tổ chức cho các tiến trình này được độc quyền truy xuất/sử dụng trên các tài nguyên dùng chung này.

Những tài nguyên được hệ điều hành chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung, mà có nguy cơ dẫn đến sự tranh chấp giữa các tiến trình này khi sử dụng chúng, được gọi là tài nguyên căng. Tài nguyên căng có thể là tài nguyên phân cứng hoặc tài nguyên phân mềm, có thể là tài nguyên phân chia được hoặc không phân chia được, nhưng đa số thường là tài nguyên phân chia được như

là: các biến chung, các file chia sẻ.

Các ví dụ sau đây cho thấy hậu quả của việc sử dụng tài nguyên găng trong các chương trình có các tiến trình hoạt động đồng thời:

**Ví dụ 1:** Giả sử có một chương trình, trong đó có hai tiến trình P1 và P2 hoạt động đồng thời với nhau. Tiến trình P1 phải tăng biến Count lên 1 đơn vị, tiến trình P2 phải tăng biến Count lên 1 đơn vị, với mục đích tăng Count lên được 2 đơn vị.

Chương trình có thể thực hiện như sau:

1. Tiến trình P1 ghi nội dung biến toàn cục Count vào biến cục bộ L1
2. Tiến trình P2 ghi nội dung biến toàn cục Count vào biến cục bộ L2
3. Tiến trình P1 thực hiện  $L1 := L1 + 1$  và  $Count := L1$
4. Tiến trình P2 thực hiện  $L2 := L2 + 1$  và  $Count := L2$

Như vậy thoạt nhìn ta thấy rằng chắc chắn Count đã tăng được 2 đơn vị, nhưng trong thực tế có thể Count chỉ tăng được 1 đơn vị. Bởi vì, nếu P1 và P2 đồng thời nhận giá trị của Count (giả sử ban đầu  $Count = 4$ ) vào L1 và L2, sau đó P1 tăng L1 lên 1 và P2 tăng L2 lên 1 ( $L1 = 5$ ,  $L2 = 5$ ), rồi sau đó cả P1 và P2 đồng thời ghi giá trị biến L của nó vào lại Count, thì Count chỉ tăng được 1 đơn vị,  $Count = 5$ . Đây là điều mà chương trình không mong muốn nhưng cả chương trình và hệ điều hành đều khó có thể phát hiện được.

Nguyên nhân ở trên là do 2 tiến trình P1 và P2 đồng thời truy xuất biến Count, cả khi nhận giá trị của count, lẫn khi ghi giá trị vào Count. Trong trường hợp này nếu hệ điều hành không cho phép hai tiến trình P1 và P2 đồng thời truy xuất Count, hoặc hệ điều hành cho phép mỗi tiến trình được độc quyền truy xuất Count trong đoạn code sau, thì lỗi trên sẽ không xảy ra.

P1: Begin	P2: Begin
$L1 := Count$ ;	$L2 := Count$ ;
$L1 := L1 + 1$ ;	$L2 := L2 + 1$ ;
$Count := L1$ ;	$Count := L2$ ;
End;	End;

Trong trường hợp này tài nguyên găng là biến count.

**Ví dụ 2:** Giả sử có một ứng dụng Kế toán, hoạt động trong môi trường đa nhiệm, đa người sử dụng. Mỗi người sử dụng trong môi trường này khi cần thực hiện thao tác rút tiền từ trong tài khoản chung thì phải khởi tạo một tiến trình, tạm gọi là tiến trình rút tiền, tiến trình rút tiền chỉ có thể thực hiện được thao tác rút tiền khi số tiền cần rút nhỏ hơn số tiền còn lại trong tài khoản chung. Trong môi trường này có thể có nhiều người sử dụng đồng thời thực hiện thao tác rút tiền từ tài khoản chung của hệ thống.

Như vậy các tiến trình rút tiền, giả sử có hai tiến trình rút tiền P1 và P1, có

thể hoạt động đồng thời với nhau và cùng chia sẻ không gian nhớ lưu trữ biến **Tài khoản**, cho biết số tiền còn trong tài khoản dùng chung của hệ thống. Và mỗi tiến trình rút tiền khi muốn rút một khoảng tiền từ tài khoản (**Tiền rút**) thì phải thực hiện kiểm tra **Tài khoản** sau đó mới thực hiện việc rút tiền. Tức là mỗi tiến trình rút tiền, khi cần rút tiền đều phải thực hiện đoạn code sau đây:

```
IF (Tài khoản - Tiền rút >= 0)           {kiểm tra tài khoản}
    Tài khoản := Tài khoản - Tiền rút    {thực hiện rút tiền}
Else
    Thông báo lỗi                        {không thể rút tiền}
EndIf;
```

Nếu tại một thời điểm nào đó:

- Trong tài khoản còn 800 ngàn đồng (**Tài khoản = 800**).
- Tiến trình rút tiền P1 cần rút 500 ngàn đồng (**Tiền rút = 500**).
- Tiến trình rút tiền P2 cần rút 400 ngàn đồng (**Tiền rút = 400**).
- Tiến trình P1 và P2 đồng thời rút tiền.

Thì theo nguyên tắc điều trên không thể xảy ra, vì tổng số tiền mà hai tiến trình cần rút lớn hơn số tiền còn lại trong tài khoản ( $500 + 400 > 800$ ). Nhưng trong môi trường đa nhiệm, đa người sử dụng nếu hệ điều hành không giám sát tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời thì điều trên vẫn có thể xảy ra. tức là, cả hai tiến trình P1 và P2 đều thành công trong thao tác rút tiền, mà ứng dụng cũng như hệ điều hành không hề phát hiện. Bởi vì, quá trình rút tiền của các tiến trình P1 và P2 có thể diễn ra như sau:

1. P1 được cấp processor để thực hiện việc rút tiền: P1 thực hiện kiểm tra tài khoản: **Tài khoản - Tiền rút =  $800 - 500 = 300 > 0$** , P1 ghi nhận điều này và chuẩn bị rút tiền.

2. Nhưng khi P1 chưa kịp rút tiền thì bị hệ điều hành thu hồi lại processor, và hệ điều hành cấp processor cho P2. P1 được chuyển sang trạng thái ready.

3. P2 nhận được processor, được chuyển sang trạng thái running, nó bắt đầu thực hiện việc rút tiền như sau: kiểm tra tài khoản: **Tài khoản - Tiền rút =  $800 - 400 = 500 \geq 0$** , P2 ghi nhận điều này và thực hiện rút tiền:

$$\text{Tài khoản} = \text{Tài khoản} - \text{Tiền rút} = 800 - 400 = 400.$$

4. P2 hoàn thành nhiệm vụ rút tiền, nó kết thúc xử lý và trả lại processor cho hệ điều hành. Hệ điều hành cấp lại processor cho P1, tái kích hoạt lại P1 để nó tiếp tục thao tác rút tiền.

5. Khi được hoạt động trở lại P1 thực hiện ngay việc rút tiền mà không thực hiện việc kiểm tra tài khoản (vì đã kiểm tra trước đó):

$$\text{Tài khoản} = \text{Tài khoản} - \text{Tiền rút} = 400 - 500 = -100.$$

6. P1 hoàn thành nhiệm vụ rút tiền và kết thúc tiến trình.

Như vậy cả 2 tiến trình P1 và P2 đều hoàn thành việc rút tiền, không thông báo lỗi, mà không gặp bất kỳ một lỗi hay một trở ngại nào. Nhưng đây là một lỗi nghiêm trọng đối với ứng dụng, vì không thể rút một khoảng tiền lớn hơn số tiền còn lại trong tài khoản, hay **Tài khoản** không thể nhận giá trị âm.

Nguyên nhân của lỗi này không phải là do hai tiến trình P1 và P2 đồng thời truy xuất biến Tài khoản, mà do hai thao tác: kiểm tra tài khoản và thực hiện rút tiền, của các tiến trình này bị tách rời nhau. Nếu hệ điều hành làm cho hai thao tác này không tách rời nhau thì lỗi này sẽ không xảy ra.

Trong trường hợp này tài nguyên căng là biến Tài khoản.

**Ví dụ 3:** Giả sử một hệ điều hành đa nhiệm, cung cấp cho các tiến trình của các chương trình người sử dụng một thủ tục Echo. Thủ tục Echo này cho phép các tiến trình nhận một kí tự từ bàn phím rồi đưa kí tự này lên màn hình, mỗi khi gọi nó. Tất cả các tiến trình của chương trình người sử dụng trong hệ thống có thể đồng thời gọi Echo mỗi khi cần đưa một kí tự từ bàn phím lên màn hình. Sau đây là code của thủ tục Echo:

```
Procedure Echo;  
Var  
    out, in: chracter;  
Begin  
    Input(In, keyboard);    {Input là hàm nhập, nó nhận kí tự}  
    Out:=In;                {từ bàn phím đưa vào In. Output là}  
    Output(Out, Screen);    {hàm xuất, nó đưa kí tự từ biến Out}  
End;                       {lên màn hình}
```

Để tiết kiệm bộ nhớ hệ điều hành nạp Echo vào không gian nhớ toàn cục của hệ thống và các tiến trình sẽ chia sẻ không gian nhớ chứa thủ tục Echo này. Sự chia sẻ này là cần thiết và hữu ích, nhưng các tiến trình, hai tiến trình P1 và P2, có thể không đạt được mục tiêu khi gọi Echo, có thể tiến trình P1 gõ kí tự A nhưng màn hình lại xuất hiện kí tự B, B là kí tự của tiến trình P2. Bởi vì hệ thống có thể xảy ra trường hợp sau:

1. Tiến trình P1 gọi thủ tục Echo và bị ngắt ngay lập tức sau khi hàm nhập Input được thực hiện. Tại thời điểm này, kí tự vừa được nhập gần đây nhất là A, được lưu trữ trong biến In.
2. Tiến trình P2 được kích hoạt và gọi thủ tục Echo, và thủ tục được chạy cho đến khi kết thúc. Giả sử đã nhập và xuất kí tự B ra màn hình.
3. Tiến trình P1 được tiếp tục trở lại. Lúc này giá trị A của biến In đã bị ghi đè, có thể là kí tự B của tiến trình P2, biến In = B. Tiến trình P1 tiếp tục

công việc của thủ tục Echo,  $\text{Out} := \text{In}$  và  $\text{Out} = \text{B}$ . Sau đó hàm xuất Output sẽ đưa giá trị của biến out lên màn hình. Tức là trên màn hình xuất hiện kí tự B. Đây là điều mà tiến trình P1 không hề mong muốn.

Như vậy là kí tự A bị mất, nhưng kí tự B lại xuất hiện hai lần. Bản chất của vấn đề này là nằm ở biến toàn cục In (tài nguyên găng là biến In). Vì hệ điều hành đã để cho nhiều tiến trình hoạt động đồng thời trên hệ thống có quyền truy xuất và truy xuất đồng thời vào biến này. Để tránh lỗi này hệ điều hành cần phải có cơ chế để bảo vệ biến toàn cục dùng chung và chỉ cho phép một tiến trình duy nhất điều khiển các code truy xuất đến nó. Nếu hệ điều hành chấp nhận quy tắc: tại một thời điểm chỉ có một tiến trình được phép sử dụng thủ tục Echo và thủ tục này phải chạy cho đến khi hoàn thành mới được trao cho tiến trình khác. Thì lỗi trên sẽ không còn xuất hiện nữa. Việc sử dụng thủ tục Echo của các tiến trình P1 và P2 có thể xảy ra theo thứ tự như sau:

1. Tiến trình P1 gọi thủ tục Echo và bị dừng lại ngay sau khi hàm input được thực hiện xong. Giả sử  $\text{In} = \text{A}$ .
2. Tiến trình P2 được kích hoạt và gọi thủ tục Echo. Nhưng vì tiến trình P1 còn đang ở trong thủ tục này, cho dù đang bị treo, nên P2 phải được chuyển sang trạng thái blocked để chờ thủ tục Echo rồi.
3. Một khoảng thời gian sau, tiến trình P1 được tái kích hoạt trở lại. P1 tiếp tục thủ tục echo cho đến khi hoàn thành. Tức là, đã hiển thị kí tự A lên màn hình.
4. Khi kết thúc P1 trả lại thủ tục echo. Khi đó P2 toàn quyền sử dụng thủ tục Echo để nhập và hiển thị kí tự lên màn hình.

Trường hợp này không xảy ra lỗi là do tiến trình P2 không tiếp tục thủ tục Echo, mặc dù đã gọi, vì nó biết P1 đã đang ở trong thủ tục Echo. Chúng ta nên lưu ý điều này, điều này sẽ được thảo luận trong mục các phương pháp điều độ tiến trình qua đoạn găng ngay đây.

Qua các ví dụ trên ta thấy rằng trong các hệ thống đa chương, đa người sử dụng thường xảy ra hiện tượng, nhiều tiến trình đồng thời cùng đọc/ghi dữ liệu vào một vùng nhớ, nơi chứa các biến của chương trình, và nếu không có sự can thiệp của hệ điều hành thì có thể gây hậu quả nghiêm trọng cho ứng dụng và cho cả hệ thống. Để ngăn chặn các tình huống trên hệ điều hành phải thiết lập cơ chế độc quyền truy xuất trên tài nguyên dùng chung. Tức là, tại mỗi thời điểm chỉ có một tiến trình duy nhất được phép truy xuất trên các tài nguyên dùng chung. Nếu có nhiều tiến trình hoạt động đồng thời cùng yêu cầu truy xuất tài nguyên dùng chung thì chỉ có một tiến trình được chấp nhận truy xuất, các tiến trình khác phải xếp hàng chờ để được truy xuất sau.

Chúng ta cũng thấy rằng nguyên nhân tiềm ẩn của sự xung đột giữa các tiến

trình hoạt động đồng thời khi sử dụng tài nguyên gắng là: các tiến trình này hoạt động đồng thời với nhau một cách hoàn toàn độc lập và không trao đổi thông tin với nhau nhưng sự thực thi của các tiến trình này lại ảnh hưởng đến nhau. Trường hợp lỗi trong ví dụ 3 ở trên minh chứng cho điều này.

### II.2.5. Đoạn găng (Critical Section)

Đoạn code trong các tiến trình đồng thời, có tác động đến các tài nguyên có thể trở thành tài nguyên gắng được gọi là đoạn găng hay miền găng. Tức là, các đoạn code trong các chương trình dùng để truy cập đến các vùng nhớ chia sẻ, các tập tin chia sẻ được gọi là các đoạn găng.

Trong ví dụ 2 ở trên, đoạn code sau đây là đoạn găng:

```
{ IF (Tài khoản - Tiền rút >= 0)
    Tài khoản := Tài khoản - Tiền rút }
```

Trong ví dụ 1 ở trên có hai đoạn găng là:

```
{ L1 := Count    và    Count := L1    }.
```

Để hạn chế các lỗi có thể xảy ra do sử dụng tài nguyên gắng, hệ điều hành phải điều khiển các tiến trình sao cho, tại một thời điểm chỉ có một tiến trình nằm trong đoạn găng, nếu có nhiều tiến trình cùng muốn vào (thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi (kết thúc) đoạn găng phải báo cho hệ điều hành và/hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng, vv. Các công tác điều khiển tiến trình thực hiện đoạn găng của hệ điều hành được gọi là *điều độ tiến trình qua đoạn găng*. Để công tác điều độ tiến trình qua đoạn găng được thành công, thì cần phải có sự phối hợp giữa vi xử lý, hệ điều hành và người lập trình. Vi xử lý đưa ra các chỉ thị, hệ điều hành cung cấp các công cụ để người lập trình xây dựng các sơ đồ điều độ hợp lý, để đảm bảo sự độc quyền trong việc sử dụng tài nguyên gắng của các tiến trình.

Trong phần sau đây chúng ta sẽ tìm hiểu về các phương pháp và các sơ đồ điều độ tiến trình qua đoạn găng. Nhưng trước hết ở đây chúng ta chấp nhận một mẫu chương trình được sử dụng trong các sơ đồ điều độ tiến trình. Mẫu chương trình này mang tính chất trừu tượng, dùng để minh họa cho các ý tưởng điều độ. Rất ít ngôn ngữ lập trình hỗ trợ cú pháp viết chương trình điều độ này. Mặc dầu đã cung cấp đầy đủ các công cụ điều độ tiến trình cho người lập trình, nhưng các hệ điều hành hiện nay đều tổ chức điều độ tiến trình ngay trong lõi (kernel) của nó nên người lập trình ít quan tâm đến tổ chức điều độ tiến trình khi lập trình. Sau đây là sơ đồ điều độ minh họa:

Program MutualExclusion;

Const

N = ..... /\*số lượng tiến trình \*/

{-----}

```

Procedure P(i: integer);
Begin
    Repeat
        EnterCritical(R);    {kiểm tra và xác lập quyền vào đoạn găng}
        <Đoạn găng của P>;
        ExitCritical(R);      {xác lập khi rời đoạn găng}
        <Đoạn không găng của>;
    Until .F.
End;
{-----}
BEGIN                      {*chương trình chính chứa các tiến trình đồng thời*}
    PerBegin
        P(1);
        P(2);
        ....
        P(n);
    ParEnd;
END.
{-----}

```

Sơ đồ trên tổ chức điều độ cho n tiến trình P, n tiến trình này hoạt động đồng thời với nhau và chia sẻ tài nguyên dùng chung R. Mỗi tiến trình trong trường hợp này có một đoạn găng với tài nguyên R. Để tổ chức truy xuất độc quyền trên tài nguyên găng, mỗi tiến trình trước khi vào đoạn găng tiến trình phải gọi thủ tục EnterCritical để thiết lập quyền vào đoạn găng, để báo cho các tiến trình biết là tiến trình hiện tại đang ở trong đoạn găng. Để ra khỏi đoạn găng mỗi tiến trình phải gọi thủ tục ExitCritical, để báo cho các tiến trình khác biết là tiến trình hiện tại đã ra khỏi đoạn găng.

## II.2.6. Yêu cầu của công tác điều độ qua đoạn găng

Trước hết chúng ta lưu ý lại rằng, nhiệm vụ điều độ tiến trình phải là sự phối hợp giữ phần cứng vi xử lý, hệ điều hành, ngôn ngữ lập trình và người lập trình, trong đó nhiệm vụ chính là của hệ điều hành và người lập trình. Vi xử lý, hệ điều hành và ngôn ngữ lập trình cung cấp các công cụ để hệ điều hành và/hoặc người lập trình tổ chức sơ đồ điều độ. Hệ điều hành sẽ giám sát và tổ chức thực hiện các sơ đồ điều độ này. Cho dù nhiệm vụ điều độ là của thành phần nào, thì tất cả phải đạt được các yêu cầu sau:

1. Tại một thời điểm không thể có hai tiến trình nằm trong đoạn găng.



2. Nếu có nhiều tiến trình đồng thời cùng xin được vào đoạn găng thì chỉ có một tiến trình được phép vào đoạn găng, các tiến trình khác phải xếp hàng chờ trong hàng đợi.
3. Tiến trình chờ ngoài đoạn găng không được ngăn cản các tiến trình khác vào đoạn găng.
4. Không có tiến trình nào được phép ở lâu vô hạn trong đoạn găng và không có tiến trình phải chờ lâu mới được vào đoạn găng (chờ trong hàng đợi).
5. Nếu tài nguyên găng được giải phóng thì hệ điều hành có nhiệm vụ đánh thức các tiến trình trong hàng đợi ra để tạo điều kiện cho nó vào đoạn găng.

Trước khi tìm hiểu về các giải pháp điều độ tiến trình qua đoạn găng chúng ta cần lưu ý một lần nữa rằng: nguyên lý cơ bản của điều độ là tổ chức truy xuất độc quyền trên tài nguyên găng, nhưng sự bắt buộc độc quyền này còn tồn tại hai hạn chế lớn:

1. Có thể dẫn đến tắc nghẽn (Deadlock) trong hệ thống. Chúng ta sẽ tìm hiểu về tắc nghẽn sau, bây giờ chúng ta hãy xem một ví dụ về tắc nghẽn: Giả như có hai tiến trình P1 và P2, và hai tài nguyên găng R1 và R2, mỗi tiến trình đều cần truy xuất đến để mã thực hiện một hàm của nó. Và trường hợp sau đây hoàn toàn có thể xảy ra: R1 đang được giao cho P2, R2 được giao cho P1. Mỗi tiến trình đều chờ đợi được sử dụng tài nguyên thứ hai. Không một tiến trình nào giải phóng tài nguyên mà nó đang sở hữu cho đến khi có nhận được tài nguyên còn lại và thực hiện đoạn găng của nó. Cả hai tiến trình đó đều bị tắc nghẽn.

2. Các tiến trình có thể bị đói (Starvation) tài nguyên: Ví dụ sau đây cho thấy sự đói tài nguyên của các tiến trình trên hệ thống: Giả sử rằng có 3 tiến trình P1, P2, P3, mỗi tiến trình đều cần truy xuất định kỳ đến tài nguyên R. Xét trường hợp P1 đang sở hữu tài nguyên còn hai tiến trình P2, P3 phải chờ đợi tài nguyên đó. Khi mà P1 thoát khỏi đoạn găng của nó, cả P2 lẫn P3 đều có thể được chấp nhận truy xuất đến R. Giả sử rằng P3 được truy xuất R, sau đó trước khi P3 kết thúc đoạn găng của nó P1 lại một lần nữa cần truy xuất, và giả như P1 được truy xuất sau khi P3 kết thúc đoạn găng, và nếu như P1, P3 thay nhau nhận được quyền truy xuất thì P2 hầu như không thể truy cập đến tài nguyên, cho dù không có sự tắc nghẽn nào xảy ra.

## **I.17. Điều độ tiến trình qua đoạn găng**

### **II.3.5. Các giải pháp phân cứng**

#### **II.3.2.a. Dùng cặp chỉ thị STI & CLI**

Một số vi xử lý cung cấp cặp chỉ thị CLI và STI để người lập trình thực hiện các

thao tác mở ngắt (STI: Setting Interrupt) và cấm ngắt (CLI: Clean Interrupt) của hệ thống trong lập trình. Người lập trình có thể dùng cặp chỉ thị này để tổ chức điều độ cho các tiến trình như sau: Trước khi vào đoạn găng tiến trình thực hiện chỉ thị CLI, để yêu cầu cấm các ngắt trong hệ thống, khi đó ngắt đồng hồ không thể phát sinh, nghĩa là không có một tiến trình nào khác có thể phát sinh, nhờ đó mà tiến trình trong đoạn găng toàn quyền sử dụng tài nguyên găng cho đến hết thời gian xử lý của nó. Khi kết thúc truy xuất tài nguyên găng, tiến trình ra khỏi đoạn găng, tiến trình thực hiện chỉ thị STI để cho phép ngắt trở lại. Khi đó các tiến trình khác có thể tiếp tục hoạt động và có thể vào đoạn găng.

Trong sơ đồ điều độ này tiến trình  $P_i$  được viết như sau:

Procedure P(i: integer);

Begin

Repeat

**CLI;** {cấm ngắt trước khi vào đoạn găng}

<Đoạn găng của P>;

**STI;** {mở ngắt khi ra khỏi đoạn găng }

<Đoạn không găng>;

Until .F.

End;

{-----}

Sơ đồ trên cho thấy, khi tiến trình ở trong đoạn găng nó không hề bị ngắt, do đã cấm ngắt phát sinh, nên nó được độc quyền sử dụng tài nguyên găng cho đến khi ra khỏi đoạn găng.

Sơ đồ điều độ này đơn giản, dễ cài đặt. Tuy nhiên, cần phải có sự hỗ trợ của vi xử lý và dễ gây ra hiện tượng treo toàn bộ hệ thống, khi tiến trình trong đoạn găng không có khả năng ra khỏi đoạn găng. Tiến trình không ra khỏi đoạn găng nên nó không thể thực hiện chỉ thị STI để mở ngắt cho hệ thống, nên hệ thống bị treo hoàn toàn.

Giải pháp này không thể sử dụng trên các hệ thống multiprocessor, vì CLI chỉ cấm ngắt trên vi xử lý hiện tại chứ không thể cấm ngắt của các vi xử lý khác. Tức là, sau khi đã cấm ngắt, tiến trình trong đoạn găng vẫn có thể bị tranh chấp tài nguyên găng bởi các tiến trình trên các vi xử lý khác trong hệ thống.

### II.3.2.b. Dùng chỉ thị TSL (Test and set)

Trong ví dụ 2 ở trên ta đã thấy, nguyên nhân của lỗi là do hai thao tác kiểm tra tài khoản và rút tiền, bị tách rời nhau. Để tổ chức điều độ cho những trường hợp như vậy, một số vi xử lý cung cấp một chỉ thị đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia được, gọi là Test and

Set lock (TSL). TSL được định nghĩa như sau :

```
Function TestAndSetLock(Var I:Integer):Boolean;  
    Begin  
        IF I = 0 Then  
            Begin  
                I := 1;                {hai lệnh này  
không}                               {thể tách rời}  
                TestAndSetLock:=True;  
            End  
        Else  
            TestAndSetLock := False  
    End;  
    {-----}
```

Để tổ chức điều độ tiến trình với TSL chương trình phải sử dụng biến chia sẻ Lock, khởi gán bằng 0. Theo đó, mỗi tiến trình trước khi vào đoạn găng phải kiểm tra giá trị của Lock. Nếu Lock = 0 thì vào đoạn găng. Nếu Lock = 1 thì phải đợi cho đến khi Lock = 0. Như vậy, trước khi vào đoạn găng tiến trình phải gọi hàm TestAndSetLock, để kiểm tra giá trị trả về của hàm này:

- Nếu bằng False, là đang có một tiến trình trong đoạn găng, thì phải chờ cho đến khi hàm trả về True, có một tiến trình vừa ra khỏi đoạn găng.
- Nếu bằng True, thì tiến trình sẽ vào đoạn găng để sử dụng tài nguyên găng. Khi kết thúc sử dụng tài nguyên găng ra khỏi đoạn găng thì tiến trình phải đặt lại giá trị của Lock, Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Nên nhớ rằng TestAndSetLock là chỉ thị của processor, nên hệ thống đã tổ chức thực hiện độc quyền cho nó. Tức là, các thao tác mà hệ thống phải thực hiện trong chỉ thị này là không thể tách rời nhau.

Trong sơ đồ điều độ này tiến trình P được viết như sau:

```
Procedure P(Lock: integer);  
    Begin  
        Repeat  
            While (TestAndSetlock(lock)) DO;  
                <Đoạn găng của P>;  
                Lock:= 0;  
                <Đoạn không găng>;  
        Until .F.  
    End;  
    {-----}
```

Sơ đồ này đơn giản, dễ cài đặt nhưng cần phải có sự hỗ trợ của vi xử lý. Ngoài ra nó còn một hạn chế lớn là gây lãng phí thời gian xử lý của processor do tồn tại hiện tượng chờ đợi tích cực trong sơ đồ (While (TestAndSetlock(lock)) DO;). Hiện tượng chờ đợi tích cực là hiện tượng processor chỉ chờ một sự kiện nào đó xảy ra mà không làm gì cả.

➤ Tóm lại: Việc sử dụng các chỉ thị phần cứng đặc biệt để tổ chức điều độ tiến trình qua đoạn găng, hay còn gọi là tổ chức truy xuất độc quyền trên tài nguyên găng, có những thuận lợi và bất lợi sau đây:

**Thuận lợi:**

- Nó thích hợp với một số lượng bất kỳ các tiến trình cả trên hệ thống Uniprocessor và hệ thống Multiprocessor.
- Nó khá đơn giản cho nên dễ xác định độ chính xác.
- Nó có thể được sử dụng để hỗ trợ cho nhiều đoạn găng; mỗi đoạn găng có thể định nghĩa cho nó một biến riêng.

**Bất lợi:**

- Trong khi một tiến trình đang chờ đợi được vào đoạn găng thì nó tiếp tục làm tốn thời gian xử lý của processor, mà ta gọi là chờ đợi tích cực.
- Sự đói tài nguyên có thể xảy ra. Khi một tiến trình rời khỏi một đoạn găng, bộ phận điều độ tiến trình phải chọn một tiến trình trong số nhiều tiến trình ngoài đoạn găng để cho nó vào đoạn găng. Việc chọn này có thể dẫn đến hiện tượng có một tiến trình đợi mãi mà không thể vào đoạn găng được.
- Sự tắc nghẽn có thể xảy ra. Hãy xét một tình huống trên một hệ thống uniprocessor. Tiến trình P1 thực thi chỉ thị đặc biệt (TestAndSetLock, Exchange) và vào đoạn găng của nó. P1 sau đó bị ngắt để nhường processor cho P2, P2 là tiến trình có độ ưu tiên cao hơn. Nếu như P2 cũng định sử dụng tài nguyên như P1, P2 sẽ bị từ chối truy xuất bởi vì cơ chế độc quyền. Do đó P2 sẽ đi vào vòng lặp busy-waiting. Tuy nhiên, P1 sẽ không bao giờ được cấp processor để tiếp tục vì nó có độ ưu tiên thấp hơn so với P2.

## **II.3.6. Các giải pháp dùng biến khoá**

### **II.3.3.a. Dùng biến khoá chung**

Xuất phát từ nguyên tắc cơ bản của tổ chức độc quyền là, tại mỗi thời điểm chỉ có duy nhất một tiến trình có thể truy xuất đến một vùng nhớ chia sẻ, các hệ điều hành sử dụng biến khoá chung để tổ chức truy xuất độc quyền trên tài nguyên găng. Phương pháp này còn gọi là phương pháp Busy and Waiting (bận và đợi), nó được nhà toán học người Hà Lan tên là Dekker đề xuất.

Với mỗi tài nguyên găng, hệ điều hành dùng một biến chung để điều khiển việc sử dụng tài nguyên này của các tiến trình đồng thời. Tạm gọi là biến chung này là Lock, Lock được chia sẻ cho nhiều tiến trình và được khởi gán = 0.

Theo đó, mỗi tiến trình trước khi vào đoạn găng phải kiểm tra giá trị của Lock:

- Nếu Lock = 1, tức là đã có tiến trình nào đó trong đoạn găng, thì tiến trình phải chờ cho đến khi Lock = 0 (có thể chuyển sang trạng thái blocked để chờ).
- Nếu Lock = 0, tức là không có tiến trình nào trong đoạn găng, thì tiến trình thiết lập quyền vào đoạn găng, đặt Lock = 1, và vào đoạn găng. Tiến trình vừa ra khỏi đoạn găng phải đặt Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Trong sơ đồ điều độ này tiến trình P được viết như sau:

Procedure P(Lock: integer);

Begin

Repeat

```
While Lock = 1 DO ; {đợi cho đến khi Lock = 0}
Lock :=1;           {thiết lập quyền vào đoạn găng}
<Đoạn găng của P>; {vào đoạn găng}
Lock:= 0;           {thông báo là đã rời đoạn găng }
<Đoạn không găng>;
```

Until .F.

End;

{-----}

Sơ đồ điều độ dùng biến khoá chung này đơn giản, dễ xây dựng nhưng vẫn xuất hiện hiện tượng chờ đợi tích cực, khi chờ cho đến khi Lock = 0 (While Lock = 1 DO;). Hiện tượng chờ đợi tích cực gây lãng phí thời gian của processor.

Nếu một tiến trình trong đoạn găng không thể ra khỏi đoạn găng, thì các tiến trình chờ ngoài đoạn găng có thể chờ đợi vô hạn (vì Lock không được đặt lại = 0).

### II.3.3.b. Dùng biến khoá riêng

Để khắc phục hạn chế của phương pháp dùng biến chung, các hệ điều hành có thể dùng giải pháp biến riêng để tổ chức điều độ tiến trình. Mỗi tiến trình sử dụng một biến khoá Lock riêng, tương ứng với một tài nguyên găng trong hệ thống. Biến khoá riêng của tất cả các tiến trình đều được khởi gán bằng 0, tức là chưa vào đoạn găng

Theo đó, mỗi tiến trình trước khi vào đoạn găng ứng với một tài nguyên găng nào đó thì trước hết phải kiểm tra biến khoá riêng, tương ứng với tài nguyên găng mà tiến trình muốn truy xuất, của tất cả các tiến trình còn lại:

- Nếu tồn tại một biến khoá riêng của một tiến trình nào đó bằng 1, Lock

= 1, tức là đã có một tiến trình nào đó ở trong đoạn găng, thì tiến trình phải chờ ngoài đoạn găng cho đến khi tất cả biến khoá riêng = 0.

- Nếu tất cả các biến khoá riêng của các tiến trình đều = 0, Lock = 0, tức là không có tiến trình nào trong đoạn găng, thì tiến trình thiết lập quyền vào đoạn găng, đặt Lock = 1, và vào đoạn găng. Tiến trình vừa ra khỏi đoạn găng phải đặt Lock = 0, để các tiến trình khác có thể vào đoạn găng.

Sau đây là sơ đồ điều độ dùng biến khoá riêng cho hai tiến trình đồng thời P1 và P2. Hai tiến trình này dùng hai biến khoá riêng là Lock1 và Lock2:

```
Program MutualExclusion;
Const      N:2;
Var
    Lock1, Lock2: byte;
BEGIN
    Lock1 = 0; Lock2 = 0;
ParBegin
    P1: Repeat                                {tiến trình P1}
        While Lock2 = 1 Do ; {P2 đang ở trong đoạn găng }
        Lock1 := 1;          {P1 thiết lập quyền vào đoạn găng}
        <Đoạn găng của P1>;
        Lock1 := 0;          {P1 ra khỏi đoạn găng}
        <Đoạn không găng của P1>;
    Until .F.
    P2: Repeat                                {tiến trình P2}
        While Lock1 = 1 Do; {P1 đang ở trong đoạn găng }
        Lock2 := 1;          {P2 thiết lập quyền vào đoạn găng}
        <Đoạn găng của P2>;
        Lock2 := 0;          {P2 ra khỏi đoạn găng}
        <Đoạn không găng của P2>;
    Until .F.
ParEnd
END.
{-----}
```

Sơ đồ này đơn giản dễ cài đặt. Một tiến trình nào đó ở ngoài đoạn găng bị blocked sẽ không ngăn cản được các tiến trình khác vào đoạn găng, nhưng nếu tiến trình trong đoạn găng bị lỗi không thể ra khỏi đoạn găng, Lock luôn luôn = 0, thì

các tiến trình khác sẽ không được quyền vào đoạn găng.

Phương pháp này vẫn còn tồn tại hiện tượng chờ đợi tích cực và sơ đồ điều độ sẽ trở nên phức tạp khi có nhiều hơn hai tiến trình muốn vào đoạn găng.

Sơ đồ này có thể xảy ra một lỗi nghiêm trọng đó là: Có thể có hai tiến trình cùng nằm trong đoạn găng. Nguyên nhân của lỗi này là do việc kiểm tra quyền vào đoạn găng và việc xác lập quyền vào đoạn găng của tiến trình bị tách rời khi thực hiện. Tức là, P1 và P2 có thể bị điều phối thực hiện theo thứ tự sau:

1. P1 được cấp processor: P1 thực thi vòng lặp While và tìm xem thử Lock2 = 1 không. Khi P1 vừa nhìn thấy Lock2 = 0, thì bị thu hồi processor.
2. P2 được cấp processor: P2 thực thi vòng lặp While và tìm xem thử Lock1 = 1 không. Khi P2 vừa nhìn thấy Lock1 = 0, thì bị thu hồi processor.
3. P1 được cấp processor trở lại: P1 không kiểm tra lại Lock2 mà chỉ đặt Lock1 = 1 và vào đoạn găng của nó. Khi vừa vào đoạn găng thì bị thu hồi processor.
4. P2 được cấp processor trở lại: P2 không kiểm tra lại Lock1 mà chỉ đặt Lock2 = 1 và vào đoạn găng của nó.

Rõ ràng với thực tế này thì cả P1 và P2 đều nằm trong đoạn găng. Và chúng ta đã biết điều gì sẽ xảy ra khi hai tiến trình đồng thời truy xuất tài nguyên găng trong các ví dụ về tài nguyên găng ở trên.

Nhiều nhà thiết kế hệ điều hành đã cải tiến sơ đồ điều độ ở trên để khắc phục hạn chế trên đây và một số hạn chế khác của nó.

### II.3.7. Các giải pháp được hỗ trợ bởi hệ điều hành và ngôn ngữ lập trình

Các giải pháp trên tồn tại hiện tượng chờ đợi tích cực, gây lãng phí thời gian xử lý của processor. Điều này có thể khắc phục bằng một nguyên tắc rất cơ bản: nếu một tiến trình khi chưa đủ điều kiện vào đoạn găng thì được chuyển ngay sang trạng thái blocked để nó trả lại processor cho hệ thống, để hệ thống cấp cho tiến trình khác. Để thực hiện được điều này cần phải có sự hỗ trợ của hệ điều hành và các ngôn ngữ lập trình để các tiến trình có thể chuyển trạng thái của nó. Hai thủ tục **Sleep** và **Wakeup** được hệ điều hành cung cấp để sử dụng cho mục đích này:

- Khi tiến trình chưa đủ điều kiện vào đoạn găng nó sẽ thực hiện một lời gọi hệ thống để gọi Sleep để chuyển nó sang trạng thái blocked, và tiến trình được gọi này đưa vào hàng đợi để đợi cho đến khi có một tiến trình khác gọi thủ tục Wakeup để giải phóng nó ra khỏi hàng đợi và có thể đưa nó vào đoạn găng.
- Một tiến trình khi ra khỏi đoạn găng phải gọi Wakeup để đánh thức một tiến trình trong hàng đợi blocked ra để tạo điều kiện cho tiến trình này vào đoạn găng.

Như vậy giải pháp này được áp dụng trên nhóm các tiến trình hoạt động đồng thời có trao đổi thông tin với nhau, và các tiến trình phải hợp tác với nhau để hoàn thành nhiệm vụ. Các tiến trình này liên lạc với nhau bằng cách gửi tín hiệu cho nhau. Một tiến trình trong hệ thống này có thể bị buộc phải dừng (bị blocked) cho đến khi nhận được một tín hiệu nào đó từ tiến trình bên kia, đó là tiến trình hợp tác với nó.

Thực tế đã chỉ ra được rằng, nếu chỉ dùng hai thủ tục trên thì sơ đồ điều độ sẽ không đáp ứng được các yêu cầu của công tác điều độ, do đó khi cài đặt các hệ điều hành chỉ sử dụng ý tưởng của Sleep và Wakeup. Sau đây là các giải pháp sử dụng ý tưởng của Sleep và Wakeup.

### II.3.3.a. Giải pháp dùng Semaphore (sự đánh tín hiệu bằng cờ) (đèn báo)

Giải pháp này được Dijkstra đề xuất vào năm 1965. Semaphore (sự đánh tín hiệu bằng cờ) được định nghĩa để sử dụng trong các sơ đồ điều độ như sau:

- Semaphore (sự đánh tín hiệu bằng cờ) S là một biến nguyên, khởi gán bằng một giá trị không âm, đó là khả năng phục vụ của tài nguyên gắng tương ứng với nó.
- Ứng với S có một hàng đợi F(s) để lưu các tiến trình đang bị blocked trên S.
- Chỉ có hai thao tác Down và Up được tác động đến semaphore (sự đánh tín hiệu bằng cờ) S. Down giảm S xuống một đơn vị, Up tăng S lên một đơn vị.
- Mỗi tiến trình trước khi vào đoạn găng thì phải gọi Down để kiểm tra và xác lập quyền vào đoạn găng. Khi tiến trình gọi Down(S) thì hệ thống sẽ thực hiện như sau:  $S := S - 1$ , nếu  $S \geq 0$  thì tiến trình tiếp tục xử lý và vào đoạn găng, nếu  $S < 0$  thì tiến trình phải vào hàng đợi để chờ cho đến khi  $S \geq 0$ . Down được cài đặt như sau:

```

Procedure Down(s);
Begin
    S := S - 1;
    If S < 0 Then                                {S ≥ 0 thì tiếp tục}
        Begin
            Status(p) = blocked;                {chuyển tiến trình sang
blocked}
            Enter(p, F(s));                      {đưa tiến trình vào hàng đợi F(S)}
        end;
End;
```

- Mỗi tiến trình ngay sau khi ra khỏi đoạn găng phải gọi Up để kiểm tra



xem có tiến trình nào đang đợi trong hàng đợi hay không, nếu có thì đưa tiến trình trong hàng đợi vào đoạn găng. Khi tiến trình gọi Up thì hệ thống sẽ thực hiện như sau:  $S := S + 1$ , nếu  $S \leq 0$  đưa một tiến trình trong  $F(s)$  vào đoạn găng. Up được cài đặt như sau:

```

Procedure Up(s);
Begin
    S := S + 1;
    If S <= 0 Then
        Begin
            Exit(Q,F(s) );           {đưa tiến trình ra khỏi F(S)}
            Status(Q) = ready ;      {chuyển tiến trình sang ready}
            Enter(Q, ready-list ) ;  {đưa tiến trình vào ready list}
        End;
    End;

```

Sau đây là sơ đồ điều độ dùng Semaphore (sự đánh tín hiệu bằng cờ) cho 2 tiến trình P1 và P2, hai tiến trình này hoạt động đồng thời cùng truy xuất đến tài nguyên găng tương ứng với semaphore (sự đánh tín hiệu bằng cờ) S. Tài nguyên găng này chỉ có thể đáp ứng cho một tiến trình tại một thời điểm nên S được khởi gán bằng 1.

```

Program MutualExclusion;
Const
    n = 2;
Var
    s: semaphore (sự đánh tín hiệu bằng cờ);
    {-----}
Procedure P(i:Integer);
Begin
    Repeat
        Down(s);           {kiểm tra và xác lập quyền vào đoạn găng}
        <Đoạn găng>;
        Up(s);              {rời đoạn găng và kích hoạt tiến trình
khác}
        <Đoạn không găng>;
    Until .F.;
End;
{-----}
BEGIN
    S := 1;

```

```

ParBegin
    P(1);
    P(2);
ParEnd;
END.
{-----}

```

Ở đây chúng ta cần lưu ý rằng: Down và Up là các thủ tục của hệ điều hành, nên hệ điều hành đã cài đặt cơ chế độc quyền cho nó, tức là các lệnh bên trong nó không thể tách rời nhau. Nếu điều này không được thực hiện thì sơ đồ này trở nên vô nghĩa. Hai thủ tục Down(S) và Up(S) mà chúng tôi đưa ra ở trên chỉ để minh họa cho nguyên lý hoạt động của Down và Up.

Sử dụng semaphore (sự đánh tín hiệu bằng cờ) để điều độ tiến trình, mang lại những thuận lợi sau:

- Mỗi tiến trình chỉ kiểm tra quyền vào đoạn găng một lần, khi chờ nó không làm gì cả, tiến trình ra khỏi đoạn găng phải đánh thức nó.
- Không xuất hiện hiện tượng chờ đợi tích cực, nên khai thác tối đa thời gian xử lý của processor.
- Nhờ cơ chế hàng đợi mà hệ điều hành có thể thực hiện gán độ ưu tiên cho các tiến trình khi chúng ở trong hàng đợi.
- Trị tuyệt đối của S cho biết số lượng các tiến trình đang đợi trên F(S).

Nên nhớ rằng, Down và Up là các thủ tục của hệ điều hành nên sơ đồ điều độ sẽ bị thay đổi khi thay đổi hệ điều hành. Đây là một trở ngại của việc sử dụng semaphore (sự đánh tín hiệu bằng cờ) để tổ chức điều độ tiến trình.

**Các ví dụ sau đây thay cho sự giải thích về sơ đồ điều độ ở trên:**

**Ví dụ 1:** Sự thực hiện của hai tiến trình P1 và P2 trong sơ đồ điều độ trên

P thực hiện	Down/Up	S	Trạng thái của P1/P2
		<b>1</b>	
1. <b>P1</b>	Down(S)	0	P1 hoạt động
2. <b>P2</b>	Down(S)	-1	P2 chờ
3. <b>P1</b>	Up(S)	0	P2 hoạt động
4. <b>P1</b>	Down(S)	-1	P1 chờ
5. <b>P2</b>	Down(S)	0	P1 hoạt động

**Ví dụ 2:** Nếu trong hệ thống có 6 tiến trình hoạt động đồng thời, cùng sử dụng tài nguyên găng, tài nguyên găng này chỉ cho phép một tiến trình truy xuất

đến nó tại một thời điểm. Tức là hệ điều hành phải tổ chức truy xuất độc quyền trên tài nguyên găng này. Thứ tự yêu cầu sử dụng tài nguyên găng của các tiến trình, cùng với thời gian mà tiến trình cần processor khi nó ở trong đoạn găng (cần tài nguyên găng) và độ ưu tiên của các tiến trình, được mô tả như sau:

- Có 6 tiến trình yêu cầu sử dụng tài nguyên găng tương ứng với S lần lượt là:  

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>
----------	----------	----------	----------	----------	----------
- Độ ưu tiên của các tiến trình là (5 là độ ưu tiên cao nhất):  

1	1	2	4	2	5
---	---	---	---	---	---
- Thời gian các tiến trình cần sử dụng tài nguyên găng là:  

4	2	2	2	1	1
---	---	---	---	---	---

Nếu dùng sơ đồ điều độ semaphore (sự đánh tín hiệu bằng cờ) ở trên để tổ chức điều độ cho 6 tiến trình này thì ta có được bảng mô tả sự thực hiện của các tiến trình A, B, C, D, E, F như sau:

T	Down/ Up	Tiến trình thực hiện	S	Tiến trình hoạt động	Các tiến trình trong hàng đợi
0	-	-	<b>1</b>	-	-
1	Down	A	0	A	-
2	Down	B	-1	A	B
3	Down	C	-2	A	C   B
4	Down	D	-3	A	D   C   B
5	Up	A	-2	D	C   B
6	Down	E	-3	D	C   E   B
7	Up	D	-2	C	E   B
8	Down	F	-3	C	F   E   B
9	Up	C	-2	F	E   B
10	Up	F	-1	E	B
11	Up	E	0	B	-
12	Up	B	<b>1</b>	-	-

Bảng trên lưu ý với chúng ta hai điều. Thứ nhất, trị tuyệt đối của S cho biết số lượng các tiến trình trong hành đợi F(S). Thứ hai, tiến trình chưa được vào đoạn găng thì được đưa vào hàng đợi và tiến trình ra khỏi đoạn găng sẽ đánh thức tiến trình có độ ưu tiên cao nhất trong hành đợi để đưa nó vào đoạn găng. Tiến trình được đưa vào hàng đợi sau nhưng có độ ưu tiên cao hơn sẽ được đưa vào đoạn găng trước các tiến trình được đưa vào hàng đợi trước nó.

### II.3.3.b. Giải pháp dùng Monitors

Giải pháp này được Hoar đề xuất năm 1974 sau đó vào năm 1975 được Brinch & Hanssen đề xuất lại. Monitor là cấu trúc phần mềm đặc biệt được cung cấp bởi ngôn ngữ lập trình, nó bao gồm các thủ tục, các biến và các cấu trúc dữ liệu được định nghĩa bởi Monitor. Monitor được định nghĩa trong các ngôn ngữ lập trình như pascal+, Modula-2, Modula-3. Monitor của Hoar có các tính chất sau đây:

1. Các biến và cấu trúc dữ liệu bên trong monitor chỉ có thể được thao tác bởi các thủ tục được định nghĩa bên trong monitor đó.
2. Một tiến trình muốn vào monitor phải gọi một thủ tục của monitor đó.
3. Tại một thời điểm, chỉ có một tiến trình duy nhất được hoạt động bên trong monitor. Các tiến trình khác đã gọi monitor phải hoãn lại để chờ monitor rảnh.

Hai tính chất 1 và 2 tương tự như các tính chất của các đối tượng trong lập trình hướng đối tượng. Như vậy một hệ điều hành hoặc một ngôn ngữ lập trình hướng đối tượng có thể cài đặt monitor như là một đối tượng có các tính chất đặc biệt.

Với tính chất thứ 3 monitor có khả năng thực hiện các cơ chế độc quyền, các biến trong monitor có thể được truy xuất chỉ bởi một tiến trình tại một thời điểm. Như vậy các cấu trúc dữ liệu dùng chung bởi các tiến trình có thể được bảo vệ bằng cách đặt chúng bên trong monitor. Nếu dữ liệu bên trong monitor là tài nguyên căng thì monitor cung cấp sự độc quyền trong việc truy xuất đến tài nguyên căng đó.

Monitor cung cấp các công cụ đồng bộ hoá để người lập trình sử dụng trong các sơ đồ điều độ. Công cụ đồng bộ hoá được định nghĩa để sử dụng trong các sơ đồ điều độ như sau: Trong một monitor, có thể định nghĩa các *biến điều kiện* và hai thao tác kèm theo là Wait và Signal, chỉ có wait và signal được tác động đến các biến điều kiện.

- Giả sử C là biến điều kiện được định nghĩa trong monitor.
- **Wait(c)**: khi một tiến trình gọi wait, thì wait sẽ chuyển tiến trình gọi sang trạng thái blocked, và đặt tiến trình này vào hàng đợi trên biến điều kiện c. Wait được cài đặt như sau:

```
Procedure Wait(c);  
Begin  
    Status(p) = blocked;  
    Enter(p,f(c));  
End;
```

- **Signal(c)**: khi một tiến trình gọi signal, thì signal sẽ kiểm tra trong hàng

đội của c có tiến trình nào hay không, nếu có thì tái kích hoạt tiến trình đó, và tiến trình gọi signal sẽ rời khỏi monitor. Signal được cài đặt như sau:

```

Procedure Signal(c);
Begin
    If f(c) <> Null Then
        Begin
            Exit(Q,f(c));      {Q là tiến trình chờ trên C}
            Status(Q) = ready;
            Enter(Q,ready-lits);
        end;
    End,

```

Trình biên dịch chịu trách nhiệm thực hiện việc truy xuất độc quyền đến dữ liệu trong monitor. Để thực hiện điều này, hệ điều hành dùng một semaphore (sự đánh tín hiệu bằng cờ) nhị phân. Mỗi monitor có một hàng đợi toàn cục lưu các tiến trình đang chờ được vào monitor, ngoài ra mỗi biến điều kiện c cũng gắn với một hàng đợi F(c).

Với mỗi nhóm tài nguyên găng, có thể định nghĩa một monitor trong đó đặc tả tất cả các thao tác trên tài nguyên này với một số điều kiện nào đó. Sau đây là cấu trúc một Monitor. Sau đây là cấu trúc của monitor:

```

Monitor    <Tên monitor>
Condition  <Danh sách các biến điều kiện>;
{-----}
    Procure  Action1();      {thao tác i}
    Begin
        .....
    End;
    {-----}
    Procedure Actionn();      {thao tác n}
    Begin
        .....
    End;
    {-----}
End monitor;

```

Mỗi tiến trình muốn sử dụng tài nguyên găng chỉ có thể thao tác thông qua các thủ tục bên trong monitor.

Sau đây là sơ đồ điều độ sử dụng monitor cho 2 tiến trình P1 và P2.

```
Program MutualExclusion;  
Monitor ..... Endmonitor;           {monitor được định nghĩa như trên}  
{-----}  
BEGIN  
    ParBegin  
        P1: Repeat  
            <Đoạn không găng của P1>;  
            <monitor>.Actioni;           {Đoạn găng của P1};  
            <Đoạn không găng của P1>;  
        Until .F.  
        P2: Repeat  
            <Đoạn không găng của P2>;  
            <monitor>.Actionj;           {Đoạn găng của P2};  
            <Đoạn không găng của P2>;  
        Until .F.  
    Parend  
END.  
{-----}
```

Với monitor, việc tổ chức truy xuất độc quyền được trình biên dịch thực hiện, nên nó đơn giản hơn cho người lập trình. Tuy nhiên hiện nay rất ít ngôn ngữ lập trình hỗ trợ cấu trúc monitor cho lập trình.

### II.3.3.c. Giải pháp trao đổi Message (thông điệp)

Khi các tiến trình có sự tương tác với tiến trình khác, hai yêu cầu cơ bản cần phải được thỏa mãn đó là: sự đồng bộ hoá (synchronization) và sự truyền thông (communication). Các tiến trình phải được đồng bộ để thực hiện độc quyền. Các tiến trình hợp tác có thể cần phải trao đổi thông tin. Một hướng tiếp cận để cung cấp cả hai chức năng đó là sự truyền thông điệp (message passing). Truyền thông điệp có ưu điểm là có thể thực hiện được trên cả hai hệ thống uniprocessor và multiprocessor, khi các hệ thống này hoạt động trên mô hình bộ nhớ chia sẻ

Các hệ thống truyền thông điệp có thể có nhiều dạng. Trong phần này chúng tôi giới thiệu một dạng chung nhất mà trong đó đề cập đến các đặc trưng có trong nhiều hệ thống khác nhau. Các hàm của truyền thông điệp trên thực tế có dạng tương tự như hai hàm sau:

- **Send(destination, message):** gửi thông điệp đến tiến trình đích.
- **Receive(source, message):** nhận thông điệp từ tiến trình nguồn.

Một tiến trình gửi thông tin dưới dạng một thông điệp (message) đến một tiến trình khác, bằng hàm Send, được nhận biết bởi tham số destination. Một tiến trình nhận thông điệp (message), bằng hàm Receive, từ một tiến trình được nhận biết bởi tham số source. Tiến trình gọi Receive phải chờ cho đến khi nhận được message từ tiến trình source thì mới có thể tiếp tục được.

Việc sử dụng Send và Receive để tổ chức điều độ được thực hiện như sau:

- Có một tiến trình kiểm soát việc sử dụng tài nguyên căng.
- Có nhiều tiến trình khác yêu cầu sử dụng tài nguyên căng này.
- Tiến trình có yêu cầu tài nguyên căng sẽ gửi một thông điệp đến tiến trình kiểm soát và sau đó chuyển sang trạng thái blocked cho đến khi nhận được một thông điệp chấp nhận cho truy xuất từ tiến trình kiểm soát tài nguyên căng.
- Khi sử dụng xong tài nguyên căng, tiến trình vừa sử dụng tài nguyên căng gửi một thông điệp khác đến tiến trình kiểm soát để báo kết thúc truy xuất.
- Tiến trình kiểm soát, khi nhận được thông điệp yêu cầu tài nguyên căng, nó sẽ chờ cho đến khi tài nguyên căng sẵn sàng để cấp phát thì gửi một thông điệp đến tiến trình đang bị khoá trên tài nguyên đó để đánh thức tiến trình này.

Trong sơ đồ điều độ dùng message tiến trình P được viết như sau:

Procedure P(i: Integer);

Begin

**Repeat**

Send(process controler, request message);

**Receive(process controler, accept message );**

<Đoạn căng của P>;

Send(process controler ,end message);

<Đoạn không căng của P>;

**Until .F.**

End;

{-----}

Giải pháp này thường được cài đặt trên các hệ thống mạng máy tính, đặc biệt là trên các hệ thống mạng phân tán. Đây là lợi thế mà semaphore (sự đánh tín hiệu bằng cờ) và monitor không có được.

Sơ đồ điều độ dùng message phải chú ý sự đồng bộ giữa các tiến trình nhận và gửi message, nếu không các tiến trình này sẽ không thoát khỏi trạng thái

blocked để tiếp tục được. Điều này cũng có nghĩa là công tác điều độ có thể không thành công mặc dù sơ đồ điều độ đã được tổ chức rất tốt. Sau đây chúng ta sẽ xem xét về sự đồng bộ giữ tiến trình send và tiến trình receiver trong trường hợp này.

### II.3.8. Hai bài toán điều phối làm ví dụ

➤ **Bài toán 1:** Giả sử có một ứng dụng, tạm gọi là ứng dụng Producer/Consumer, trong hệ thống đa nhiệm – đa người sử dụng. Ứng dụng này có hai tiến trình chính đó là, tiến trình người sản xuất (Producer) và tiến trình người tiêu thụ (Consumer), hai tiến trình này hoạt động đồng thời với nhau và cùng chia sẻ một bộ đệm (Buffer) có kích thước giới hạn, chỉ có 3 phần tử. Tiến trình Producer tạo ra dữ liệu và đặt vào Buffer, tiến trình Consumer nhận dữ liệu từ Buffer ra để xử lý. Rõ ràng hệ thống này cần phải có các ràng buộc sau:

1. Hai tiến trình Producer và Consumer không được đồng thời truy xuất Buffer (Buffer là tài nguyên găng).
2. Tiến trình Producer không được ghi dữ liệu vào Buffer khi Buffer đã bị đầy.
3. Tiến trình Consumer không được đọc dữ liệu từ Buffer khi Buffer rỗng.

Hãy dùng các giải pháp Semaphore, Monitor, Message để tổ chức điều độ cho các tiến trình Producer và Consumer trong bài toán trên.

#### II.3.4.1. Giải pháp dùng Semaphore (sự đánh tín hiệu bằng cờ)

Với giải pháp này sơ đồ điều độ phải sử dụng 3 Semaphore (sự đánh tín hiệu bằng cờ):

- Full: dùng để theo dõi số chỗ đã có dữ liệu trong bộ đệm, nó được khởi gán bằng 0. Tức là, ban đầu Buffer rỗng.
- Empty: dùng để theo dõi số chỗ còn trống trên bộ đệm, nó được khởi gán bằng 3. Tức là, ban đầu Buffer không chứa một phần tử dữ liệu nào.
- Mutex: dùng để kiểm tra truy xuất đồng thời trên bộ đệm, nó được khởi gán bằng 1. Tức là, chỉ có 1 tiến trình được phép truy xuất buffre.

Sơ đồ điều độ sẽ như sau:

Program Producer/Consumer;

Var Full, Empty, Mutex: Semaphore (sự đánh tín hiệu bằng cờ);

{-----}

Procedure Producer();

Begin

Repeat

< Tạo dữ liệu>;



```

        Down(empty);           {kiểm tra xem buffer còn chỗ trống ?}
        Down(mutex);          {kiểm tra và xác lập quyền truy xuất Buffer}
        <Đặt dữ liệu vào Buffer>;
        Up(mutex);             {kết thúc truy xuất buffer}
        Up(Full);               {đã có 1 phần tử dữ liệu trong Buffer}
    Until .F.
End;
{-----}
Procedure Consumer();
Begin
    Repeat
        Down(full);            {còn phần tử dữ liệu trong Buffer?}
        Down(mutex);          {kiểm tra và xác lập quyền truy xuất Buffer}
        <Nhận dữ liệu từ đệm>;
        Up(mutex);             {kết thúc truy xuất buffer}
        Up(empty);             {đã lấy 1 phần tử dữ liệu trong Buffer}
    Until .F.
End;
{-----}
BEGIN
    Full = 0; Empty = 3; Mutex = 1;
    Producer();
    Consumer();

                                END.
{-----}

```

#### II.3.4.2. Giải pháp dùng Monitor

Với giải pháp này người lập trình phải định nghĩa một monitor, có tên là ProducerConsumer, trong đó có hai thủ tục Enter và Remove, dùng để thao tác trên Buffer. Xử lý của các thủ tục này phụ thuộc vào các biến điều kiện full và empty. Full và Empty được quy định định sử dụng như trong giải pháp semaphore (sự đánh tín hiệu bằng cờ).

**Sơ đồ điều độ sẽ như sau:**

```

Program Producer/Consumer;
Monitor ProducerConsumer;
Condition Full, Empty;

```

```

Var    Count: Integer;    {để đếm số phần tử dữ liệu được đưa vào Buffer}
      N: Integer;        {số phần tử của Buffer}
{-----}
Procedure Enter();
Begin
    If Count = N Then Wait(Full); {nếu Buffer đầy thì đợi }
    <Đặt dữ liệu vào đệm>;        {Buffer rỗng}
    Count := Count + 1;
    If Count = 1 Then Signal(Empty); {nếu Buffer không rỗng thì}
End;                                {báo cho consumer biết}
{-----}
Procedure Remove();
Begin
    If Count = 0 Then Wait(Empty); {nếu Buffer rỗng thì đợi đầy}
    <Nhận dữ liệu từ đệm>;
    Count := Count - 1;
    If Count = N - 1 Then Signal(Full); {nếu Buffer không đầyf
thì}
End;                                {báo cho producer}
                                Endmonitor;
{-----}
BEGIN
    Count = 0; N = 3;
ParBegin
Procedure Producer();
Begin
    Repeat
        <Tạo dữ liệu>;
        Producer/Consumer.Enter;
    Until .F.
End;
{-----}
Procedure Consumor();
Begin
    Repeat

```

```

        Producer/Consumer.Remove;
        <Xử lý dữ liệu>;
    Until .F.
End;
Parend
END.
{-----}

```

### II.3.4.3. Giải pháp dùng Message

Với giải pháp này chương trình dùng thông điệp empty. Empty hàm ý có một chỗ trống. Buffer. Khi khởi tạo tiến trình Consumer gửi ngay N thông điệp empty đến tiến trình Producer. Tiến trình Producer tạo ra một dữ liệu mới và chờ đến khi nhận được một thông điệp empty từ consumer thì gửi ngược lại cho Consumer một thông điệp có chứa dữ liệu mà nó tạo ra. Sau khi gửi đi thông điệp Emtry, tiến trình consumer sẽ chờ để nhận thông điệp chứa dữ liệu từ tiến trình producer. Sau khi xử lý xong dữ liệu thì consumer gửi lại một thông điệp empty đến tiến trình producer.

**Sơ đồ điều độ sẽ như sau:**

```

Program   Producer/Consumer;
Var
    Buffersize: integer;           {kích thước Buffer}
    M, m': Message;
{ -----}
BEGIN
    Buffersize = N;
ParBegin
    Procedure Producer();
    Begin
        Repeat
            <Tạo dữ liệu>;
            Receive(Consumer,m);
            <Tạo thông điệp dữ liệu>
            Send(Consumer,m)
        Until .F.
    End;
{ -----}
    Procedure Consumer ()

```

```

Var I:integer;
Begin
    For I := 0 to N Do Send(Producer ,m);
    Repeat
        Receive(Producer ,m);
        <Lấy dữ liệu từ thông điệp>
        Send (Producer,m);
        <Xử lý dữ liệu >
    Until .F.
End.
Parend
END.

```

{-----}

➤ **Bài toán 2:** Trong môi trường hệ điều hành đa nhiệm, có thể tồn tại các file chia sẻ, có thể là các file cơ sở dữ liệu. Nhiều tiến trình hoạt động đồng thời trong hệ thống có thể được chia sẻ sử dụng một file cơ sở dữ liệu này. Tiến trình cần đọc nội dung của file cơ sở dữ liệu được gọi là tiến trình Reader. Tiến trình cần cập nhật thông tin vào file cơ sở dữ liệu được gọi là tiến trình Writer. Trong hệ thống này, công tác điều độ tiến trình cần phải thực hiện các ràng buộc sau:

1. Có thể có nhiều tiến trình Reader đồng thời đọc file cơ sở dữ liệu.
2. Không cho phép một tiến trình Writer ghi vào cơ sở dữ liệu khi các tiến trình Reader khác đang đọc cơ sở dữ liệu.
3. Chỉ có duy nhất một tiến trình Writer được phép ghi vào file cơ sở dữ liệu

Hãy dùng các giải pháp Semaphore, Monitor, Message để tổ chức điều độ cho các tiến trình Reader và Writer trong bài toán ở trên.

#### **II.3.4.4. Giải pháp dùng Semaphore (sự đánh tín hiệu bằng cờ)**

Giải pháp này sử dụng một biến chung RC và hai semaphore (sự đánh tín hiệu bằng cờ) là Mutex và DB.

- RC (readcount) dùng để ghi nhận số lượng các tiến trình Reader muốn truy xuất file cơ sở dữ liệu, khởi gán bằng 0.
- Mutex: dùng để kiểm soát truy xuất đến RC, khởi gán bằng 1.
- DB: dùng để kiểm tra sự truy xuất độc quyền đến cơ sở dữ liệu, khởi gán bằng 1.

**Sau đây là sơ đồ điều độ:**

Program    Producer/Consumer;

Const

    Mutex: Semaphore = 1;

    Db     : Semaphore = 1;

    Rc     : byte = 0;

{-----}

BEGIN

**ParBegin**

Procedure Reader();

Begin

    Repeat

        Down(mutex);

        Rc = Rc+1;

        If Rc = 1 then Down(db);

        Up(mutex);

{chấm dứt truy xuất Rc}

        <Đọc dữ liệu >;

        Down(mutex)

        Rc = Rc-1

        If Rc = 0 then Up(db);

        Up(mutex);

        < Xử lý dữ liệu đọc được >

    Until .F.

End;

{-----}

Procedure Writer();

Begin

    Repeat

        <Tạo dữ liệu >;

        Down(Db);

        <cập nhận dữ liệu >

        Up(db);

    Until .F.

End;

**ParEnd**

End.

{-----}

#### II.3.4.5. Giải pháp dùng Monitor

Giải pháp này sử dụng một biến chung RC, để ghi nhận số lượng các tiến trình reader muốn truy xuất cơ sở dữ liệu. Tiến trình Writer phải chuyển sang trạng thái khoá nếu  $RC > 0$ . Khi ra khỏi đoạn găng tiến trình Reader cuối cùng sẽ đánh thức tiến trình Write đang bị khoá.

**Sau đây là sơ đồ điều độ:**

Program    Producer/Consumer;

Monitor    Readerwriter

Condition Okwrite, Okread

Var

    Rc: integer;

    Busy: boolean = False;

{-----}

Procedure Beginread()

Begin

    If (busy) then wait(okread);

    Rc = Rc + 1;

    Signal(okread);

End;

Procedure Finishread()

Begin

    Rc = Rc - 1;

    If Rc = 0 Then Wait(okwrite);

End;

Procedure Beginwrite();

Begin

    Rc = Rc - 1;

    If (busy) or (Rc <> 0) Then Wait(okwrite);

    Busy = True;

End;

Procedure FinishWrite()

Begin

    Busy = False;

```

        If (Okread) Then Signal(okread)
        Else Signal(okwrite);
    End;
Endmonitor.
{-----}
BEGIN
ParBegin
    Procedure Reader ();
    Begin
        Repeat
            ReaderWriter.BeginRead();
            <đọc dữ liệu>
            ReaderWriter.FinishRead();
        Until .F.
    End;
    Procedure Writer ();
    Begin
        Repeat
            ReaderWriter.BeginWrite();
            <đọc dữ liệu>
            ReaderWriter.FinishWrite();
        Until .F.
    End;
Parend
END.
{-----}

```

#### II.3.4.6. Giải pháp dùng Message

Giải pháp này cần phải có một tiến trình Sever điều khiển việc truy xuất cơ sở dữ liệu. Các tiến trình Writer và Reader gửi các thông điệp yêu cầu truy xuất đến server và nhận từ Sever các thông điệp hồi đáp tương ứng.

**Sơ đồ điều độ sẽ như sau:**

```

Program Producer/Consumer;
Begin
    ParBegin

```

```

Procedure Reader();
Begin
    Repeat
        Send (Sever,Requesread);
        Receive(sever,value);
        Print(value);
    Until .F.
End;
Procedure Writer();
Begin
    Repeat
        <Tạo dữ liệu>;
        Send (Sever, Requeswrite,value);
        Receive(sever, okwrite);
    Until .F.
End;
ParEnd
End.
{-----}

```

## I.18. Tắc nghẽn (Deadlock) và chống tắc nghẽn

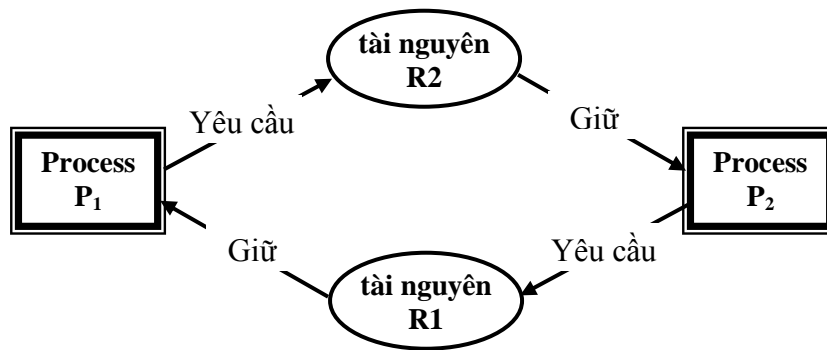
### II.4.5. Tắc nghẽn

Tất cả hiện tượng tắc nghẽn đều bắt nguồn từ sự xung đột về tài nguyên của hai hoặc nhiều tiến trình đang hoạt động đồng thời trên hệ thống. Tài nguyên ở đây có thể là một ổ đĩa, một record trong cơ sở dữ liệu, hay một không gian địa chỉ trên bộ nhớ chính. Sau đây là một số ví dụ để minh họa cho điều trên.

**Ví dụ 1:** Giả sử có hai tiến trình  $P_1$  và  $P_2$  hoạt động đồng thời trong hệ thống. Tiến trình  $P_1$  đang giữ tài nguyên  $R_1$  và xin được cấp  $R_2$  để tiếp tục hoạt động, trong khi đó tiến trình  $P_2$  đang giữ tài nguyên  $R_2$  và xin được cấp  $R_1$  để tiếp tục hoạt động. Trong trường hợp này cả  $P_1$  và  $P_2$  sẽ không tiếp tục hoạt động được. Như vậy  $P_1$  và  $P_2$  rơi vào trạng thái tắc nghẽn. Ví dụ này có thể được minh họa bởi sơ đồ ở hình 2.

Tắc nghẽn thường xảy ra do xung đột về tài nguyên thuộc loại không phân chia được, một số ít trường hợp xảy ra với tài nguyên phân chia được. Ví dụ sau đây là trường hợp tắc nghẽn do xung đột về tài nguyên bộ nhớ, là tài nguyên thuộc loại phân chia được.





**Hình 2.6:** Chờ đợi vòng tròn

**Ví dụ 2:** Giả sử không gian bộ nhớ còn trống là 200Kb, và trong hệ thống có hai tiến trình  $P_1$  và  $P_2$  hoạt động đồng thời.  $P_1$  và  $P_2$  yêu cầu được sử dụng bộ nhớ như sau:

$P_1$	$P_2$
....	....
Request1 80Kb	Request1 70Kb
.....	.....
Request2 30Kb	Request2 40Kb
.....	.....

Tắc nghẽn xảy ra khi cả hai tiến trình cùng yêu cầu thêm bộ nhớ lần thứ hai. Tại thời điểm này không gian bộ nhớ còn trống là 50Kb, lớn hơn lượng bộ nhớ mà mỗi tiến trình yêu cầu (30Kb và 40Kb), nhưng vì cả hai tiến trình đồng thời yêu cầu thêm bộ nhớ, nên hệ thống không thể đáp ứng được, và tắc nghẽn xảy ra.

**Ví dụ 3:** Trong các ứng dụng cơ sở dữ liệu, một chương trình có thể khoá một vài record mà nó sử dụng, để dành quyền điều khiển về cho nó. Nếu tiến trình  $P_1$  khoá record R1, tiến trình  $P_2$  khoá record R2, và rồi sau đó mỗi tiến trình lại cố gắng khoá record của một tiến trình khác. Tắc nghẽn sẽ xảy ra.

Như vậy tắc nghẽn là hiện tượng: Trong hệ thống xuất hiện một tập các tiến trình, mà mỗi tiến trình trong tập này đều chờ được cấp tài nguyên, mà tài nguyên đó đang được một tiến trình trong tập này chiếm giữ. Và sự đợi này có thể kéo dài vô hạn nếu không có sự tác động từ bên ngoài.

Trong trường hợp của ví dụ 1 ở trên: hai tiến trình  $P_1$  và  $P_2$  sẽ rơi vào trạng thái tắc nghẽn, nếu không có sự can thiệp của hệ điều hành. Để phá bỏ tắc nghẽn này hệ điều hành có thể cho tạm dừng tiến trình  $P_1$  để thu hồi lại tài nguyên R1, lấy R1 cấp cho tiến trình  $P_2$  để  $P_2$  hoạt động và kết thúc, sau đó thu hồi cả R1 và R2 từ tiến trình  $P_2$  để cấp cho  $P_1$  và tái kích hoạt  $P_1$  để  $P_1$  hoạt động trở lại. Như vậy sau một khoảng thời gian cả  $P_1$  và  $P_2$  đều ra khỏi tình trạng tắc nghẽn.

Trong trường hợp của ví dụ 2 ở trên: nếu hai tiến trình này không đồng thời yêu cầu thêm bộ nhớ thì tắc nghẽn không thể xảy ra, hoặc khi cả hai tiến trình đồng thời yêu cầu thêm bộ nhớ thì hệ điều hành phải kiểm tra lượng bộ nhớ còn trống của hệ thống, nếu không đáp ứng cho cả hai tiến trình thì hệ điều hành phải có cơ chế ngăn chặn (từ chối) một tiến trình và chỉ cho một tiến trình được quyền sử dụng bộ nhớ (đáp ứng) thì tắc nghẽn cũng không thể xảy ra. Tuy nhiên để giải quyết vấn đề tắc nghẽn do thiếu bộ nhớ, các hệ điều hành thường sử dụng cơ chế bộ nhớ ảo. Bộ nhớ ảo là một phần quan trọng của hệ điều hành mà chúng ta sẽ khảo sát ở chương *Quản lý bộ nhớ* của tài liệu này.

Khi hệ thống xảy ra tắc nghẽn nếu hệ điều hành không kịp thời phá bỏ tắc nghẽn thì hệ thống có thể rơi vào tình trạng treo toàn bộ hệ thống. Như trong trường hợp tắc nghẽn ở ví dụ 1, nếu sau đó có tiến trình  $P_3$ , đang giữ tài nguyên  $R_3$ , cần  $R_2$  để tiếp tục thì  $P_3$  cũng sẽ rơi vào tập tiến trình bị tắc nghẽn, rồi sau đó nếu có tiến trình  $P_4$  cần tài nguyên  $R_1$  và  $R_3$  để tiếp tục thì  $P_4$  cũng rơi vào tập các tiến trình bị tắc nghẽn như  $P_3$ , ... cứ thế dần dần có thể dẫn đến một thời điểm tất cả các tiến trình trong hệ thống đều rơi vào tập tiến trình tắc nghẽn. Và như vậy hệ thống sẽ bị treo hoàn toàn.

#### **II.4.6. Điều kiện hình thành tắc nghẽn**

Năm 1971, Coffman đã đưa ra và chứng tỏ được rằng, nếu hệ thống tồn tại đồng thời bốn điều kiện sau đây thì hệ thống sẽ xảy ra tắc nghẽn:

1. Loại trừ lẫn nhau (mutual exclusion) hay độc quyền sử dụng: Đối với các tài nguyên không phân chia được thì tại mỗi thời điểm chỉ có một tiến trình sử dụng được tài nguyên.
2. Giữ và đợi (hold and wait): Một tiến trình hiện tại đang chiếm giữ tài nguyên, lại xin cấp phát thêm tài nguyên mới.
3. Không ưu tiên (No preemption): Không có tài nguyên nào có thể được giải phóng từ một tiến trình đang chiếm giữ nó.

Trong nhiều trường hợp các điều kiện trên là rất cần thiết đối với hệ thống. Sự thực hiện độc quyền là cần thiết để bảo đảm tính đúng đắn của kết quả và tính toàn vẹn của dữ liệu (chúng ta đã thấy điều này ở phần tài nguyên gắng trên đây). Tương tự, sự ưu tiên không thể thực hiện một cách tùy tiện, đặt biệt đối với các tài nguyên có liên quan với nhau, việc giải phóng từ một tiến trình này có thể ảnh hưởng đến kết quả xử lý của các tiến trình khác.

Sự tắc nghẽn có thể tồn tại với ba điều kiện trên, nhưng cũng có thể không xảy ra chỉ với 3 điều kiện đó. Để chắc chắn tắc nghẽn xảy ra cần phải có điều kiện thứ tư

4. Đợi vòng tròn (Circular wait): Đây là trường hợp của ví dụ 1 mà chúng ta đã nêu ở trên. Tức là, mỗi tiến trình đang chiếm giữ tài nguyên mà tiến

trình khác đang cần.

Ba điều kiện đầu là điều kiện cần chứ không phải là điều kiện đủ để xảy ra tắc nghẽn. Điều kiện thứ tư là kết quả tất yếu từ ba điều kiện đầu.

#### **II.4.7. Ngăn chặn tắc nghẽn (Deadlock Prevention)**

Ngăn chặn tắc nghẽn là thiết kế một hệ thống sao cho hiện tượng tắc nghẽn bị loại trừ. Các phương thức ngăn chặn tắc nghẽn đều tập trung giải quyết bốn điều kiện gây ra tắc nghẽn, sao cho hệ thống không thể xảy ra đồng thời bốn điều kiện tắc nghẽn:

- Đối với điều kiện độc quyền: Điều kiện này gần như không tránh khỏi, vì sự độc quyền là cần thiết đối với tài nguyên thuộc loại phân chia được như các biến chung, các tập tin chia sẻ, hệ điều hành cần phải hỗ trợ sự độc quyền trên các tài nguyên này. Tuy nhiên, với những tài nguyên thuộc loại không phân chia được hệ điều hành có thể sử dụng kỹ thuật SPOOL (Simultaneous Peripheral Operation Online) để tạo ra nhiều tài nguyên ảo cung cấp cho các tiến trình đồng thời.

- Đối với điều kiện giữ và đợi: Điều kiện này có thể ngăn chặn bằng cách yêu cầu tiến trình yêu cầu tất cả tài nguyên mà nó cần tại một thời điểm và tiến trình sẽ bị khoá (blocked) cho đến khi yêu cầu tài nguyên của nó được hệ điều hành đáp ứng. Phương pháp này không hiệu quả. Thứ nhất, tiến trình phải đợi trong một khoảng thời gian dài để có đủ tài nguyên mới có thể chuyển sang hoạt động được, trong khi tiến trình chỉ cần một số ít tài nguyên trong số đó là có thể hoạt động được, sau đó yêu cầu tiếp. Thứ hai, lãng phí tài nguyên, vì có thể tiến trình giữ nhiều tài nguyên mà chỉ đến khi sắp kết thúc tiến trình mới sử dụng, và có thể đây là những tài nguyên mà các tiến trình khác đang rất cần. Ở đây hệ điều hành có thể tổ chức phân lớp tài nguyên hệ thống. Theo đó tiến trình phải trả tài nguyên ở mức thấp mới được cấp phát tài nguyên ở cấp cao hơn.

- Đối với điều kiện No preemption: Điều kiện này có thể ngăn chặn bằng cách, khi tiến trình bị rơi vào trạng thái khoá, hệ điều hành có thể thu hồi tài nguyên của tiến trình bị khoá để cấp phát cho tiến trình khác và cấp lại đầy đủ tài nguyên cho tiến trình khi tiến trình được đưa ra khỏi trạng thái khoá.

- Đối với điều kiện chờ đợi vòng tròn: Điều kiện này có thể ngăn chặn bằng cách phân lớp tài nguyên của hệ thống. Theo đó, nếu một tiến trình được cấp phát tài nguyên ở lớp L, thì sau đó nó chỉ có thể yêu cầu các tài nguyên ở lớp thấp hơn lớp L.

#### **II.4.8. Nhận biết tắc nghẽn (Deadlock Detection)**

Các phương thức ngăn chặn tắc nghẽn ở trên đều tập trung vào việc hạn chế quyền truy xuất đến tài nguyên và áp đặt các ràng buộc lên các tiến trình. Điều này có thể ảnh hưởng đến mục tiêu khai thác hiệu quả tài nguyên của hệ điều hành, ngăn chặn

độc quyền trên tài nguyên là một ví dụ, hệ điều hành phải cài đặt các cơ chế độc quyền để bảo vệ các tài nguyên chia sẻ. Và như đã phân tích ở trên việc cấp phát tài nguyên một lần cho các tiến trình để ngăn chặn hiện tượng hold and wait cũng tồn tại một vài hạn chế.

Các hệ điều hành có thể giải quyết vấn đề tắc nghẽn theo hướng phát hiện tắc nghẽn để tìm cách thoát khỏi tắc nghẽn. Phát hiện tắc nghẽn không giới hạn truy xuất tài nguyên và không áp đặt các ràng buộc lên tiến trình. Với phương thức phát hiện tắc nghẽn, các yêu cầu cấp phát tài nguyên được đáp ứng ngay nếu có thể. Để phát hiện tắc nghẽn hệ điều hành thường cài đặt một thuật toán để phát hiện hệ thống có tồn tại hiện tượng chờ đợi vòng tròn hay không.

Việc kiểm tra, để xem thử hệ thống có khả năng xảy ra tắc nghẽn hay không có thể được thực hiện liên tục mỗi khi có một yêu cầu tài nguyên, hoặc chỉ thực hiện thỉnh thoảng theo chu kỳ, phụ thuộc vào sự tắc nghẽn xảy ra như thế nào. Việc kiểm tra tắc nghẽn mỗi khi có yêu cầu tài nguyên sẽ nhận biết được khả năng xảy ra tắc nghẽn nhanh hơn, thuật toán được áp dụng đơn giản hơn vì chỉ dựa vào sự thay đổi trạng thái của hệ thống. Tuy nhiên, hệ thống phải tốn nhiều thời gian cho mỗi lần kiểm tra tắc nghẽn.

Mỗi khi tắc nghẽn được phát hiện, hệ điều hành thực hiện một vài giải pháp để thoát khỏi tắc nghẽn. Sau đây là một vài giải pháp có thể:

1. Thoát tất cả các tiến trình bị tắc nghẽn. Đây là một giải pháp đơn giản nhất, thường được các hệ điều hành sử dụng nhất.

2. Sao lưu lại mỗi tiến trình bị tắc nghẽn tại một vài điểm kiểm tra được định nghĩa trước, sau đó khởi động lại tất cả các tiến trình. Giải pháp này yêu cầu hệ điều hành phải lưu lại các thông tin cần thiết tại điểm dừng của tiến trình, đặc biệt là con trỏ lệnh và các tài nguyên tiến trình đang sử dụng, để có thể khởi động lại tiến trình được. Giải pháp này có nguy cơ xuất hiện tắc nghẽn trở lại là rất cao, vì khi tất cả các tiến trình đều được reset trở lại thì việc tranh chấp tài nguyên là khó tránh khỏi. Ngoài ra hệ điều hành thường phải chi phí rất cao cho việc tạm dừng và tái kích hoạt tiến trình.

3. Chỉ kết thúc một tiến trình trong tập tiến trình bị tắc nghẽn, thu hồi tài nguyên của tiến trình này, để cấp phát cho một tiến trình nào đó trong tập tiến trình tắc nghẽn để giúp tiến trình này ra khỏi tắc nghẽn, rồi gọi lại thuật toán kiểm tra tắc nghẽn để xem hệ thống đã ra khỏi tắc nghẽn hay chưa, nếu rồi thì dừng, nếu chưa thì tiếp tục giải phóng thêm tiến trình khác. Và lần lượt như thế cho đến khi tất cả các tiến trình trong tập tiến trình tắc nghẽn đều ra khỏi tình trạng tắc nghẽn. Trong giải pháp này vấn đề đặt ra đối với hệ điều hành là nên chọn tiến trình nào để giải phóng đầu tiên và dựa vào tiêu chuẩn nào để chọn lựa sao cho chi phí để giải phóng tắc nghẽn là thấp nhất.

4. Tập trung toàn bộ quyền ưu tiên sử dụng tài nguyên cho một tiến trình,

để tiến trình này ra khỏi tắc nghẽn, và rồi kiểm tra xem hệ thống đã ra khỏi tắc nghẽn hay chưa, nếu rồi thì dừng lại, nếu chưa thì tiếp tục. Lần lượt như thế cho đến khi hệ thống ra khỏi tắc nghẽn. Trong giải pháp này hệ điều hành phải tính đến chuyện tái kích hoạt lại tiến trình sau khi hệ thống ra khỏi tắc nghẽn.

Đối với các giải pháp 3 và 4, hệ điều hành dựa vào các tiêu chuẩn sau đây để chọn lựa tiến trình giải phóng hay ưu tiên tài nguyên: Thời gian xử lý ít nhất; Thời gian cần processor còn lại ít nhất; Tài nguyên cần cấp phát là ít nhất; Quyền ưu tiên là thấp nhất.

## **I.19. Điều phối tiến trình**

Trong môi trường hệ điều hành đa nhiệm, bộ phận điều phối tiến trình có nhiệm vụ xem xét và quyết định khi nào thì dừng tiến trình hiện tại để thu hồi processor và chuyển processor cho tiến trình khác, và khi đã có được processor thì chọn tiến trình nào trong số các tiến trình ở trạng thái ready để cấp processor cho nó. Ở đây chúng ta cần phân biệt sự khác nhau giữa điều độ tiến trình và điều phối tiến trình.

### **II.5.4. Mục tiêu điều phối tiến trình**

➤ **Các cơ chế điều phối tiến trình:** Trong công tác điều phối tiến trình bộ điều phối sử dụng hai cơ chế điều phối: Điều phối độc quyền và điều phối không độc quyền.

- Điều phối độc quyền: Khi có được processor tiến trình toàn quyền sử dụng processor cho đến khi tiến trình kết thúc xử lý hoặc tiến trình tự động trả lại processor cho hệ thống. Các quyết định điều phối xảy ra khi: Tiến trình chuyển trạng thái từ Running sang Blocked hoặc khi tiến trình kết thúc.

- Điều phối không độc quyền: Bộ phận điều phối tiến trình có thể tạm dừng tiến trình đang xử lý để thu hồi processor của nó, để cấp cho tiến trình khác, sao cho phù hợp với công tác điều phối hiện tại. Các quyết định điều phối xảy ra khi: Tiến trình chuyển trạng thái hoặc khi tiến trình kết thúc.

➤ **Các đặc điểm của tiến trình:** Khi tổ chức điều phối tiến trình, bộ phận điều phối tiến trình của hệ điều hành thường dựa vào các đặc điểm của tiến trình. Sau đây là một số đặc điểm của tiến trình:

- Tiến trình thiên hướng Vào/Ra: Là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các thao tác xuất/nhập dữ liệu, so với thời gian mà tiến trình cần để thực hiện các chỉ thị trong nó, được gọi là các tiến trình thiên hướng Vào/Ra.

- Tiến trình thiên hướng xử lý: Ngược lại với trên, đây là các tiến trình cần nhiều thời gian hơn cho việc thực hiện các chỉ thị trong nó, so với thời gian mà tiến trình để thực hiện các thao tác Vào/Ra.

- Tiến trình tương tác hay xử lý theo lô: Tiến trình cần phải trả lại kết quả

tức thời (như trong hệ điều hành tương tác) hay kết thúc xử lý mới trả về kết quả (như trong hệ điều hành xử lý theo lô).

- **Độ ưu tiên của tiến trình:** Mỗi tiến trình được gán một độ ưu tiên nhất định, độ ưu tiên của tiến trình có thể được phát sinh tự động bởi hệ thống hoặc được gán tường minh trong chương trình của người sử dụng. Độ ưu tiên của tiến trình có hai loại: Thứ nhất, độ ưu tiên tĩnh: là độ ưu tiên gán trước cho tiến trình và không thay đổi trong suốt thời gian sống của tiến trình. Thứ hai, độ ưu tiên động: là độ ưu tiên được gán cho tiến trình trong quá trình hoạt động của nó, hệ điều hành sẽ gán lại độ ưu tiên cho tiến trình khi môi trường xử lý của tiến trình bị thay đổi. Khi môi trường xử lý của tiến trình bị thay đổi hệ điều hành phải thay đổi độ ưu tiên của tiến trình cho phù hợp với tình trạng hiện tại của hệ thống và công tác điều phối tiến trình của hệ điều hành.

- **Thời gian sử dụng processor của tiến trình:** Tiến trình cần bao nhiêu khoảng thời gian của processor để hoàn thành xử lý.

- **Thời gian còn lại tiến trình cần processor:** Tiến trình còn cần bao nhiêu khoảng thời gian của processor nữa để hoàn thành xử lý.

Bộ phận điều phối tiến trình thường dựa vào đặc điểm của tiến trình để thực hiện điều phối ở mức tác vụ, hay điều phối tác vụ. Điều phối tác vụ được phải thực hiện trước điều phối tiến trình. Ở mức này hệ điều hành thực hiện việc chọn tác vụ để đưa vào hệ thống. Khi có một tiến trình được tạo lập hoặc khi có một tiến trình kết thúc xử lý thì bộ phận điều phối tác vụ được kích hoạt. Điều phối tác vụ quyết định sự đa chương của hệ thống và hiệu quả cũng như mục tiêu của điều phối của bộ phận điều phối tiến trình. Ví dụ, để khi thác tối đa thời gian xử lý của processor thì bộ phận điều phối tác vụ phải đưa vào hệ thống số lượng các tiến trình tính hướng Vào/Ra cân đối với số lượng các tiến trình tính hướng xử lý, các tiến trình này thuộc những tác vụ nào. Nếu trong hệ thống có quá nhiều tiến trình tính hướng Vào/Ra thì sẽ lãng phí thời gian xử lý của processor. Nếu trong hệ thống có quá nhiều tiến trình tính hướng xử lý thì processor không thể đáp ứng và có thể các tiến trình phải đợi lâu trong hệ thống, dẫn đến hiệu quả tương tác sẽ thấp.

➤ **Mục tiêu điều phối:** bộ phận điều phối tiến trình của hệ điều hành phải đạt được các mục tiêu sau đây trong công tác điều phối của nó.

- **Sự công bằng (Fairness):** Các tiến trình đều công bằng với nhau trong việc chia sẻ thời gian xử lý của processor, không có tiến trình nào phải chờ đợi vô hạn để được cấp processor.

- **Tính hiệu quả (Efficiency):** Tận dụng được 100% thời gian xử lý của processor. Trong công tác điều phối, khi processor rỗi bộ phận điều phối sẽ chuyển ngay nó cho tiến trình khác, nếu trong hệ thống có tiến trình đang ở trạng thái chờ processor, nên mục tiêu này dễ đạt được. Tuy nhiên, nếu hệ điều hành đưa vào hệ

thông quá nhiều tiến trình thiên hướng vào/ra, thì nguy cơ processor bị rối là có thể. Do đó, để đạt được mục tiêu này hệ điều hành phải tính toán và quyết định nên đưa vào hệ thống bao nhiêu tiến trình thiên hướng vào/ra, bao nhiêu tiến trình thiên hướng xử lý, là thích hợp.

- Thời gian đáp ứng hợp lý (Response time): Đối với các tiến trình tương tác, đây là khoảng thời gian từ khi tiến trình đưa ra yêu cầu cho đến khi nhận được sự hồi đáp. Một tiến trình đáp ứng yêu cầu của người sử dụng, phải nhận được thông tin hồi đáp từ yêu cầu của nó thì nó mới có thể trả lời người sử dụng. Do đó, theo người sử dụng thì bộ phận điều phối phải cực tiểu hoá thời gian hồi đáp của các tiến trình, có như vậy thì tính tương tác của tiến trình mới tăng lên.

- Thời gian lưu lại trong hệ thống (Turnaround time): Đây là khoảng thời gian từ khi tiến trình được đưa ra đến khi được hoàn thành. Bao gồm thời gian thực hiện thực tế cộng với thời gian đợi tài nguyên (bao gồm cả đợi processor). Đại lượng này dùng trong các hệ điều hành xử lý theo lô. Do đó, bộ phận điều phối phải cực tiểu thời gian hoàn thành (lưu lại trong hệ thống) của các tác vụ xử lý theo lô.

- Thông lượng tối đa (Throughtput): Chính sách điều phối phải cố gắng để cực đại được số lượng tiến trình hoàn thành trên một đơn vị thời gian. Mục tiêu này ít phụ thuộc vào chính sách điều phối mà phụ thuộc nhiều vào thời gian thực hiện trung bình của các tiến trình.

Công tác điều phối của hệ điều hành khó có thể thỏa mãn đồng thời tất cả các mục tiêu trên vì bản thân các mục tiêu này đã có sự mâu thuẫn với nhau. Các hệ điều hành chỉ có thể dung hòa các mục tiêu này ở một mức độ nào đó.

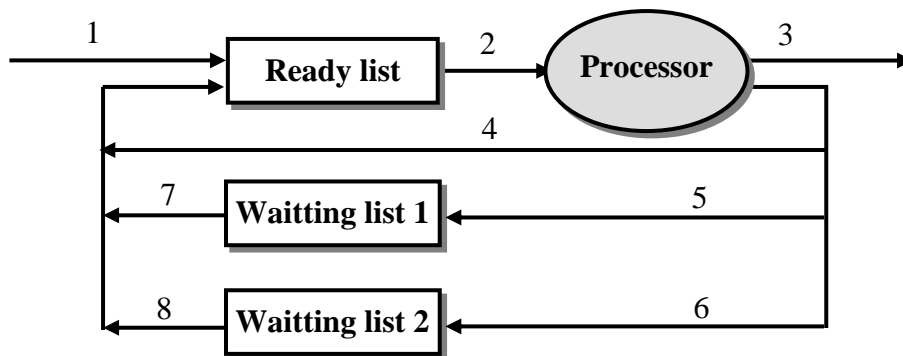
Ví dụ: Giả sử trong hệ thống có bốn tiến trình  $P_1, P_2, P_3, P_4$ , thời gian (t) mà các tiến trình này cần processor để xử lý lần lượt là 1, 12, 2, 1. Nếu ban đầu có 2 tiến trình  $P_1$  và  $P_2$  ở trạng thái ready thì chắc chắn bộ phận điều phối sẽ cấp processor cho  $P_1$ . Sau khi  $P_1$  kết thúc thì processor sẽ được cấp cho  $P_2$  để  $P_2$  hoạt động (running), khi  $P_2$  thực hiện được  $2t$  thì  $P_3$  được đưa vào trạng thái ready. Nếu để  $P_2$  tiếp tục thì  $P_3$  phải chờ lâu (chờ  $8t$ ), như vậy sẽ vi phạm mục tiêu thời gian hồi đáp và thông lượng tối đa (đối với  $P_3$ ). Nếu cho  $P_2$  dừng để cấp processor cho  $P_3$  hoạt động đến khi kết thúc, khi đó thì  $P_4$  vào trạng thái ready, bộ điều phối sẽ cấp processor cho  $P_4$ , và cứ như thế, thì  $P_2$  phải chờ lâu, như vậy sẽ đạt được mục tiêu: thời gian hồi đáp và thông lượng tối đa nhưng vi phạm mục tiêu: công bằng và thời gian lưu lại trong hệ thống (đối với  $P_2$ ).

### **II.5.5. Tổ chức điều phối tiến trình**

Để tổ chức điều phối tiến trình hệ điều hành sử dụng hai danh sách: Danh sách sẵn sàng (Ready list) dùng để chứa các tiến trình ở trạng thái sẵn sàng. Danh sách đợi (Waiting list) dùng để chứa các tiến trình đang đợi để được bổ sung vào danh sách

sẵn sàng.

Chỉ có những tiến trình trong ready list mới được chọn để cấp processor. Các tiến trình bị chuyển về trạng thái blocked sẽ được bổ sung vào waiting list. Hệ thống chỉ có duy nhất một ready list, nhưng có thể tồn tại nhiều waiting list. Thông thường hệ điều hành thiết kế nhiều waiting list, mỗi waiting list dùng để chứa các tiến trình đang đợi được cấp phát một tài nguyên hay một sự kiện riêng biệt nào đó. Hình sau đây minh họa cho việc chuyển tiến trình giữa các danh sách:



**Hình 2.7:** Sơ đồ chuyển tiến trình vào các danh sách

Trong đó:

1. Tiến trình trong hệ thống được cấp đầy đủ tài nguyên chỉ thiếu processor.
2. Tiến trình được bộ điều phối chọn ra để cấp processor để bắt đầu xử lý.
3. Tiến trình kết thúc xử lý và trả lại processor cho hệ điều hành.
4. Tiến trình hết thời gian được quyền sử dụng processor (time-out), bị bộ điều phối tiến trình thu hồi lại processor.
5. Tiến trình bị khóa (blocked) do yêu cầu tài nguyên nhưng chưa được hệ điều hành cấp phát. Khi đó tiến trình được đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 1).
6. Tiến trình bị khóa (blocked) do đang đợi một sự kiện nào đó xảy ra. Khi đó tiến trình được bộ điều phối đưa vào danh sách các tiến trình đợi tài nguyên (waiting list 2).
7. Tài nguyên mà tiến trình yêu cầu đã được hệ điều hành cấp phát. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sàng (ready list) để chờ được cấp processor để được hoạt động.
8. Sự kiện mà tiến trình chờ đã xảy ra. Khi đó tiến trình được bộ điều phối chuyển sang danh sách các tiến trình ở trạng thái sẵn sàng (ready list) để chờ được cấp processor.



### II.5.6. Các chiến lược điều phối tiến trình

➤ **Chiến lược FIFO (First In First Out):** trong chiến lược này, khi processor rỗi thì hệ điều hành sẽ cấp nó cho tiến trình đầu tiên trong ready list, đây là tiến trình được chuyển sang trạng thái ready sớm nhất, có thể là tiến trình được đưa vào hệ thống sớm nhất. FIFO được sử dụng trong điều phối độc quyền nên khi tiến trình được cấp processor nó sẽ sở hữu processor cho đến khi kết thúc xử lý hay phải đợi một thao tác vào/ra hoàn thành, khi đó tiến trình chủ động trả lại processor cho hệ thống.

Ví dụ: Nếu hệ điều hành cần cấp processor cho 3 tiến trình P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, với thời điểm vào ready list và khoảng thời gian mỗi tiến trình cần processor được mô tả trong bảng sau:

Tiến trình	thời điểm vào	t/g xử lý
P <sub>1</sub>	0	24
P <sub>2</sub>	1	3
P <sub>3</sub>	2	3

Thì thứ tự cấp processor cho các tiến trình diễn ra như sau:

Tiến trình:	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>
Thời điểm:	0	24	27

Vậy thời gian chờ của tiến trình P<sub>1</sub> là 0, của P<sub>2</sub> là 23 (24 - 0), của P<sub>3</sub> là 25 (24 + 3 - 2). Và thời gian chờ đợi trung bình của các tiến trình là:

$$(0 + 23 + 25)/3 = 16.$$

Như vậy FIFO tồn tại một số hạn chế: Thứ nhất, có thời gian chờ đợi trung bình lớn nên không phù hợp với các hệ thống chia sẻ thời gian. Thứ hai, khả năng tương tác kém khi nó được áp dụng trên các hệ thống uniprocessor. Thứ ba, nếu các tiến trình ở đầu ready list cần nhiều thời gian của processor thì các tiến trình ở cuối ready list sẽ phải chờ lâu mới được cấp processor.

➤ **Chiến lược phân phối xoay vòng (RR: Round Robin):** trong chiến lược này, ready list được thiết kế theo dạng danh sách nối vòng. Tiến trình được bộ điều phối chọn để cấp processor cũng là tiến trình ở đầu ready list, nhưng sau một khoảng thời gian nhất định nào đó thì bộ điều phối lại thu hồi lại processor của tiến trình vừa được cấp processor và chuyển processor cho tiến trình kế tiếp (bây giờ đã trở thành tiến trình đầu tiên) trong ready list, tiến trình vừa bị thu hồi processor được đưa vào lại cuối ready list. Rõ ràng đây là chiến lược điều phối không độc quyền.

Khoảng khoản thời gian mà mỗi tiến trình được sở hữu processor để hoạt động là bằng nhau, và thường được gọi là Quantum.

Ví dụ: Nếu hệ điều hành cần cấp processor cho 3 tiến trình P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub> với thời

điểm vào ready list và khoảng thời gian mỗi tiến trình cần processor được mô tả trong bảng sau:

Tiến trình	thời điểm vào	t/g xử lý	
P <sub>1</sub>	0	24	
P <sub>2</sub>	1	3	
P <sub>3</sub>	2	3	Quantum = 4

Thì thứ tự cấp processor cho các tiến trình lần lượt là:

Tiến trình	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>	P <sub>1</sub>
Thời điểm	0	4	7	10	14	18	22	26

Vậy thời gian chờ đợi trung bình sẽ là:  $(0 + 6 + 3 + 5)/3 = 4.46$

Như vậy RR có thời gian chờ đợi trung bình nhỏ hơn so với FIFO

**Trong chiến lược này, vấn đề đặt ra đối với công tác thiết kế là: nên chọn quantum bằng bao nhiêu là thích hợp, nếu quantum nhỏ thì hệ thống phải tốn nhiều thời gian cho việc cập nhật ready list và chuyển trạng thái tiến trình, dẫn đến vi phạm mục tiêu: khai thác tối đa thời gian xử lý của processor. Nếu quantum lớn thì thời gian chờ đợi trung bình và thời gian hồi đáp sẽ tăng lên, dẫn đến tính tương tác của hệ thống bị giảm xuống.**

➤ **Chiến lược theo độ ưu tiên:** trong chiến lược này, bộ phận điều phối tiến trình dựa vào độ ưu tiên của các tiến trình để tổ chức cấp processor cho tiến trình. Tiến trình được chọn để cấp processor là tiến trình có độ ưu tiên cao nhất, tại thời điểm hiện tại.

Ở đây hệ điều hành thường tổ chức gán độ ưu tiên cho tiến trình theo nguyên tắc kết hợp giữ gán tĩnh và gán động. Khi khởi tạo tiến trình được gán độ ưu tiên tĩnh, sau đó phụ thuộc vào môi trường hoạt động của tiến trình và công tác điều phối tiến trình của bộ phận điều phối mà hệ điều hành có thể thay đổi độ ưu tiên của tiến trình.

Khi hệ thống phát sinh một tiến trình ready mới, thì bộ phận điều phối sẽ so sánh độ ưu tiên của tiến trình mới phát sinh với độ ưu tiên của tiến trình đang sở hữu processor (tạm gọi là tiến trình hiện tại). Nếu tiến trình mới có độ ưu tiên thấp hơn tiến trình hiện tại thì bộ phận điều phối sẽ chen nó vào ready list tại vị trí thích hợp. Nếu tiến trình mới có độ ưu tiên cao hơn tiến trình hiện tại thì bộ điều phối sẽ thu hồi processor từ tiến trình hiện tại để cấp cho tiến trình mới yêu cầu, nếu là điều phối không độc quyền, hoặc chen tiến trình mới vào ready list tại vị trí thích hợp, nếu là điều phối độc quyền.

Chiến lược này cũng phải sử dụng ready list, và ready list luôn được xếp theo

thứ tự giảm dần của độ ưu tiên kể từ đầu danh sách. Điều này có nghĩa là tiến trình được chọn để cấp processor là tiến trình ở đầu ready list.

Ví dụ: Nếu hệ điều hành cần cấp processor cho 3 tiến trình  $P_1$ ,  $P_2$ ,  $P_3$  với độ ưu tiên và khoảng thời gian mỗi tiến trình cần processor được mô tả trong bảng sau:

Tiến trình	độ ưu tiên	thời gian xử lý
$P_1$	3	24
$P_2$	1	3
$P_3$	2	3

Thì thứ tự cấp processor (theo nguyên tắc độc quyền) cho các tiến trình lần lượt là:

Tiến trình	$P_2$	$P_3$	$P_1$
Thời điểm	0	4	7

Chiến lược này có thể dẫn đến hậu quả: các tiến trình có độ ưu tiên thấp sẽ rơi vào tình trạng chờ đợi vô hạn. Để khắc phục điều này hệ điều hành thường hạ độ ưu tiên của các tiến trình có độ ưu tiên cao sau mỗi lần nó được cấp processor.

➤ **Chiến lược SJF** (Shortest Job First: công việc ngắn nhất): Đây là trường hợp đặc biệt của chiến lược theo độ ưu tiên. Trong chiến lược này độ ưu tiên  $P$  của mỗi tiến trình là  $1/t$ , với  $t$  là khoảng thời gian mà tiến trình cần processor. Bộ điều phối sẽ chọn tiến trình có  $P$  lớn để cấp processor, tức là ưu tiên cho những tiến trình có thời gian xử lý (thời gian cần processor) nhỏ.

Chiến lược này có thể có thời gian chờ đợi trung bình đạt cực tiểu. Nhưng hệ điều hành khó có thể đoán được thời gian xử lý mà tiến trình yêu cầu.

➤ **Chiến lược nhiều cấp độ ưu tiên:** Hệ điều hành phân lớp các tiến trình theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng lớp tiến trình. Mỗi cấp độ ưu tiên có một ready list riêng. Bộ điều phối dùng chiến lược điều phối thích hợp cho từng ready list. Hệ điều hành cũng phải thiết kế một cơ chế thích hợp để điều phối tiến trình giữa các lớp.

Trong chiến lược này hệ điều hành sử dụng độ ưu tiên tĩnh, và điều phối không độc quyền, do đó một tiến trình thuộc ready list ở cấp ưu tiên  $i$  sẽ chỉ được cấp phát processor khi trong ready list ở cấp ưu tiên  $j$  ( $j > i$ ) không còn một tiến trình nào.

Các tiến trình ở ready list có độ ưu tiên thấp sẽ phải chờ đợi processor trong một khoảng thời gian dài, có thể là vô hạn. Để khắc phục điều này hệ điều hành xây dựng chiến lược điều phối: Nhiều mức độ ưu tiên xoay vòng. Trong chiến lược này hệ điều hành chuyển dần một tiến trình ở ready list có độ ưu tiên cao xuống ready list có độ ưu tiên thấp hơn sau mỗi lần sử dụng processor, và ngược lại một tiến trình ở lâu trong ready list có độ ưu tiên thấp thì sẽ được chuyển dần lên ready list

có độ ưu tiên cao hơn.

Khi xây dựng chiến lược nhiều mức độ ưu tiên xoay vòng hệ điều hành cần xác định các thông tin sau: Số lượng các lớp ưu tiên. Chiến lược điều phối riêng cho từng read list trong mỗi lớp ưu tiên. Một tiến trình ready mới sẽ được đưa vào ready list nào. Khi nào thì thực hiện việc di chuyển một tiến trình từ ready list này sang ready list khác.

## I.20. Tiến trình trong Windows NT

### ➤ Giới thiệu

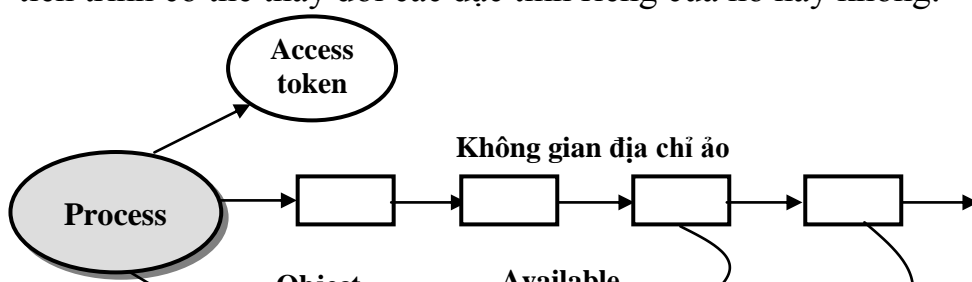
Việc thiết kế tiến trình trong Windows NT được thực hiện từ nhu cầu cung cấp sự hỗ trợ cho nhiều môi trường hệ điều hành khác nhau. Các tiến trình được hỗ trợ bởi các hệ điều hành khác nhau có một số điểm khác nhau, bao gồm:

- Cách đặt tên cho các tiến trình.
- Có hay không các tiểu trình bên trong tiến trình.
- Các tiến trình được thể hiện như thế nào.
- Các tài nguyên của tiến trình được bảo vệ như thế nào.
- Các chiến lược được sử dụng trong giao tiếp và đồng bộ tiến trình
- Cách mà các tiến trình liên hệ với mỗi tiến trình khác

Do đó, các cấu trúc tiến trình và các dịch vụ ban đầu được cung cấp bởi kernel của windows NT là phải đạt mục tiêu đơn giản và tổng quát, cho phép mỗi hệ thống con hệ điều hành mô phỏng một cấu trúc tiến trình riêng và thiết thực. Các đặc tính quan trọng của các tiến trình windows NT là như sau:

- Các tiến trình NT được cài đặt như là các đối tượng.
- Một tiến trình có khả năng thi hành có thể chứa một hay nhiều tiểu trình.
- Cả các đối tượng tiến trình và tiểu trình có sẵn khả năng đồng bộ.
- Kernel của Windows NT không duy trì mối quan hệ nào giữa các tiến trình mà nó đã tạo ra, kể cả các mối quan hệ cha con.

Hình sau đây minh họa cách một tiến trình liên hệ với các tài nguyên nó điều khiển hoặc sử dụng.



- Liên quan đến tiến trình là một dãy các block định nghĩa không gian địa chỉ ảo đang được kết gán cho tiến trình đó. Tiến trình không thể trực tiếp sửa đổi các cấu trúc này mà phải dựa vào thành phần quản lý bộ nhớ ảo, trình quản lý bộ nhớ cung cấp dịch vụ định vị bộ nhớ cho tiến trình.
- Cuối cùng, tiến trình bao gồm một bảng đối tượng, với các handle đến các đối tượng khác liên quan đến tiến trình này. Một handle tồn tại cho mỗi tiểu trình chứa trong đối tượng này. Trong hình này, tiến trình có một tiểu trình đơn. Ngoài ra, tiến trình truy xuất đến một đối tượng file và đến một đối tượng section mà nó xác định một section bộ nhớ chia sẻ.

### ➤ **Các đối tượng tiến trình và tiểu trình**

Cấu trúc hướng đối tượng của windows NT làm cho việc phát triển tiến trình của nó được dễ dàng hơn. Windows NT có hai kiểu đối tượng liên quan đến tiến trình: *các tiến trình và tiểu trình*. Một tiến trình là một thực thể tương ứng với một công việc của người sử dụng hay ứng dụng mà nó sở hữu các tài nguyên, như bộ nhớ, và các tập tin được mở. Một tiểu trình là một đơn vị có thể điều phối, sự thực thi của nó có thể được thực hiện tuần tự hay bị ngắt, do đó processor có thể chuyển từ tiểu trình này sang tiểu trình khác.

Mỗi tiến trình windows NT được thể hiện bởi một đối tượng, mà cấu trúc chung của nó bao gồm: loại đối tượng (Object type), các thuộc tính đối tượng (Object Attributes) và các dịch vụ (Service).

Mỗi tiến trình được định nghĩa bởi một tập các thuộc tính và các dịch vụ mà nó có thể thực hiện. Một tiến trình sẽ thực hiện một dịch vụ để nhận thông điệp thích hợp.

Khi windows NT tạo ra một tiến trình mới, nó sử dụng lớp đối tượng, hay

kiểu, định nghĩa cho tiến trình windows NT như một template để tạo ra một thể hiện mới của đối tượng. Tại thời điểm tạo đó, các giá trị thuộc tính sẽ được gán.

Một tiến trình windows NT phải chứa ít nhất một tiểu trình để thực thi. Tiểu trình đó sau đó có thể tạo ra các tiểu trình khác. Trong hệ thống đa xử lý, nhiều tiểu trình của cùng tiến trình có thể thực thi song song.

Một số thuộc tính của một tiểu trình tương tự với các thuộc tính của một tiến trình. Trong những trường hợp đó, giá trị thuộc tính của tiểu trình được thừa kế từ giá trị thuộc tính của tiến trình. Ví dụ, các processor liên quan đến các tiểu trình là tập các processor trong một hệ thống multiprocessor (đa vi xử lý) mà nó có thể thực thi tiểu trình này; tập đó tương đương hay một tập con của các processor liên quan đến tiến trình.

Một trong những thuộc tính của đối tượng tiểu trình là ngữ cảnh (context) của tiểu trình. Thông tin này cho phép các tiểu trình có thể tạm dừng và tái kích hoạt trở lại được. Hơn thế nữa, nó có thể thay đổi hành vi của một tiểu trình bằng cách thay đổi ngữ cảnh của nó khi nó bị tạm dừng.

### ➤ **Multithreading (Đa tiểu trình)**

Windows NT hỗ trợ đồng thời nhiều tiến trình bởi vì các tiểu trình trong các tiến trình khác nhau có thể thực thi đồng thời. Hơn nữa, nhiều tiểu trình bên trong cùng một tiến trình có thể định vị tách biệt các processor và thực thi đồng thời. Một tiến trình đa tiểu trình đạt được sự đồng thời mà không cần sử dụng nhiều tiến trình. Các tiểu trình bên trong cùng tiến trình có thể trao đổi thông tin thông qua bộ nhớ chia sẻ và truy xuất các tài nguyên chia sẻ của tiến trình.

Một tiến trình đa tiểu trình hướng đối tượng là một công cụ hiệu quả cho việc cung cấp các ứng dụng server. Một tiến trình server đơn lẻ có thể phục vụ một số client. Mỗi client yêu cầu khởi phát việc tạo một tiểu trình mới bên trong server.

### ➤ **Hỗ trợ các hệ thống con hệ điều hành**

Mục tiêu chung nhất của tiến trình và tiểu trình là phải hỗ trợ các cấu trúc tiến trình và tiểu trình của các client trên các hệ điều hành khác nhau. Đó là trách nhiệm của mỗi hệ thống con hệ điều hành, để khai thác các đặc tính của tiến trình và tiểu trình windows NT, để mô phỏng dễ dàng tiến trình và tiểu trình của hệ điều hành tương ứng của nó.

Sau đây chúng ta hãy quan sát quá trình tạo tiến trình trong windows NT để thấy được sự hỗ trợ tiến trình và tiểu trình cho các hệ thống con hệ điều hành. Việc tạo tiến trình bắt đầu với một yêu cầu một tiến trình mới từ một ứng dụng hệ điều hành. Yêu cầu tạo tiến trình được phát ra từ một ứng dụng đến hệ thống con được bảo vệ tương ứng. Đến lượt nó, hệ thống con phát ra một yêu cầu tiến trình cho thành phần Executive của windows NT. Windows NT tạo một đối tượng tiến trình và trả ra một handle của đối tượng đó đến cho hệ thống con. Khi windows NT tạo

một tiến trình, nó không tự động tạo một tiểu trình. Do đó, đối với các hệ điều hành này, hệ thống con gọi trình quản lý tiến trình windows NT một lần nữa để tạo ra một tiểu trình cho tiến trình mới, mà nó nhận được một handle của tiểu trình từ windows NT. Thông tin tiểu trình và tiến trình thích hợp sau đó được trả lại cho ứng dụng. Trong Windows 16-bit và POSIX, các tiểu trình không được hỗ trợ. Do đó, đối với các hệ điều hành như thế này, hệ thống con có một tiểu trình cho tiến trình mới từ windows NT, cho nên tiến trình có thể được kích hoạt, nhưng chỉ trả lại thông tin tiến trình cho ứng dụng. Trong thực tế, tiến trình ứng dụng được cài đặt bằng cách sử dụng một tiểu trình không thấy cho ứng dụng.

Khi một tiến trình mới được tạo trong Win32 hay OS/2, tiến trình mới thừa kế nhiều đặc tính của nó từ tiến trình tạo. Tuy nhiên, trong môi trường hệ điều hành windows NT, việc tạo tiến trình này không được thực hiện trực tiếp. Một tiến trình client ứng dụng phát ra yêu cầu tạo tiến trình của nó đến cho hệ thống con hệ điều hành; sau đó một tiến trình trong hệ thống con đến lượt nó phát ra một yêu cầu tiến trình cho thành phần Executive của windows NT. Vì tác dụng mong đợi là tiến trình mới thừa kế các đặc tính của tiến trình client và không thừa kế từ tiến trình server, nên windows NT cho phép hệ thống con chỉ định cha của tiến trình mới. Sau đó tiến trình mới thừa kế token truy xuất, thời gian quota, độ ưu tiên cơ sở, và mối quan hệ processor mặc định của tiến trình cha.

Trong windows NT, không có quan hệ được định nghĩa trước giữa các tiến trình. Tuy nhiên, cả hệ điều hành POSIX và hệ điều hành OS/2 đều áp đặt một mối quan hệ phân cấp. Mỗi tiến trình mở rộng một tiến trình ban đầu được tạo bởi một tiến trình khác và được xem như cấp dưới tiến trình tạo ra nó. Bằng cách sử dụng các handle đối tượng, hệ thống con hệ điều hành có thể duy trì các mối quan hệ giữa các tiến trình này.

### Chương III

## QUẢN LÝ BỘ NHỚ

---

Quản lý bộ nhớ là một trong những nhiệm vụ quan trọng và phức tạp nhất của hệ điều hành. Bộ phận quản lý bộ nhớ xem bộ nhớ chính như là một tài nguyên của hệ thống dùng để cấp phát và chia sẻ cho nhiều tiến trình đang ở trong trạng thái active. Các hệ điều hành đều mong muốn có nhiều hơn các tiến trình trên bộ nhớ chính. Công cụ cơ bản

của quản lý bộ nhớ là sự phân trang (paging) và sự phân đoạn (segmentation). Với sự phân trang mỗi tiến trình được chia thành nhiều phần nhỏ có quan hệ với nhau, với kích thước của trang là cố định. Sự phân đoạn cung cấp cho chương trình người sử dụng các khối nhớ có kích thước khác nhau. Hệ điều hành cũng có thể kết hợp giữa phân trang và phân đoạn để có được một chiến lược quản lý bộ nhớ linh hoạt hơn.

### **III.9. Nhiệm vụ của quản lý bộ nhớ**

Trong các hệ thống đơn chương trình (uniprogramming), trên bộ nhớ chính ngoài hệ điều hành, chỉ có một chương trình đang thực hiện. Trong các hệ thống đa chương (multiprogramming) trên bộ nhớ chính ngoài hệ điều hành, có thể có nhiều tiến trình đang hoạt động. Do đó nhiệm vụ quản lý bộ nhớ của hệ điều hành trong hệ thống đa chương trình sẽ phức tạp hơn nhiều so với trong hệ thống đơn chương trình. Trong hệ thống đa chương bộ phận quản lý bộ nhớ phải có nhiệm vụ đưa bất kỳ một tiến trình nào đó vào bộ nhớ khi nó có yêu cầu, kể cả khi trên bộ nhớ không còn không gian trống, ngoài ra nó phải bảo vệ chính hệ điều hành và các tiến trình trên bộ nhớ tránh các trường hợp truy xuất bất hợp lệ xảy ra. Như vậy việc quản lý bộ nhớ trong các hệ thống đa chương là quan trọng và cần thiết. Bộ phận quản lý bộ nhớ phải thực hiện các nhiệm vụ sau đây:

➤ **Sự tái định vị (Relocation):** Trong các hệ thống đa chương, không gian bộ nhớ chính thường được chia sẻ cho nhiều tiến trình khác nhau và yêu cầu bộ nhớ của các tiến trình luôn lớn hơn không gian bộ nhớ vật lý mà hệ thống có được. Do đó, một chương trình đang hoạt động trên bộ nhớ cũng có thể bị đưa ra đĩa (swap-out) và nó sẽ được đưa vào lại (swap-in) bộ nhớ tại một thời điểm thích hợp nào đó sau này. Vấn đề đặt ra là khi đưa một chương trình vào lại bộ nhớ thì hệ điều hành phải định vị nó vào đúng vị trí mà nó đã được nạp trước đó. Để thực hiện được điều này hệ điều hành phải có các cơ chế để ghi lại tất cả các thông tin liên quan đến một chương trình bị swap-out, các thông tin này là cơ sở để hệ điều hành swap-in chương trình vào lại bộ nhớ chính và cho nó tiếp tục hoạt động. Hệ điều hành buộc phải swap-out một chương trình vì nó còn không gian bộ nhớ chính để nạp tiến trình khác, do đó sau khi swap-out một chương trình hệ điều hành phải tổ chức lại bộ nhớ để chuẩn bị nạp tiến trình vừa có yêu cầu. Các nhiệm vụ trên do bộ phận quản lý bộ nhớ của hệ điều hành thực hiện. Ngoài ra trong nhiệm vụ này hệ điều hành phải có khả năng chuyển đổi các địa chỉ bộ nhớ được ghi trong code của chương trình thành các địa chỉ vật lý thực tế trên bộ nhớ chính khi chương trình thực hiện các thao tác truy xuất trên bộ nhớ, bởi vì người lập trình không hề biết trước hiện trạng của bộ nhớ chính và vị trí mà chương trình được nạp khi chương trình của họ hoạt động. Trong một số trường hợp khác các chương trình bị swap-out có thể được swap-in vào lại bộ nhớ tại vị trí khác với vị trí mà nó được nạp



trước đó.

➤ **Bảo vệ bộ nhớ (Protection):** Mỗi tiến trình phải được bảo vệ để chống lại sự truy xuất bất hợp lệ vô tình hay có chủ ý của các tiến trình khác. Vì thế các tiến trình trong các chương trình khác không thể tham chiếu đến các vùng nhớ đã dành cho một tiến trình khác để thực hiện các thao tác đọc/ghi mà không được phép (permission), mà nó chỉ có thể truy xuất đến không gian địa chỉ bộ nhớ mà hệ điều hành đã cấp cho tiến trình đó. Để thực hiện điều này hệ thống quản lý bộ nhớ phải biết được không gian địa chỉ của các tiến trình khác trên bộ nhớ và phải kiểm tra tất cả các yêu cầu truy xuất bộ nhớ của mỗi tiến trình khi tiến trình đưa ra địa chỉ truy xuất. Điều này khó thực hiện vì không thể xác định địa chỉ của các chương trình trong bộ nhớ chính trong quá trình biên dịch mà phải thực hiện việc tính toán địa chỉ tại thời điểm chạy chương trình. Hệ điều hành có nhiều chiến lược khác nhau để thực hiện điều này.

Điều quan trọng nhất mà hệ thống quản lý bộ nhớ phải thực hiện là không cho phép các tiến trình của người sử dụng truy cập đến bất kỳ một vị trí nào của chính hệ điều hành, ngoại trừ vùng dữ liệu và các routine mà hệ điều hành cung cấp cho chương trình người sử dụng.

➤ **Chia sẻ bộ nhớ (Sharing):** Bất kỳ một chiến lược nào được cài đặt đều phải có tính mềm dẻo để cho phép nhiều tiến trình có thể truy cập đến cùng một địa chỉ trên bộ nhớ chính. Ví dụ, khi có nhiều tiến trình cùng thực hiện một chương trình thì việc cho phép mỗi tiến trình cùng truy cập đến một bản copy của chương trình sẽ thuận lợi hơn khi cho phép mỗi tiến trình truy cập đến một bản copy sở hữu riêng. Các tiến trình đồng thực hiện (co-operating) trên một vài tác vụ có thể cần để chia sẻ truy cập đến cùng một cấu trúc dữ liệu. Hệ thống quản lý bộ nhớ phải điều khiển việc truy cập đến không gian bộ nhớ được chia sẻ mà không vi phạm đến các yêu cầu bảo vệ bộ nhớ. Ngoài ra, trong môi trường hệ điều hành đa nhiệm hệ điều hành phải chia sẻ không gian nhớ cho các tiến trình để hệ điều hành có thể nạp được nhiều tiến trình vào bộ nhớ để các tiến trình này có thể hoạt động đồng thời với nhau.

➤ **Tổ chức bộ nhớ logic (Logical organization):** Bộ nhớ chính của hệ thống máy tính được tổ chức như là một dòng hoặc một mảng, không gian địa chỉ bao gồm một dãy có thứ tự các byte hoặc các word. Bộ nhớ phụ cũng được tổ chức tương tự. Mặc dù việc tổ chức này có sự kết hợp chặt chẽ với phần cứng thực tế của máy nhưng nó không phù hợp với các chương trình. Đa số các chương trình đều được chia thành các modun, một vài trong số đó là không thể thay đổi (read only, execute only) và một vài trong số đó chứa dữ liệu là có thể thay đổi. Nếu hệ điều hành và phần cứng máy tính có thể giao dịch một cách hiệu quả với các chương trình của người sử dụng và dữ liệu trong các modun thì một số thuận lợi có thể thấy rõ sau đây:

- Các modul có thể được viết và biên dịch độc lập, với tất cả các tham chiếu từ một modul đến modul khác được giải quyết bởi hệ thống tại thời điểm chạy.
- Các mức độ khác nhau của sự bảo vệ, read-only, execute-only, có thể cho ra các modul khác nhau.
- Nó có thể đưa ra các cơ chế để các modul có thể được chia sẻ giữa các tiến trình.

Công cụ đáp ứng cho yêu cầu này là sự phân đoạn (segmentation), đây là một trong những kỹ thuật quản lý bộ nhớ được trình bày trong chương này.

➤ **Tổ chức bộ nhớ vật lý (Physical organization):** Như chúng ta đã biết bộ nhớ máy tính được tổ chức theo 2 cấp: bộ nhớ chính và bộ nhớ phụ. Bộ nhớ chính cung cấp một tốc độ truy cập dữ liệu cao, nhưng dữ liệu trên nó phải được làm tươi thường xuyên và không thể tồn tại lâu dài trên nó. Bộ nhớ phụ có tốc độ truy xuất chậm và rẻ tiền hơn so với bộ nhớ chính nhưng nó không cần làm tươi thường xuyên. Vì thế bộ nhớ phụ có khả năng lưu trữ lớn và cho phép lưu trữ dữ liệu và chương trình trong một khoảng thời gian dài, trong khi đó bộ nhớ chính chỉ để giữ (hold) một khối lượng nhỏ các chương trình và dữ liệu đang được sử dụng tại thời điểm hiện tại.

Trong giản đồ 2 cấp này, việc tổ chức luồng thông tin giữa bộ nhớ chính và bộ nhớ phụ là một nhiệm vụ quan trọng của hệ thống. Sự chịu trách nhiệm cho luồng này có thể được gán cho từng người lập trình riêng, nhưng điều này là không hợp lý và có thể gây rắc rối, là do hai nguyên nhân:

- Không gian bộ nhớ chính dành cho các chương trình cùng với dữ liệu của nó thường là không đủ, trong trường hợp này, người lập trình phải tiến hành một thao tác được hiểu như là Overlaying, theo đó chương trình và dữ liệu được tổ chức thành các modul khác nhau có thể được gán trong cùng một vùng của bộ nhớ, trong đó có một chương trình chính chịu trách nhiệm chuyển các modul vào và ra khi cần.
- Trong môi trường đa chương trình, người lập trình không thể biết tại một thời điểm xác định có bao nhiêu không gian nhớ còn trống hoặc khi nào thì không gian nhớ sẽ trống.

Như vậy nhiệm vụ di chuyển thông tin giữa 2 cấp bộ nhớ phải do hệ thống thực hiện. Đây là nhiệm vụ cơ bản mà thành phần quản lý bộ nhớ phải thực hiện.

### **III.10. Kỹ thuật cấp phát bộ nhớ ( nạp chương trình vào bộ nhớ chính)**

#### **III.2.5. Kỹ thuật phân vùng cố định (Fixed Partitioning)**

Trong kỹ thuật này không gian địa chỉ của bộ nhớ chính được chia thành 2 phần cố định, phần nằm ở vùng địa chỉ thấp dùng để chứa chính hệ điều hành, phần còn lại,

tạm gọi là phần user program, là sẵn sàng cho việc sử dụng của các tiến trình khi các tiến trình được nạp vào bộ nhớ chính.

Trong các hệ thống đơn chương, phần user program được dùng để cấp cho chỉ một chương trình duy nhất, do đó nhiệm vụ quản lý bộ nhớ của hệ điều hành trong trường hợp này sẽ đơn giản hơn, hệ điều hành chỉ kiểm soát sự truy xuất bộ nhớ của chương trình người sử dụng, không cho nó truy xuất lên vùng nhớ của hệ điều hành. Để thực hiện việc này hệ điều hành sử dụng một thanh ghi giới hạn để ghi địa chỉ ranh giới giữa hệ điều hành và chương trình của người sử dụng, theo đó khi chương trình người sử dụng cần truy xuất một địa chỉ nào đó thì hệ điều hành sẽ so sánh địa chỉ này với giá trị địa chỉ được ghi trong thanh ghi giới hạn, nếu nhỏ hơn thì từ chối không cho truy xuất, ngược lại thì cho phép truy xuất. Việc so sánh địa chỉ này cần phải có sự hỗ trợ của phần cứng và có thể làm giảm tốc độ truy xuất bộ nhớ của hệ thống nhưng bảo vệ được hệ điều hành tránh việc chương trình của người sử dụng làm hỏng hệ điều hành dẫn đến làm hỏng hệ thống.

Trong các hệ thống đa chương, phần user program lại được phân ra thành nhiều phân vùng (partition) với các biên vùng cố định có kích thước bằng nhau hay không bằng nhau. Trong trường hợp này một tiến trình có thể được nạp vào bất kỳ partition nào nếu kích thước của nó nhỏ hơn hoặc bằng kích thước của partition và partition này còn trống. Khi có một tiến trình cần được nạp vào bộ nhớ nhưng tất cả các partition đều đã chứa các tiến trình khác thì hệ điều hành có thể chuyển một tiến trình nào đó, mà hệ điều hành cho là hợp lệ (kích thước vừa đủ, không đang ở trạng thái ready hoặc running, không có quan hệ với các tiến trình running khác, ...), ra ngoài (swap out), để lấy partition trống đó nạp tiến trình vừa có yêu cầu. Đây là nhiệm vụ phức tạp của hệ điều hành, hệ điều hành phải chi phí cao cho công việc này.

Có hai trở ngại trong việc sử dụng các phân vùng cố định với kích thước bằng nhau:

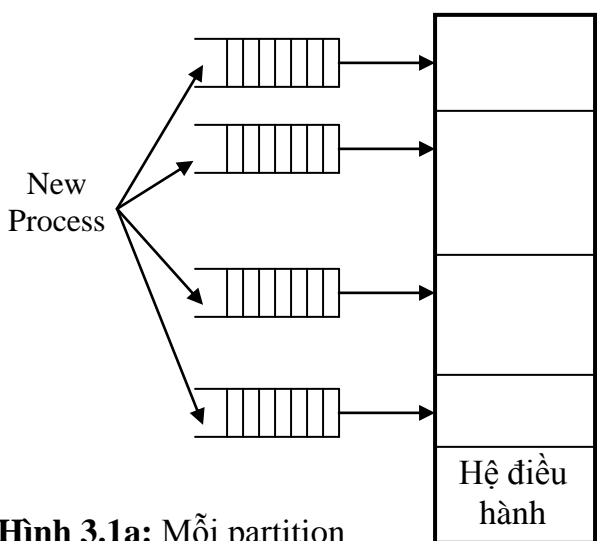
- Thứ nhất, khi kích thước của một chương trình là quá lớn so với kích thước của một partition thì người lập trình phải thiết kế chương trình theo cấu trúc overlay, theo đó chỉ những phần chia cần thiết của chương trình mới được nạp vào bộ nhớ chính khi khởi tạo chương trình, sau đó người lập trình phải nạp tiếp các modul cần thiết khác vào đúng partition của chương trình và sẽ ghi đè lên bất kỳ chương trình hoặc dữ liệu ở trong đó. Cấu trúc chương trình overlay tiết kiệm được bộ nhớ nhưng yêu cầu cao ở người lập trình.
- Thứ hai, khi kích thước của một chương trình nhỏ hơn kích thước của một partition hoặc quá lớn so với kích thước của một partition nhưng không phải là bội số của kích thước một partition thì dễ xảy ra hiện tượng phân mảnh bên trong (internal fragmentation) bộ nhớ, gây lãng phí bộ nhớ. Ví dụ,

nếu có 3 không gian trống kích thước 30K nằm rải rác trên bộ nhớ, thì cũng sẽ không nạp được một modul chương trình có kích thước 12K, hiện tượng này được gọi là hiện tượng phân mảnh bên trong.

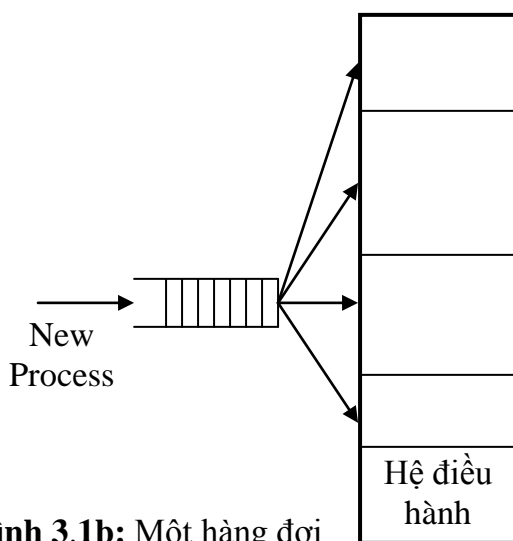
Cả hai vấn đề trên có thể được khắc phục bằng cách sử dụng các phân vùng có kích thước không bằng nhau.

Việc đưa một tiến trình vào partition trong hệ thống đa chương với phân vùng cố định kích thước không bằng nhau sẽ phức tạp hơn nhiều so với trường hợp các phân vùng có kích thước bằng nhau. Với các partition có kích thước không bằng nhau thì có hai cách để lựa chọn khi đưa một tiến trình vào partition:

- Mỗi phân vùng có một hàng đợi tương ứng, theo đó mỗi tiến trình khi cần được nạp vào bộ nhớ nó sẽ được đưa đến hàng đợi của phân vùng có kích thước vừa đủ để chứa nó, để vào/để đợi được vào phân vùng. Cách tiếp cận này sẽ đơn giản trong việc đưa một tiến trình từ hàng đợi vào phân vùng vì không có sự lựa chọn nào khác ở đây, khi phân vùng mà tiến trình đợi trống nó sẽ được đưa vào phân vùng đó. Tuy nhiên các tiếp cận này kém linh động vì có thể có một phân vùng đang trống, trong khi đó có nhiều tiến trình đang phải phải đợi để được nạp vào các phân vùng khác, điều này gây lãng phí trong việc sử dụng bộ nhớ.
- Hệ thống dùng một hàng đợi chung cho tất cả các phân vùng, theo đó tất cả các tiến trình muốn được nạp vào phân vùng nhưng chưa được vào sẽ được đưa vào hàng đợi chung này. Sau đó nếu có một phân vùng trống thì hệ thống sẽ xem xét để đưa một tiến trình có kích thước vừa đủ vào phân vùng trống đó. Cách tiếp cận này linh động hơn so với việc sử dụng nhiều hàng đợi như ở trên, nhưng việc chọn một tiến trình trong hàng đợi để đưa vào phân vùng là một việc làm khá phức tạp của hệ điều hành vì nó phải dựa vào nhiều yếu tố khác nhau như: độ ưu tiên của tiến trình, trạng thái hiện tại của tiến trình, các mối quan hệ của tiến trình,...



**Hình 3.1a:** Mỗi partition có một hàng đợi riêng



**Hình 3.1b:** Một hàng đợi chung cho tất cả partition

Mặc dầu sự phân vùng cố định với kích thước không bằng nhau cung cấp một sự mềm dẻo hơn so với phân vùng cố định với kích thước bằng nhau, nhưng cả hai loại này còn một số hạn chế sau đây:

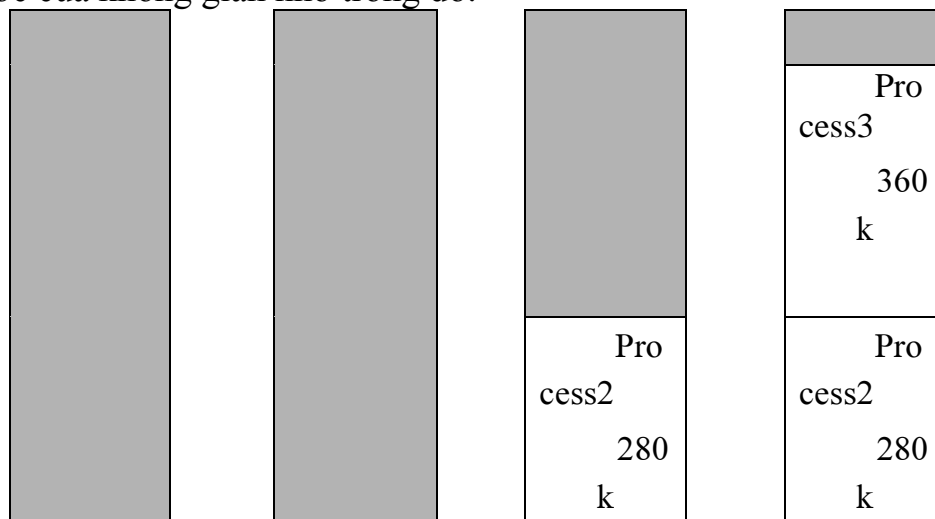
- Số lượng các tiến trình có thể hoạt động trong hệ thống tại một thời điểm phụ thuộc vào số lượng các phân vùng cố định trên bộ nhớ.
- Tương tự như trên, nếu kích thước của tiến trình nhỏ hơn kích thước của một phân vùng thì có thể dẫn đến hiện tượng phân mảnh nội vi gây lãng phí trong việc sử dụng bộ nhớ.

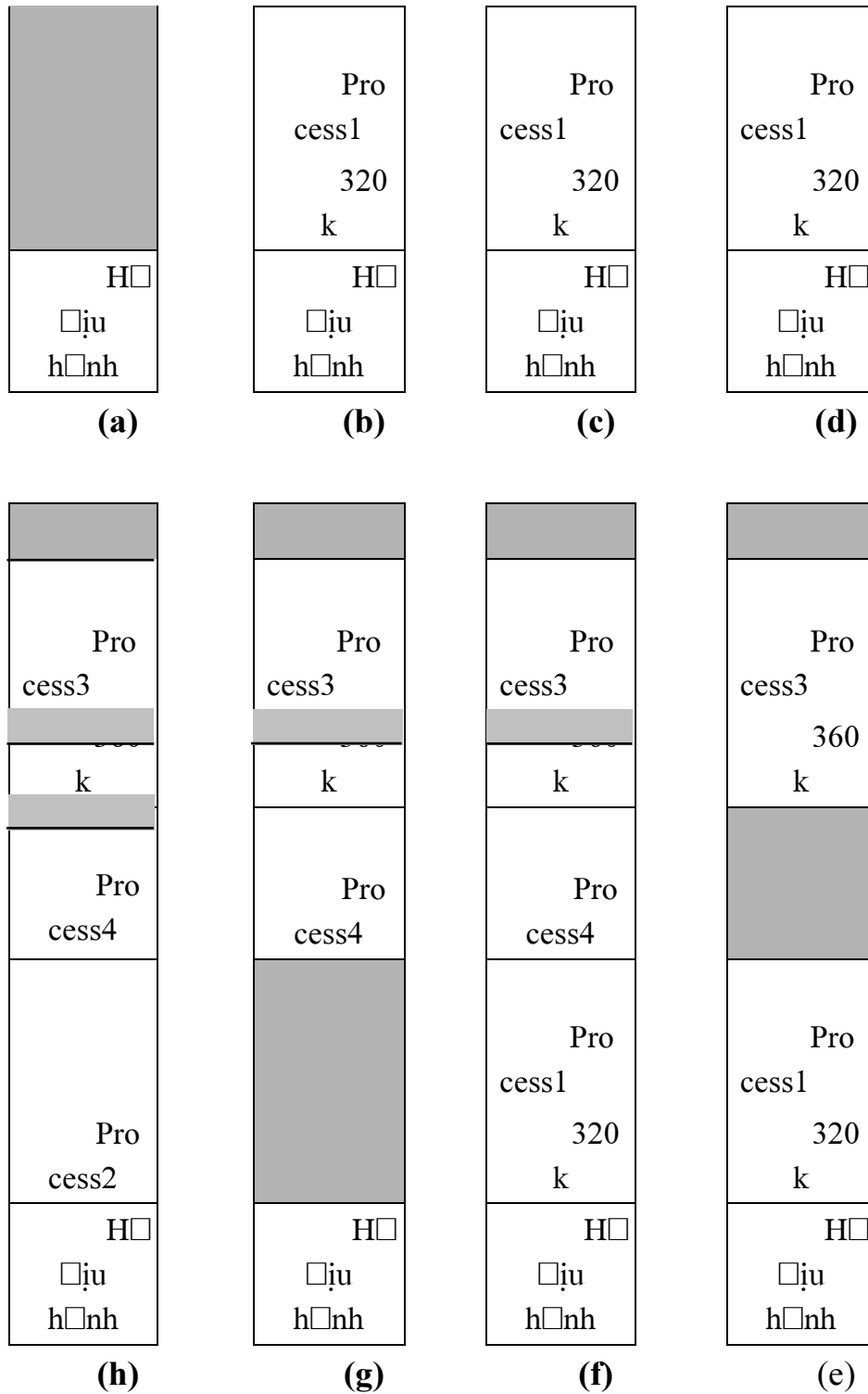
Sự phân vùng cố định ít được sử dụng trong các hệ điều hành hiện nay.

### III.2.6. Kỹ thuật phân vùng động (Dynamic Partitioning)

Để khắc phục một vài hạn chế của kỹ thuật phân vùng cố định, kỹ thuật phân vùng động ra đời. Kỹ thuật này thường được sử dụng trong các hệ điều hành gần đây như hệ điều hành mainframe của IBM, hệ điều hành OS/MVT,...

Trong kỹ thuật phân vùng động, số lượng các phân vùng trên bộ nhớ và kích thước của mỗi phân vùng là có thể thay đổi. Tức là phần user program trên bộ nhớ không được phân chia trước mà nó chỉ được ấn định sau khi đã có một tiến trình được nạp vào bộ nhớ chính. Khi có một tiến trình được nạp vào bộ nhớ nó được hệ điều hành cấp cho nó không gian vừa đủ để chứa tiến trình, phần còn lại để sẵn sàng cấp cho tiến trình khác sau này. Khi một tiến trình kết thúc nó được đưa ra ngoài và phần không gian bộ nhớ mà tiến trình này trả lại cho hệ điều hành sẽ được hệ điều hành cấp cho tiến trình khác, cả khi tiến trình này có kích thước nhỏ hơn kích thước của không gian nhớ trống đó.



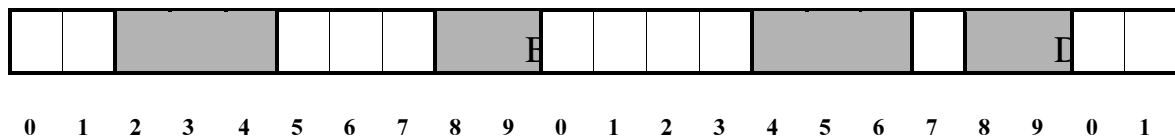


**Hình 3.2:** Kết quả của sự phân trang động với thứ tự nạp các tiến trình.

Hình vẽ 3.2 trên đây minh họa cho quá trình nạp/kết thúc các tiến trình theo thứ tự: nạp process1, nạp process2, nạp process3, kết thúc process2, nạp process4,

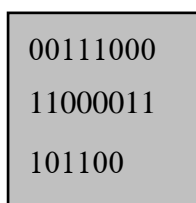
kết thúc process1, nạp process2 vào lại, trong hệ thống phân vùng động. Như vậy dần dần trong bộ nhớ hình thành nhiều không gian nhớ có kích thước nhỏ không đủ chứa các tiến trình nằm rải rác trên bộ nhớ chính, hiện tượng này được gọi là hiện tượng phân mảnh bên ngoài (external fragmentation). Để chống lại sự lãng phí bộ nhớ do phân mảnh, thỉnh thoảng hệ điều hành phải thực hiện việc sắp xếp lại bộ nhớ, để các không gian nhớ nhỏ rời rạc nằm liền kề lại với nhau tạo thành một khối nhớ có kích thước đủ lớn để chứa được một tiến trình nào đó. Việc làm này làm chậm tốc độ của hệ thống, hệ điều hành phải chi phí cao cho việc này, đặc biệt là việc tái định vị các tiến trình khi một tiến trình bị đưa ra khỏi bộ nhớ và được nạp vào lại bộ nhớ để tiếp tục hoạt động.

Trong kỹ thuật phân vùng động này hệ điều hành phải đưa ra các cơ chế thích hợp để quản lý các khối nhớ đã cấp phát hay còn trống trên bộ nhớ. Hệ điều hành sử dụng 2 cơ chế: Bản đồ bit và Danh sách liên kết. Trong cả 2 cơ chế này hệ điều hành đều chia không gian nhớ thành các đơn vị cấp phát có kích thước bằng nhau, các đơn vị cấp phát liên tiếp nhau tạo thành một khối nhớ (block), hệ điều hành cấp phát các block này cho các tiến trình khi nạp tiến trình vào bộ nhớ.

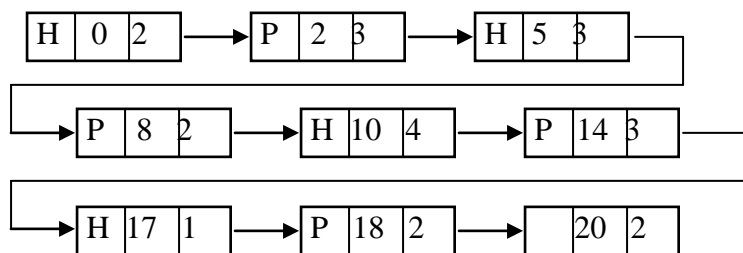


**Hình 3.3a:** Một đoạn nhớ bao gồm 22 đơn vị cấp phát, tạo thành 9 block, trong đó có 4 block đã cấp phát (tô đậm, kí hiệu là P) cho các tiến trình: A, B, C, D và 5 block chưa được cấp phát (đề trắng, kí hiệu là H).

- Trong cơ chế bản đồ bit: mỗi đơn vị cấp phát được đại diện bởi một bit trong bản đồ bit. Đơn vị cấp phát còn trống được đại diện bằng bit 0, ngược lại đơn vị cấp phát được đại diện bằng bit 1. Hình 3.3b là bản đồ bit của khối nhớ ở trên.



**Hình 3.3b:** quản lý các đơn vị cấp phát bằng bản đồ bit.



**Hình 3.3c:** quản lý các đơn vị cấp phát bằng danh sách liên kết.

- Trong cơ chế danh sách liên kết: Mỗi block trên bộ nhớ được đại diện

bởi một phần tử trong danh sách liên kết, mỗi phần tử này gồm có 3 trường chính: trường thứ nhất cho biết khối nhớ đã cấp phát (P: process) hay đang còn trống (H: Hole), trường thứ hai cho biết thứ tự của đơn vị cấp phát đầu tiên trong block, trường thứ ba cho biết block gồm bao nhiêu đơn vị cấp phát. Hình 3.3c là danh sách liên kết của khối nhớ ở trên.

Như vậy khi cần nạp một tiến trình vào bộ nhớ thì hệ điều hành phải dựa vào bản đồ bit hoặc danh sách liên kết để tìm ra một block có kích thước đủ để nạp tiến trình. Sau khi thực hiện một thao tác cấp phát hoặc sau khi đưa một tiến trình ra khỏi bộ nhớ thì hệ điều hành phải cập nhật lại bản đồ bit hoặc danh sách liên kết, điều này có thể làm giảm tốc độ thực hiện của hệ thống.

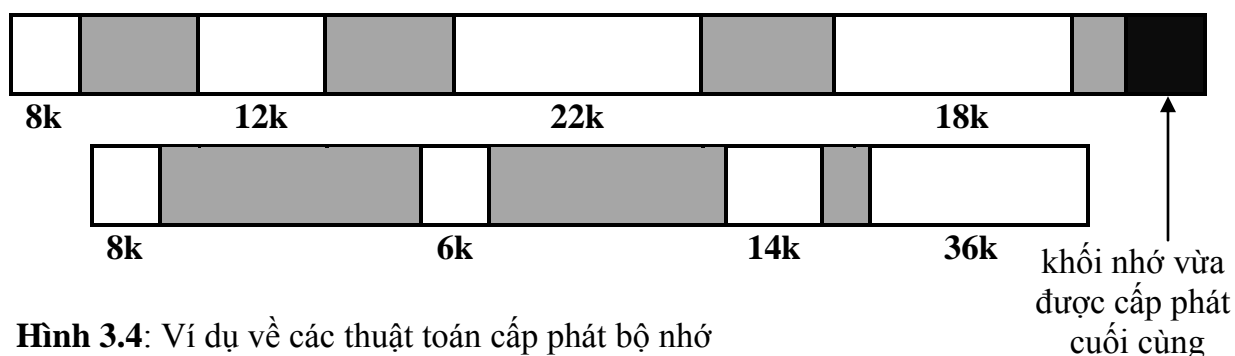
Chọn kích thước của một đơn vị cấp phát là một vấn đề quan trọng trong thiết kế, nếu kích thước đơn vị cấp phát nhỏ thì bản đồ bit sẽ lớn, hệ thống phải tốn bộ nhớ để chứa nó. Nếu kích thước của một đơn vị cấp phát lớn thì bản đồ bit sẽ nhỏ, nhưng sự lãng phí bộ nhớ ở đơn vị cấp phát cuối cùng của một tiến trình sẽ lớn khi kích thước của tiến trình không phải là bội số của một đơn vị cấp phát. Điều vừa trình bày cũng đúng trong trường hợp danh sách liên kết.

Danh sách liên kết có thể được sắp xếp theo thứ tự tăng dần hoặc giảm dần của kích thước hoặc địa chỉ, điều này giúp cho việc tìm khối nhớ trống có kích thước vừa đủ để nạp các tiến trình theo các thuật toán dưới đây sẽ đạt tốc độ nhanh hơn và hiệu quả cao hơn. Một số hệ điều hành tổ chức 2 danh sách liên kết riêng để theo dõi các đơn vị cấp phát trên bộ nhớ, một danh sách để theo dõi các block đã cấp phát và một danh sách để theo dõi các block còn trống. Cách này giúp việc tìm các khối nhớ trống nhanh hơn, chỉ tìm trên danh sách các khối nhớ trống, nhưng tốn thời gian nhiều hơn cho việc cập nhật danh sách sau mỗi thao tác cấp phát, vì phải thực hiện trên cả hai danh sách.

Khi có một tiến trình cần được nạp vào bộ nhớ mà trong bộ nhớ có nhiều hơn một khối nhớ trống (Free Block) có kích thước lớn hơn kích thước của tiến trình đó, thì hệ điều hành phải quyết định chọn một khối nhớ trống phù hợp nào để nạp tiến trình sao cho việc lựa chọn này dẫn đến việc sử dụng bộ nhớ chính là hiệu quả nhất. Có 3 thuật toán mà hệ điều hành sử dụng trong trường hợp này, đó là: Best-fit, First-fit, và Next-fit. Cả 3 thuật toán này đều phải chọn một khối nhớ trống có kích thước bằng hoặc lớn hơn kích thước của tiến trình cần nạp vào, nhưng nó có các điểm khác nhau cơ bản sau đây:

- **Best-fit:** chọn khối nhớ có kích thước vừa đúng bằng kích thước của tiến trình cần được nạp vào bộ nhớ.
- **First-fit:** trong trường hợp này hệ điều hành sẽ bắt đầu quét qua các khối nhớ trống bắt đầu từ khối nhớ trống đầu tiên trong bộ nhớ, và sẽ chọn khối nhớ trống đầu tiên có kích thước đủ lớn để nạp tiến trình.





**Hình 3.4:** Ví dụ về các thuật toán cấp phát bộ nhớ

- **Next-fit:** tương tự như First-fit nhưng ở đây hệ điều hành bắt đầu quét từ khối nhớ trống kế sau khối nhớ vừa được cấp phát và chọn khối nhớ trống kế tiếp đủ lớn để nạp tiến trình.

Hình vẽ 3.4 cho thấy hiện tại trên bộ nhớ có các khối nhớ chưa được cấp phát theo thứ tự là: 8k, 12k, 22k, 18k, 8k, 6k, 14k, 36k. Trong trường hợp này nếu có một tiến trình có kích thước 16k cần được nạp vào bộ nhớ, thì hệ điều hành sẽ nạp nó vào:

- khối nhớ 22k nếu theo thuật toán First-fit
- khối nhớ 18k nếu theo thuật toán Best-fit
- khối nhớ 36k nếu theo thuật toán Next-fit

Như vậy nếu theo Best-fit thì sẽ xuất hiện một khối phân mảnh 2k, nếu theo First-fit thì sẽ xuất hiện một khối phân mảnh 6k, nếu theo Next-fit thì sẽ xuất hiện một khối phân mảnh 20k.

Các hệ điều hành không cài đặt cố định trước một thuật toán nào, tùy vào trường hợp cụ thể mà nó chọn cấp phát theo một thuật toán nào đó, sao cho chi phí về việc cấp phát là thấp nhất và hạn chế được sự phân mảnh bộ nhớ sau này. Việc chọn thuật toán này thường phụ thuộc vào thứ tự swap và kích thước của tiến trình. Thuật toán First-fit được đánh giá là đơn giản, dễ cài đặt nhưng mang lại hiệu quả cao nhất đặc biệt là về tốc độ cấp phát. Về hiệu quả thuật toán Next-fit không bằng First-fit, nhưng nó thường xuyên sử dụng được các khối nhớ trống ở cuối vùng nhớ, các khối nhớ ở vùng này thường có kích thước lớn nên có thể hạn chế được sự phân mảnh, theo ví dụ trên thì việc xuất hiện một khối nhớ trống 20k sau khi cấp một tiến trình 16k thì không thể gọi là phân mảnh được, nhưng nếu tiếp tục như thế thì dễ dẫn đến sự phân mảnh lớn ở cuối bộ nhớ. Thuật toán Best-fit, không như tên gọi của nó, đây là một thuật toán có hiệu suất thấp nhất, trong trường hợp này hệ điều hành phải duyệt qua tất cả các khối nhớ trống để tìm ra một khối nhớ có kích thước vừa đủ để chứa tiến trình vừa yêu cầu, điều này làm giảm tốc độ cấp phát của hệ điều hành. Mặt khác với việc chọn kích thước vừa đủ có thể dẫn đến sự phân

mảnh lớn trên bộ nhớ, tức là có quá nhiều khối nhớ có kích thước quá nhỏ trên bộ nhớ, nhưng nếu xét về mặt lãng phí bộ nhớ tại thời điểm cấp phát thì thuật toán này làm lãng phí ít nhất. Tóm lại, khó có thể đánh giá về hiệu quả sử dụng của các thuật toán này, vì hiệu quả của nó được xét trong “tương lai” và trên nhiều khía cạnh khác nhau chứ không phải chỉ xét tại thời điểm cấp phát. Và hơn nữa trong bản thân các thuật toán này đã có các mâu thuẫn với nhau về hiệu quả sử dụng của nó.

Do yêu cầu của công tác cấp phát bộ nhớ của hệ điều hành, một tiến trình đang ở trên bộ nhớ có thể bị đưa ra ngoài (swap-out) để dành chỗ nạp một tiến trình mới có yêu cầu, và tiến trình này sẽ được nạp vào lại (swap-in) bộ nhớ tại một thời điểm thích hợp sau này. Vấn đề đáng quan tâm ở đây là tiến trình có thể được nạp vào lại phân vùng khác với phân vùng mà nó được nạp vào lần đầu tiên. Có một lý do khác khiến các tiến trình phải thay đổi vị trí nạp so với ban đầu là khi có sự liên kết giữa các mô đun tiến trình của một chương trình thì các tiến trình phải dịch chuyển ngay cả khi chúng đã nằm trên bộ nhớ chính. Sự thay đổi vị trí/địa chỉ nạp này sẽ ảnh hưởng đến các thao tác truy xuất dữ liệu của chương trình vì nó sẽ khác với các địa chỉ tương đối mà người lập trình đã sử dụng trong code của chương trình. Ngoài ra khi một tiến trình được nạp vào bộ nhớ lần đầu tiên thì tất cả các địa chỉ tương đối được tham chiếu trong code chương trình được thay thế bằng địa chỉ tuyệt đối trong bộ nhớ chính, địa chỉ này được xác định bởi địa chỉ cơ sở, nơi tiến trình được nạp. Ví dụ trong chương trình có code truy xuất đến địa chỉ tương đối 100k, nếu chương trình này được nạp vào phân vùng 1 có địa chỉ bắt đầu là 100k thì địa chỉ truy xuất là 200k, nhưng nếu chương trình được nạp vào phân vùng 2 có địa chỉ bắt đầu là 200k, thì địa chỉ truy xuất sẽ là 300k. Để giải quyết vấn đề này hệ điều hành phải thực hiện các yêu cầu cần thiết của công tác tái định vị một tiến trình vào lại bộ nhớ. Ngoài ra ở đây hệ điều hành cũng phải tính đến việc bảo vệ các tiến trình trên bộ nhớ tránh tình trạng một tiến trình truy xuất đến vùng nhớ của tiến trình khác. Trong trường hợp này hệ điều hành sử dụng 2 thanh ghi đặc biệt:

- Thanh ghi cơ sở (base register): dùng để ghi địa chỉ cơ sở của tiến trình tiến trình được nạp vào bộ nhớ.
- Thanh ghi giới hạn (limit register): dùng để ghi địa chỉ cuối cùng của tiến trình trong bộ nhớ.

Khi một tiến trình được nạp vào bộ nhớ thì hệ điều hành sẽ ghi địa chỉ bắt đầu của phân vùng được cấp phát cho tiến trình vào thanh ghi cơ sở và địa chỉ cuối cùng của tiến trình vào thanh ghi giới hạn. Việc thiết lập giá trị của các thanh ghi này được thực hiện cả khi tiến trình lần đầu tiên được nạp vào bộ nhớ và khi tiến trình được swap in vào lại bộ nhớ. Theo đó mỗi khi tiến trình thực hiện một thao tác truy xuất bộ nhớ thì hệ thống phải thực hiện 2 bước: Thứ nhất, cộng địa chỉ ô nhớ do tiến trình phát ra với giá trị địa chỉ trong thanh ghi cơ sở để có được địa chỉ tuyệt đối của ô nhớ cần truy xuất. Thứ hai, địa chỉ kết quả ở trên sẽ được so sánh

với giá trị địa chỉ trong thành ghi giới hạn. Nếu địa chỉ nằm trong phạm vi giới hạn thì hệ điều hành cho phép tiến trình truy xuất bộ nhớ, ngược lại thì có một ngắt về lỗi truy xuất bộ nhớ được phát sinh và hệ điều hành không cho phép tiến trình truy xuất vào vị trí bộ nhớ mà nó yêu cầu. Như vậy việc bảo vệ truy xuất bất hợp lệ được thực hiện dễ dàng ở đây.

Trong hệ thống đa chương sử dụng sự phân vùng động, nếu có một tiến trình mới cần được nạp vào bộ nhớ, trong khi bộ nhớ không còn chỗ trống và tất cả các tiến trình trên bộ nhớ đều ở trạng thái khoá (blocked), thì hệ thống phải đợi cho đến khi có một tiến trình được chuyển sang trạng thái không bị khoá (unblocked) để tiến trình này có điều kiện trả lại không gian nhớ mà nó chiếm giữ cho hệ thống: tiến trình hoạt động và kết thúc, tiến trình bị đưa ra khỏi bộ nhớ chính,..., để hệ thống nạp tiến trình vừa có yêu cầu. Sự chờ đợi này làm lãng phí thời gian xử lý của processor. Để tiết kiệm thời gian xử lý của processor trong trường hợp này hệ điều hành chọn ngay một tiến trình đang ở trạng thái khoá để đưa ra ngoài lấy không gian nhớ trống đó cấp cho tiến trình vừa có yêu cầu mà không phải đợi như ở trên. Hệ điều hành sử dụng nhiều thuật toán khác nhau cho việc chọn một tiến trình để thay thế trong trường hợp này, tất cả các thuật toán này đều hướng tới mục đích: tiết kiệm thời gian xử lý của processor, tốc độ thay thế cao, sử dụng bộ nhớ hiệu quả nhất và đặc biệt là không để dẫn đến sự trì trệ hệ thống. Chúng ta sẽ thảo

luyện rõ hơn về vấn đề này ở phần sau của chương này.

➤ **Chú ý:** Một nhược điểm lớn của các kỹ thuật ở trên là dẫn đến hiện tượng phân mảnh bộ nhớ bên trong và bên ngoài (internal, external) gây lãng phí bộ nhớ nên hiệu quả sử dụng bộ nhớ kém. Để khắc phục hệ điều hành sử dụng các kỹ thuật phân trang hoặc phân đoạn bộ nhớ.

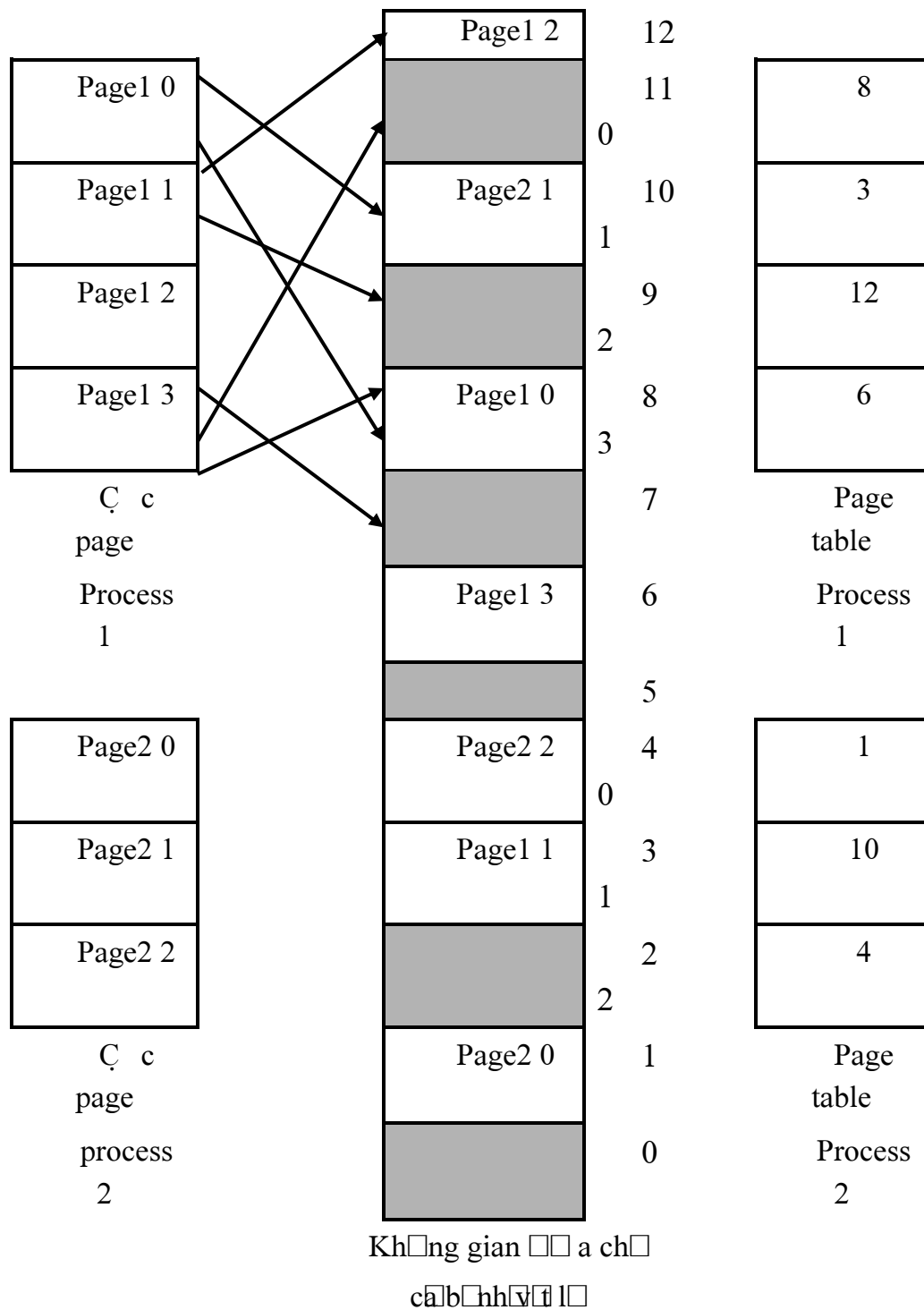
### III.2.7. Kỹ thuật phân trang đơn (Simple Paging)

Trong kỹ thuật này không gian địa chỉ bộ nhớ vật lý được chia thành các phần có kích thước cố định bằng nhau, được đánh số địa chỉ bắt đầu từ 0 và được gọi là các khung trang (page frame). Không gian địa chỉ của các tiến trình cũng được chia thành các phần có kích thước bằng nhau và bằng kích thước của một khung trang, được gọi là các trang (page) của tiến trình.

Khi một tiến trình được nạp vào bộ nhớ thì các trang của tiến trình được nạp vào các khung trang còn trống bất kỳ, có thể không liên tiếp nhau, của bộ nhớ. Khi hệ điều hành cần nạp một tiến trình có  $n$  trang vào bộ nhớ thì nó phải tìm đủ  $n$  khung trang trống để nạp tiến trình này. Nếu kích thước của tiến trình không phải là bội số của kích thước một khung trang thì sẽ xảy ra hiện tượng phân mảnh nội vi ở khung trang chứa trang cuối cùng của tiến trình. Ở đây không xảy ra hiện tượng phân mảnh ngoại vi. Trên bộ nhớ có thể tồn tại các trang của nhiều tiến trình khác nhau. Khi một tiến trình bị swap-out thì các khung trang mà tiến trình này chiếm giữ sẽ được giải phóng để hệ điều hành có thể nạp các trang tiến trình khác.

Trong kỹ thuật này hệ điều hành phải đưa ra các cơ chế thích hợp để theo dõi trạng thái của các khung trang (còn trống hay đã cấp phát) trên bộ nhớ và các khung trang đang chứa các trang của một tiến trình của các tiến trình khác nhau trên bộ nhớ. Hệ điều hành sử dụng một danh sách để ghi số hiệu của các khung trang còn trống trên bộ nhớ, hệ điều hành dựa vào danh sách này để tìm các khung trang trống trước khi quyết định nạp một tiến trình vào bộ nhớ, danh sách này được cập nhật ngay sau khi hệ điều hành nạp một tiến trình vào bộ nhớ, được kết thúc hoặc bị swap out ra bên ngoài.

Hệ điều hành sử dụng các bảng trang (PCT: page control table) để theo dõi vị trí các trang tiến trình trên bộ nhớ, mỗi tiến trình có một bảng trang riêng. Bảng trang bao gồm nhiều phần tử, thường là bằng số lượng trang của một tiến trình mà bảng trang này theo dõi, các phần tử được đánh số bắt đầu từ 0. Phần tử 0 chứa số hiệu của khung trang đang chứa trang 0 của tiến trình, phần tử 1 chứa số hiệu của khung trang đang chứa trang 1 của tiến trình, ... Các bảng trang có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng trang lớn thì nó được chứa trong bộ nhớ chính, khi đó hệ điều hành sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng trang, thanh ghi này được gọi là thanh ghi PTBR: page table base register.



(a)

(b)

(c)

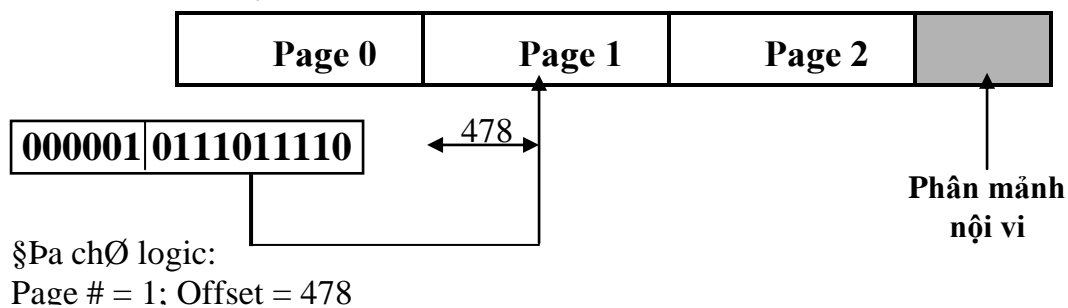
**Hình 3.6:** Các trang của 2 tiến trình process 1 và process 2 (a), được nạp vào bộ nhớ (b), và 2 bảng trang tương ứng của nó (c).

Trong kỹ thuật phân trang này khi cần truy xuất bộ nhớ CPU phải phát ra một địa chỉ logic gồm 2 thành phần: Số hiệu trang (Page): cho biết số hiệu trang

tương ứng cần truy xuất. Địa chỉ tương đối trong trang (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của trang để xác định địa chỉ vật lý của ô nhớ cần truy xuất. Việc chuyển đổi từ địa chỉ logic sang địa chỉ vật lý do processor thực hiện.

Kích thước của mỗi trang hay khung trang do phần cứng quy định và thường là lũy thừa của 2, biến đổi từ 512 byte đến 8192 byte. Nếu kích thước của không gian địa chỉ là  $2^m$  và kích thước của trang là  $2^n$  thì m-n bit cao của địa chỉ logic là số hiệu trang (page) và n bit còn lại là địa chỉ tương đối trong trang (offset). Ví dụ: nếu địa chỉ logic gồm 16 bit, kích thước của mỗi trang là 1K = 1024byte ( $2^{10}$ ), thì có 6 bit dành cho số hiệu trang, như vậy một chương trình có thể có tối đa  $2^6 = 64$  trang mỗi trang 1KB. Trong trường hợp này nếu CPU phát ra một giá trị địa chỉ 16 bit là: 0000010111011110 = 1502, thì thành phần số hiệu trang là 000001 = 1, thành phần offset là 0111011110 = 478.

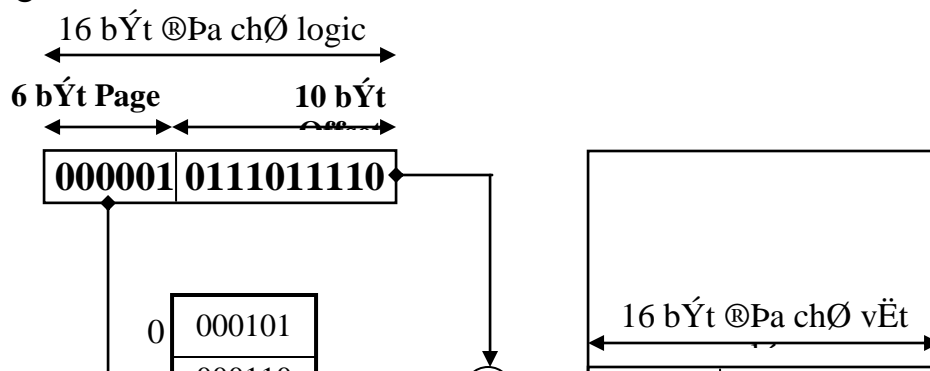
Hình minh họa:



**Hình 3.7a:** Các khung trang của bộ nhớ và địa chỉ logic

Việc chuyển từ địa chỉ logic sang địa chỉ vật lý được thực hiện theo các bước sau:

- Trích ra m-n bit trái nhất (thấp nhất) của địa chỉ logic để xác định số hiệu trang cần truy xuất.
- Sử dụng số hiệu trang ở trên để chỉ đến phần tử tương ứng trong bảng trang của tiến trình, để xác định khung trang tương ứng, ví dụ là k.
- Địa chỉ vật lý bắt đầu của khung trang là  $k \times 2^n$ , và địa chỉ vật lý của byte cần truy xuất là số hiệu trang cộng với giá trị offset. Địa chỉ vật lý không cần tính toán, nó dễ dàng có được bằng cách nối số hiệu khung trang với giá trị offset.



Trong sơ đồ ví dụ ở trên, chúng ta có địa chỉ logic là: 0000010111011110, với số hiệu trang là 1, offset là 478, giả định rằng trang này thường trú trong bộ nhớ chính tại khung tang 6 = 000110. Thì địa chỉ vật lý là khung trang số 6 và offset là  $478 = 0001100111011110$ .

➤ **Nhận xét về kỹ thuật phân trang:**

- Có thể thấy sự phân trang được mô tả ở đây tương tự như sự phân vùng cố định. Sự khác nhau là với phân trang các phân vùng có kích thước nhỏ hơn, một chương trình có thể chiếm giữa nhiều hơn một phân vùng, và các phân vùng này có thể không liên kề với nhau.
- Kỹ thuật phân trang loại bỏ được hiện tượng phân mảnh ngoại vi, nhưng vẫn có thể xảy ra hiện tượng phân mảnh nội vi khi kích thước của tiến trình không đúng bằng bội số kích thước của một trang, khi đó khung trang cuối cùng sẽ không được sử dụng hết.
- Khi cần truy xuất đến dữ liệu hay chỉ thị trên bộ nhớ thì hệ thống phải cần một lần truy xuất đến bảng trang, điều này có thể làm giảm tốc độ truy xuất bộ nhớ. Để khắc phục hệ điều hành sử dụng thêm một bảng trang cache, để lưu trữ các trang bộ nhớ vừa được truy cập gần đây nhất. Bảng trang cache này sẽ được sử dụng mỗi khi CPU phát ra một địa chỉ cần truy xuất.
- Mỗi hệ điều hành có một cơ chế tổ chức bảng trang riêng, đa số các hệ điều hành đều tạo cho mỗi tiến trình một bảng trang riêng khi nó được nạp vào bộ nhớ chính. Bảng trang lớn sẽ tốn bộ nhớ để chứa nó.
- Để bảo vệ các khung trang hệ điều hành đưa thêm một bit bảo vệ vào bảng trang. Theo đó mỗi khi tham khảo vào bảng trang để truy xuất bộ nhớ hệ hống sẽ kiểm tra các thao tác truy xuất trên khung trang tương ứng có hợp lệ với thuộc tính bảo vệ của nó hay không.

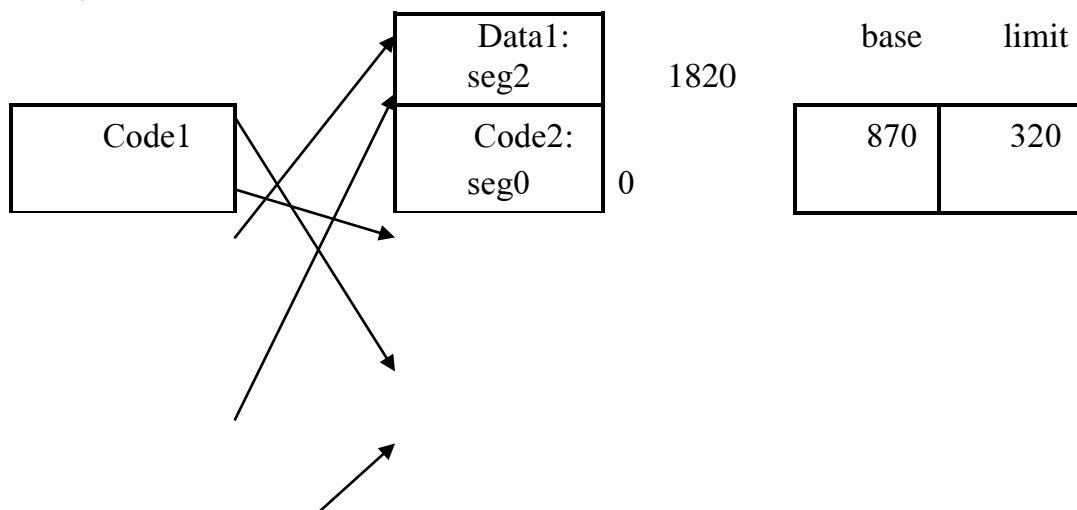
- Sự phân trang không phản ánh được cách mà người sử dụng nhìn nhận về bộ nhớ. Với người sử dụng, bộ nhớ là một tập các đối tượng chương trình và dữ liệu như các segment, các thư viện, .... và các biến, các vùng nhớ chia sẻ, stack, ... . Vấn đề đặt ra là tìm một cách thức biểu diễn bộ nhớ sao cho nó gần với cách nhìn nhận của người sử dụng hơn. Kỹ thuật phân đoạn bộ nhớ có thể thực hiện được mục tiêu này.

### III.2.8. Kỹ thuật phân đoạn đơn (Simple Segmentation)

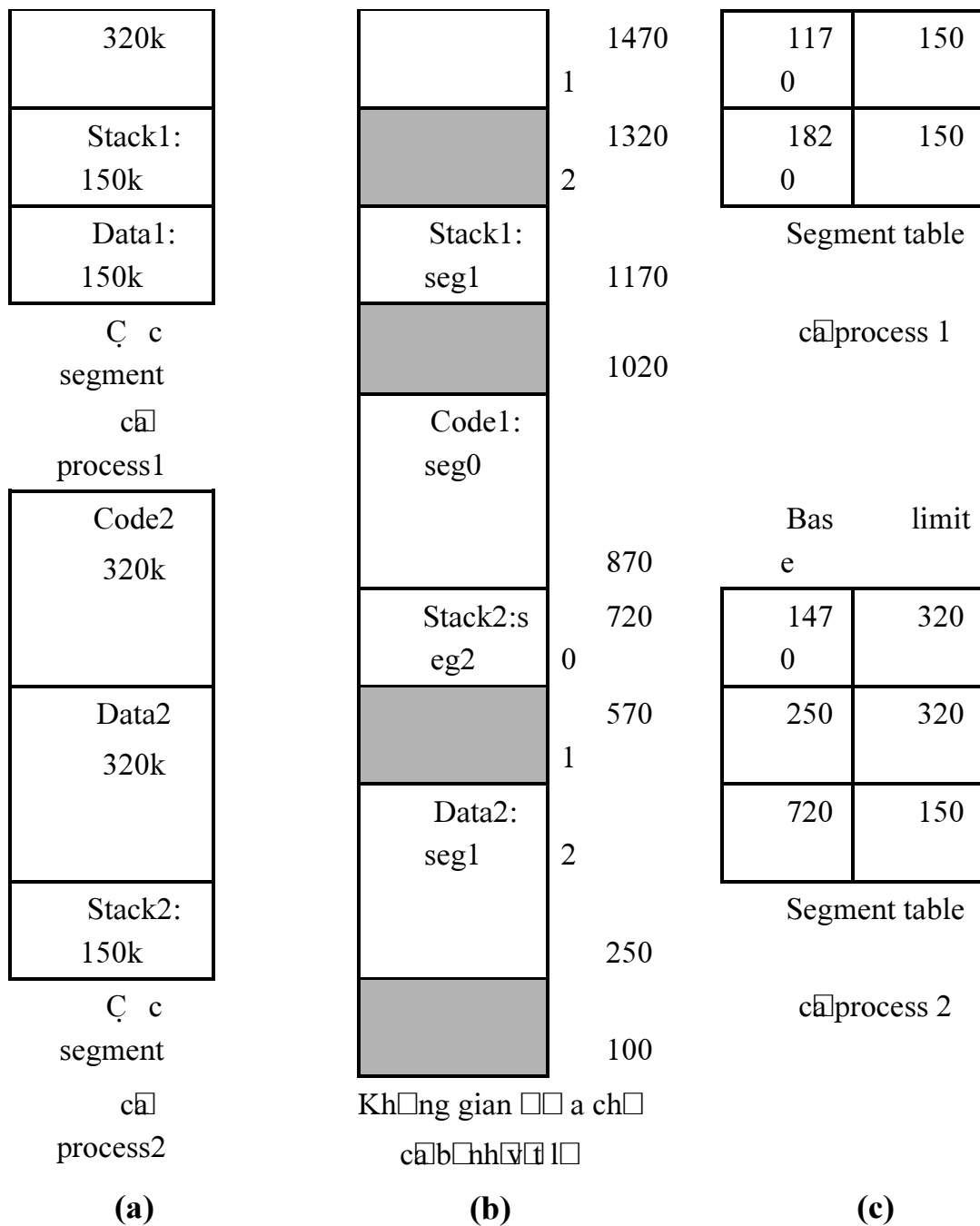
Trong kỹ thuật này không gian địa chỉ bộ nhớ vật lý được chia thành các phần cố định có kích thước không bằng nhau, được đánh số bắt đầu từ 0, được gọi là các phân đoạn (segment). Mỗi phân đoạn bao gồm số hiệu phân đoạn và kích thước của nó. Không gian địa chỉ của các tiến trình kể cả các dữ liệu liên quan cũng được chia thành các đoạn khác nhau và không nhất thiết phải có kích thước bằng nhau, thông thường mỗi thành phần của một chương trình/tiến trình như: code, data, stack, subprogram, ..., là một đoạn.

Khi một tiến trình được nạp vào bộ nhớ thì tất cả các đoạn của nó sẽ được nạp vào các phân đoạn còn trống khác nhau trên bộ nhớ. Các phân đoạn này có thể không liên tiếp nhau. Xem hình 3.8.

Để theo dõi các đoạn của các tiến trình khác nhau trên bộ nhớ, hệ điều hành sử dụng các bảng phân đoạn (SCT: Segment control Table) tiến trình, thông thường một tiến trình có một bảng phân đoạn riêng. Mỗi phần tử trong bảng phân đoạn gồm tối thiểu 2 trường: trường thứ nhất cho biết địa chỉ cơ sở (base) của phân đoạn mà đoạn chương trình tương ứng được nạp, trường thứ hai cho biết độ dài/giới hạn (length/limit) của phân đoạn, trường này còn có tác dụng dùng để kiểm soát sự truy xuất bất hợp lệ của các tiến trình. Các bảng phân đoạn có thể được chứa trong các thanh ghi nếu có kích thước nhỏ, nếu kích thước bảng phân đoạn lớn thì nó được chứa trong bộ nhớ chính, khi đó hệ điều hành sẽ dùng một thanh ghi để lưu trữ địa chỉ bắt đầu nơi lưu trữ bảng phân đoạn, thanh ghi này được gọi là thanh ghi STBR: Segment table base register. Ngoài ra vì số lượng các đoạn của một chương trình/tiến trình có thể thay đổi nên hệ điều hành dùng thêm thanh ghi STLR: Segment table length register, để ghi kích thước hiện tại của bảng phân đoạn. Hệ điều hành cũng tổ chức một danh sách riêng để theo dõi các segment còn trống trên bộ nhớ.



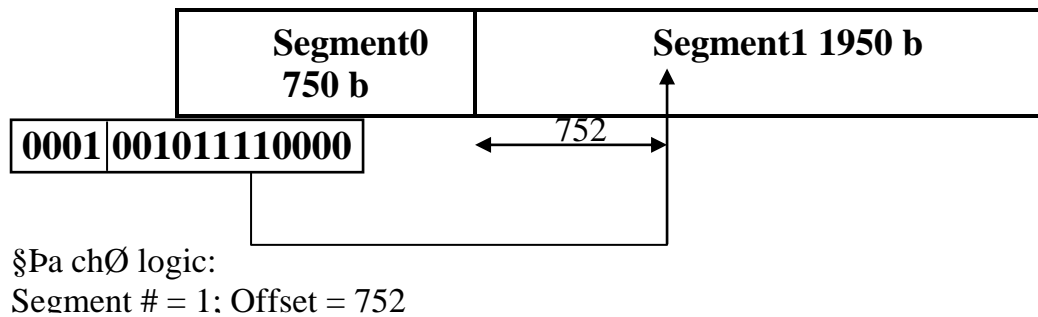




**Hình 3.8:** Các đoạn của 2 tiến trình process 1 và process 2 (a), được nạp vào bộ nhớ (b), và 2 bảng đoạn tương ứng của nó (c).

Trong kỹ thuật này địa chỉ logic mà CPU sử dụng phải gồm 2 thành phần: Số hiệu đoạn (segment): cho biết số hiệu đoạn tương ứng cần truy xuất. Địa chỉ tương đối trong đoạn (Offset): giá trị này sẽ được kết hợp với địa chỉ bắt đầu của đoạn để xác định địa chỉ vật lý của ô nhớ cần truy xuất. Việc chuyển đổi từ địa chỉ logic sang địa chỉ vật lý do processor thực hiện.

Hình minh họa:

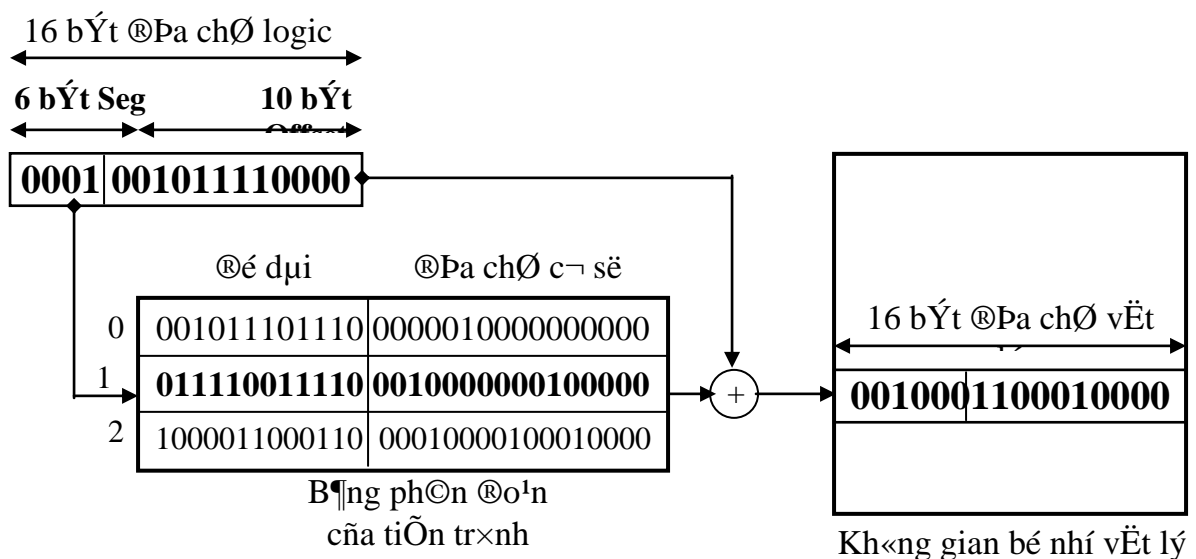


**Hình 3.9a:** Các phân đoạn trên bộ nhớ và địa chỉ logic

Nếu có một địa chỉ logic gồm  $n + m$  bit, thì  $n$  bit trái nhất là số hiệu segment,  $m$  bit phải nhất còn lại là offset. Trong ví dụ minh họa sau đây thì  $n = 4$  và  $m = 12$ , như vậy kích thước tối đa của một segment là  $2^{12} = 4096$  byte. Sau đây là các bước cần thiết của việc chuyển đổi địa chỉ:

- Trích ra  $n$  bit trái nhất của địa chỉ logic để xác định số hiệu của phân đoạn cần truy xuất.
- Sử dụng số hiệu phân đoạn ở trên để chỉ đến phần tử trong bảng phân đoạn của tiến trình, để tìm địa chỉ vật lý bắt đầu của phân đoạn.
- So sánh thành phần offset của địa chỉ logic, được trích ra từ  $m$  bit phải nhất của địa chỉ logic, với thành phần length của phân đoạn. Nếu  $\text{offset} > \text{length}$  thì địa chỉ truy xuất là không hợp lệ.
- Địa chỉ vật lý mong muốn là địa chỉ vật lý bắt đầu của phân đoạn cộng với giá trị offset.

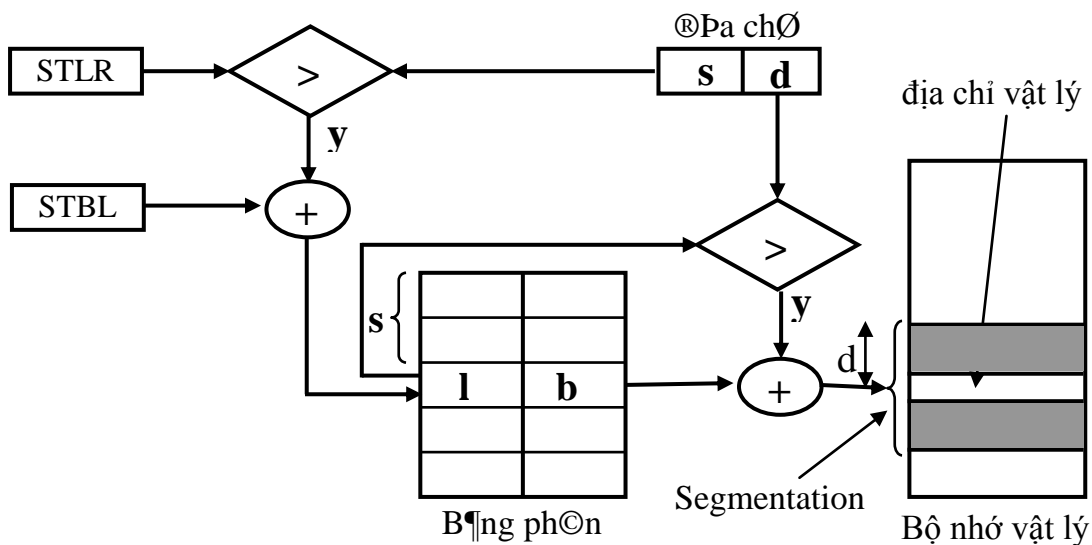
Trong sơ đồ ví dụ sau đây, ta có địa chỉ logic là: 0001001011110000, với số hiệu segment là 1, offset là 752, giả định segment này thường trú trong bộ nhớ chính tại địa chỉ vật lý là 0010000000100000, thì địa chỉ vật lý tương ứng với địa chỉ logic ở trên là: 0010000000100000 + 001011110000 = 0010001100010000.



**Hình 3.9b:** Sơ đồ chuyển đổi địa chỉ logic (segment) – vật lý

➤ **Nhận xét về kỹ thuật phân đoạn:**

- Vì các segment có kích thước không bằng nhau nên sự phân đoạn tương tự như sự phân vùng động. Sự khác nhau là với sự phân đoạn một chương trình có thể chiếm giữ hơn một phân vùng, và các phân vùng này có thể không liền kề với nhau. Sự phân vùng loại trừ được sự phân mảnh nội vi, nhưng như sự phân vùng động nó vẫn xuất hiện hiện tượng phân mảnh ngoại vi.
- Sự phân trang là không tường minh đối với người lập trình, trong khi đó sự phân đoạn là tường minh đối với người lập trình, và nó cung cấp một sự thuận lợi để người lập trình tổ chức chương trình và dữ liệu. Người lập trình hoặc trình biên dịch có thể gán các chương trình và dữ liệu đến các đoạn nhớ khác nhau.



**Hình 3.9c:** Sơ đồ chuyển địa chỉ có sử dụng STLR, STBR và so sánh offset

- Tương tự như trong kỹ thuật phân vùng động, kỹ thuật này cũng phải giải quyết vấn đề cấp phát động, ở đây hệ điều hành thường dùng thuật toán best-fit hay first-fit.
- Kỹ thuật phân đoạn thể hiện được cấu trúc logic của chương trình,

nhưng nó phải cấp phát các khối nhớ có kích thước khác nhau cho các phân đoạn của chương trình trên bộ nhớ vật lý, điều này phức tạp hơn nhiều so với việc cấp phát các khung trang. Để dung hòa vấn đề này các hệ điều hành có thể kết hợp cả phân trang và phân đoạn.

### **III.11. Kỹ thuật bộ nhớ ảo (Virtual Memory)**

#### **III.3.1. Bộ nhớ ảo**

*Sau khi tìm hiểu về hai kỹ thuật cấp phát bộ nhớ phân trang đơn và phân đoạn đơn, chúng ta nhận thấy rằng chúng có hai đặc tính nổi bật sau đây:*

- *Tất cả bộ nhớ được tham chiếu trong phạm vi một tiến trình là địa chỉ logic, địa chỉ này được chuyển thành địa chỉ vật lý một cách động tại thời điểm chạy của tiến trình. Điều này có nghĩa một tiến trình có thể được nạp vào một vị trí bất kỳ trên bộ nhớ, hoặc một tiến trình có thể bị swap out ra bộ nhớ ngoài sau đó được swap in vào lại tại một vị trí bất kỳ trên bộ nhớ chính, hoàn toàn không phụ thuộc vào vị trí mà nó được nạp trước khi bị swap out.*
- *Một tiến trình có thể được chia thành nhiều trang/đoạn khác nhau, các trang/đoạn của một tiến trình có thể được nạp vào các vị trí không liên tục nhau trong bộ nhớ trong quá trình thực hiện của tiến trình.*

Mặc dù kỹ thuật phân trang đơn và kỹ thuật phân đoạn đơn khắc phục được những nhược điểm của sự phân vùng cố định và phân vùng động, nhưng nó còn một hạn chế lớn là phải nạp tất cả các trang/đoạn của một tiến trình vào bộ nhớ để tiến trình này hoạt động. Điều này làm cản trở mục tiêu của hệ điều hành là phải nạp được nhiều tiến trình của các chương trình khác nhau vào bộ nhớ để chúng có thể hoạt động đồng thời với nhau, trong thực trạng kích thước của chương trình ngày càng lớn. Ngoài ra việc nạp tất cả các trang/đoạn của tiến trình vào bộ nhớ có thể gây lãng phí bộ nhớ, vì không phải lúc nào tất cả các trang/đoạn này đều cần thiết để tiến trình này có thể hoạt động được.

Để khắc phục hạn chế trên của kỹ thuật phân trang và phân đoạn, kỹ thuật bộ nhớ ảo ra đời. Nguyên lý cơ bản của bộ nhớ ảo là vẫn dựa trên 2 kỹ thuật phân trang và phân đoạn, nhưng trong kỹ thuật bộ nhớ ảo:

- Bộ phận quản lý bộ nhớ không nạp tất cả các trang/đoạn của một tiến trình vào bộ nhớ để nó hoạt động, mà chỉ nạp các trang/đoạn cần thiết tại thời điểm khởi tạo. Sau đó, khi cần bộ phận quản lý bộ nhớ sẽ dựa vào PCT hoặc SCT của mỗi tiến trình để nạp các trang/đoạn tiếp theo.
- Nếu có một trang/đoạn của một tiến trình cần được nạp vào bộ nhớ trong tình trạng trên bộ nhớ không còn khung trang/phân đoạn trống thì bộ phận quản lý bộ nhớ sẽ đưa một trang/đoạn không cần thiết tại thời điểm

hiện tại ra bộ nhớ ngoài (swap-out), để lấy không gian nhớ trống đó nạp trang/đoạn vừa có yêu cầu. Trang/đoạn bị swap out sẽ được đưa vào tại thời điểm thích hợp hoặc cần thiết sau này (swap-in).

Vì vậy hệ điều hành có thể cài đặt bộ nhớ ảo theo 2 kỹ thuật:

- Phân trang theo yêu cầu: Tức là phân trang kết hợp với swap.
- Phân đoạn theo yêu cầu: Tức là phân đoạn kết hợp với swap.

Cả hai kỹ thuật trên đều phải có sự hỗ trợ của phần cứng máy tính, cụ thể là processor. Đa số các hệ điều hành đều chọn kỹ thuật phân trang theo yêu cầu, vì nó đơn giản, dễ cài đặt và chi phí thấp hơn.

Để cài đặt được bộ nhớ ảo hệ điều hành cần phải có:

- Một lượng không gian bộ nhớ phụ (đĩa) cần thiết đủ để chứa các trang/đoạn bị swap out, không gian đĩa này được gọi là không gian swap.
- Có cơ chế để theo dõi các trang/đoạn của một tiến trình, của tất cả các tiến trình đang hoạt động trên bộ nhớ chính, là đang ở trên bộ nhớ chính hay ở trên bộ nhớ phụ. Trong trường hợp này hệ điều hành thường đưa thêm một bit trạng thái (bit present) vào các phần tử trong PCT hoặc SCT.
- Dựa vào các tiêu chuẩn cụ thể để chọn một trang nào đó trong số các trang đang ở trên bộ nhớ chính để swap out trong trường hợp cần thiết. Các hệ điều hành đã đưa ra các thuật toán cụ thể để phục vụ cho mục đích này.

Việc sử dụng bộ nhớ ảo mang lại các lợi ích sau đây:

- Hệ điều hành có thể nạp được nhiều tiến trình hơn vào bộ nhớ, trên bộ nhớ tồn tại các trang/đoạn của nhiều tiến trình khác nhau. Hệ thống khó có thể xả ra trường hợp không đủ bộ nhớ để nạp các tiến trình, vì bộ phận quản lý bộ nhớ không nạp tất cả tiến trình vào bộ nhớ và nếu cần có thể swap out các trang/đoạn của một tiến trình nào đó trên bộ nhớ. Lợi ích của việc nạp nhiều tiến trình vào bộ nhớ chúng ta đã biết trong chương Quản lý Tiến trình.
- Có thể nạp vào bộ nhớ một tiến trình có không gian địa chỉ lớn hơn tất cả không gian địa chỉ của bộ nhớ vật lý. Trong thực tế người lập trình có thể thực hiện việc này mà không cần sự hỗ trợ của hệ điều hành và phần cứng bằng cách thiết kế chương trình theo cấu trúc Overlay, việc làm này là quá khó đối với người lập trình. Với kỹ thuật bộ nhớ ảo người lập trình không cần quan tâm đến kích thước của chương trình và kích thước của bộ nhớ tại thời điểm nạp chương trình, tất cả mọi việc này đều do hệ điều hành và phần cứng thực hiện.

Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình mà không cần nạp tất cả tiến trình vào bộ nhớ. Các trang/đoạn của một tiến trình, đang ở trên bộ nhớ

phụ, mà chưa được nạp vào bộ nhớ chính sẽ được định vị tại một không gian nhớ đặc biệt trên bộ nhớ phụ, có thể gọi không gian nhớ này là bộ nhớ ảo của tiến trình. Với sự hỗ trợ của phần cứng hệ điều hành đã đưa ra các cơ chế thích hợp để nhận biết một trang/đoạn của tiến trình đang thực hiện là đang ở trên bộ nhớ chính hay trên bộ nhớ phụ. Như vậy bộ nhớ ảo đã mở rộng (ảo) được không gian bộ nhớ vật lý của hệ thống, chương trình của người sử dụng chỉ nhìn thấy và làm việc trên không gian địa chỉ ảo, việc chuyển đổi từ địa chỉ ảo sang địa chỉ vật lý thực do bộ phận quản lý bộ nhớ của hệ điều hành và processor thực hiện.

Trước khi tìm hiểu về cơ chế cài đặt bộ nhớ ảo của hệ điều hành chúng hãy nhìn lại sự khác biệt giữa các kỹ thuật phân trang, phân đoạn với các kỹ thuật bộ nhớ ảo, thông qua bảng sau đây:

<b>Phân trang đơn</b>	<b>Phân đoạn đơn</b>	<b>Bộ nhớ ảo</b> (Page + Swap)	<b>Bộ nhớ ảo</b> (Segment + Swap)
Bộ nhớ chính được chia thành các phần nhỏ có kích thước cố định, được gọi là các khung trang.	Bộ nhớ chính không được phân vùng trước.	Bộ nhớ chính được chia thành các phần nhỏ có kích thước cố định, được gọi là các khung trang.	Bộ nhớ chính không được phân vùng trước.
Chương trình của người sử dụng được chia thành các trang bởi trình biên dịch hoặc hệ thống quản lý bộ nhớ.	Các đoạn của chương trình được chỉ ra bởi người lập trình và được gởi đến cho trình biên dịch.	Chương trình của người sử dụng được chia thành các trang bởi trình biên dịch hoặc hệ thống quản lý bộ nhớ.	Các đoạn của chương trình được chỉ ra bởi người lập trình và được gởi đến cho trình biên dịch.
Có thể xảy ra phân mảnh nội vi trong phạm vi các frame. Không xảy ra phân mảnh ngoại vi.	Không xảy ra phân mảnh nội vi, nhưng phân mảnh ngoại vi là có thể.	Có thể xảy ra phân mảnh nội vi trong phạm vi các frame. Không xảy ra phân mảnh ngoại vi.	Không xảy ra phân mảnh nội vi, nhưng phân mảnh ngoại vi là có thể.
Hệ điều hành phải duy trì một bảng trang cho mỗi tiến trình để theo dõi các trang của tiến	Hệ điều hành phải duy trì một bảng đoạn cho mỗi tiến trình để theo dõi các	Hệ điều hành phải duy trì một bảng trang cho mỗi tiến trình để theo dõi các trang của tiến	Hệ điều hành phải duy trì một bảng đoạn cho mỗi tiến trình để theo dõi các đoạn của tiến

trình trên bộ nhớ (được nạp vào các khung trang nào)	đoạn của tiến trình trên bộ nhớ (được nạp vào địa chỉ nào, và độ dài của đoạn)	trình trên bộ nhớ (được nạp vào các khung trang nào)	trình trên bộ nhớ (được nạp vào địa chỉ nào, và độ dài của đoạn)
Hệ điều hành phải duy trì một danh sách để theo dõi các khung trang còn trống trên bộ nhớ chính.	Hệ điều hành phải duy trì một danh sách để theo dõi các phần còn trống trên bộ nhớ chính.	Hệ điều hành phải duy trì một danh sách để theo dõi các khung trang còn trống trên bộ nhớ chính.	Hệ điều hành phải duy trì một danh sách để theo dõi các phần còn trống trên bộ nhớ chính.
Processor sử dụng (page number và offset) để tính địa chỉ tuyệt đối.	Processor sử dụng (segment number và offset) để tính địa chỉ tuyệt đối.	Processor sử dụng (page number và offset) để tính địa chỉ tuyệt đối.	Processor sử dụng (segment number và offset) để tính địa chỉ tuyệt đối.
Tất cả các trang của tiến trình phải được nạp vào bộ nhớ chính để chạy trừ khi khi sử dụng các kỹ thuật Overlay.	Tất cả các đoạn của tiến trình phải được nạp vào bộ nhớ chính để chạy trừ khi khi sử dụng các kỹ thuật Overlay.	Không phải nạp tất cả các trang của tiến trình vào các khung trang trên bộ nhớ chính khi tiến trình chạy. Các trang có thể được đọc khi cần.	Không phải nạp tất cả các đoạn của tiến trình vào các khung trang trên bộ nhớ chính khi tiến trình chạy. Các trang có thể được đọc khi cần.
		Đọc một trang vào bộ nhớ chính có thể cần phải đưa một trang ra đĩa.	Đọc một trang vào bộ nhớ chính có thể cần phải đưa một hoặc đoạn ra đĩa.

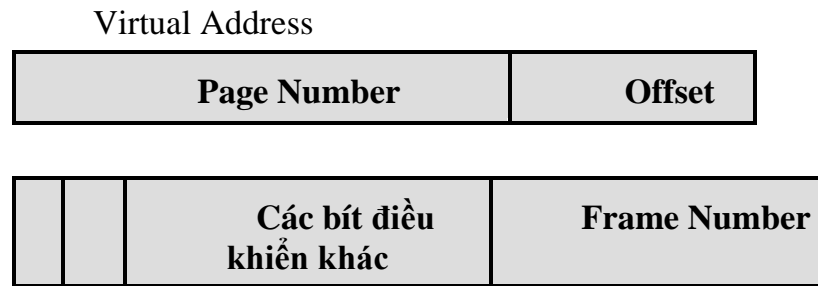
### III.3.2. Kỹ thuật bộ nhớ ảo

*Theo trên thì kỹ thuật bộ nhớ ảo thực chất là kỹ thuật phân trang hoặc phân đoạn theo yêu cầu. Trong mục III.2.3 và III.2.4 chúng ta đã tìm hiểu các vấn đề cơ bản của 2 kỹ thuật phân trang đơn và phân đoạn đơn. Trong mục này chúng ta sẽ tìm hiểu lại kỹ hơn về 2 kỹ thuật này, trong bối cảnh của kỹ thuật bộ nhớ ảo.*

#### III.3.2.a. Sự phân trang:

Trong kỹ thuật phân trang đơn, mỗi tiến trình sở hữu một bảng trang riêng, khi tất cả các trang của tiến trình được nạp vào bộ nhớ chính thì bảng trang của tiến

trình được tạo ra và cũng được nạp vào bộ nhớ (nếu lớn), mỗi phần tử trong bảng trang chỉ chứa số hiệu của khung trang mà trang tương ứng được nạp vào. Trong kỹ thuật bộ nhớ ảo cũng vậy, nhưng một phần tử trong bảng trang sẽ chứa nhiều thông tin phức tạp hơn. Bởi vì trong kỹ thuật bộ nhớ ảo chỉ có một vài page của tiến trình được nạp vào bộ nhớ chính, do đó cần phải có một bit để cho biết một page tương ứng của tiến trình là có hay không trên bộ nhớ chính và một bit cho biết page có bị thay đổi hay không so với lần nạp gần đây nhất. Cụ thể là nó phải có thêm các bit điều khiển:

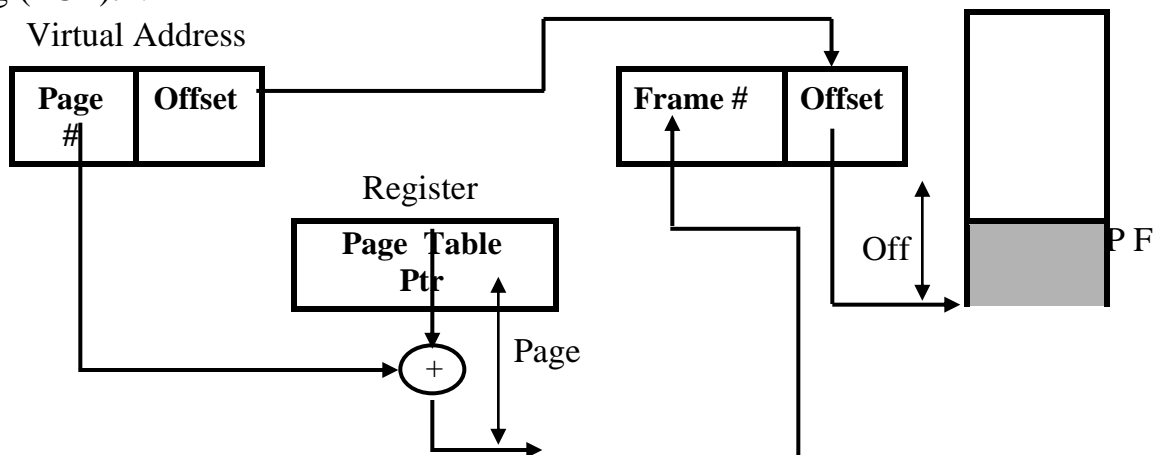


**Hình 3.10a.** Một phần tử trong bảng Trang

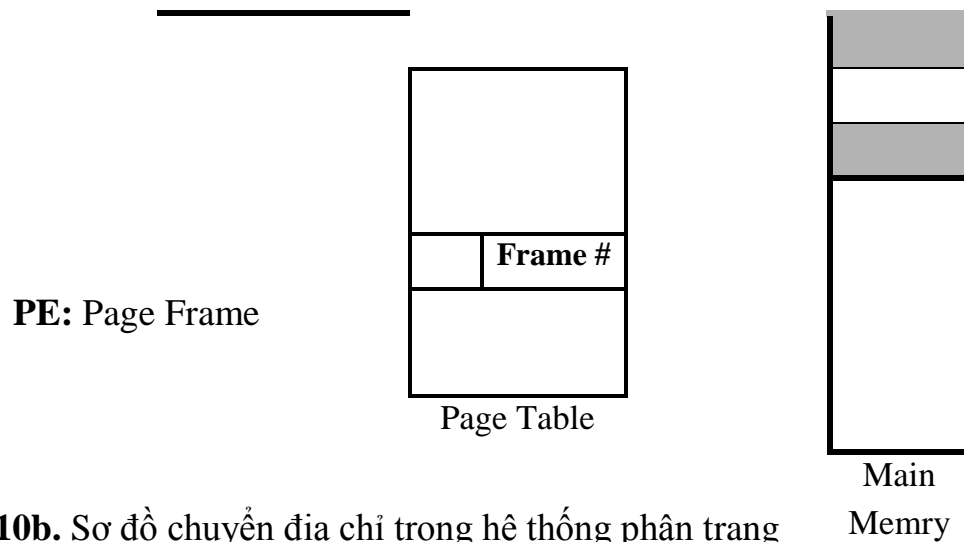
- Bít P (Present): Cho biết trang tương ứng đang ở trên bộ nhớ chính (= 1) hay ở trên bộ nhớ phụ (= 0).
- Bít M (Modify): Cho biết nội dung của trang tương ứng có bị thay đổi hay không so với lần nạp gần đây nhất. Nếu nó không bị thay đổi thì việc phải ghi lại nội dung của một trang khi cần phải đưa một trang ra lại bộ nhớ ngoài là không cần thiết, điều này giúp tăng tốc độ trong các thao tác thay thế trang trong khung trang.
- Các bit điều khiển khác: Các bit này phục vụ cho các mục đích bảo vệ trang và chia sẻ các khung trang.

#### ❑ **Chuyển đổi địa chỉ trong hệ thống phân trang:**

Chương trình của người sử dụng sử dụng địa chỉ logic hoặc virtual gồm: page number và offset để truy xuất dữ liệu trên bộ nhớ chính. Bộ phận quản lý bộ nhớ phải chuyển địa chỉ virtual này thành địa chỉ vật lý tương ứng bao gồm: page number và offset. Để thực hiện việc này bộ phận quản lý bộ nhớ phải dựa vào bảng trang (PCT). Vì





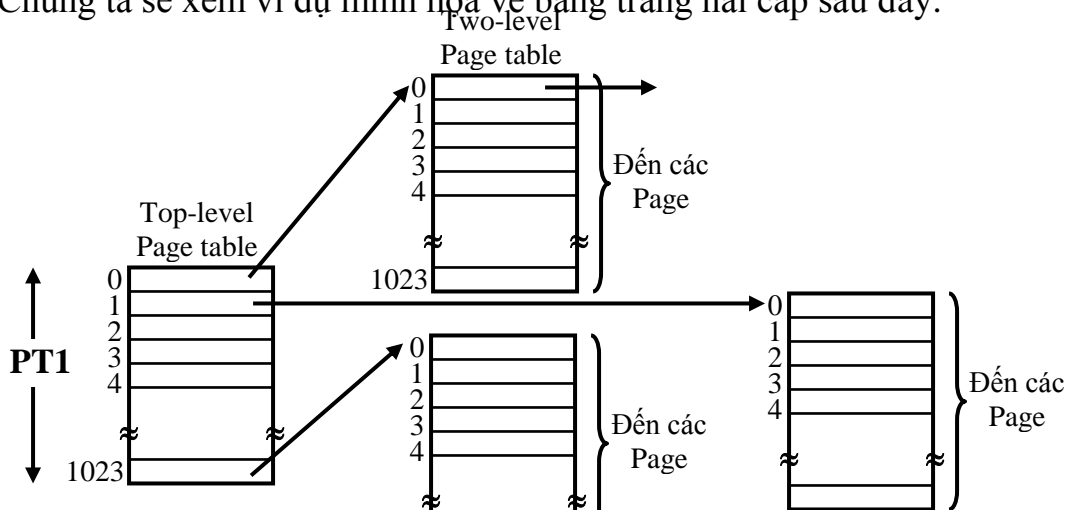


**Hình 3.10b.** Sơ đồ chuyển địa chỉ trong hệ thống phân trang

kích thước của PCT có thể lớn và thay đổi theo kích thước của tiến trình do đó trong kỹ thuật bộ nhớ ảo hệ điều hành thường chứa PCT trong bộ nhớ chính và dùng một thanh ghi để ghi địa chỉ bắt đầu của bộ nhớ nơi lưu trữ PCT của tiến trình khi tiến trình được nạp vào bộ nhớ chính để chạy.

Đa số các hệ điều hành đều thiết kế một bảng trang riêng cho mỗi tiến trình. Nhưng mỗi tiến trình có thể chiếm giữ một không gian lớn bộ nhớ ảo, trong trường hợp này bảng trang rất lớn và hệ thống phải tốn không gian bộ nhớ để chứa nó. Ví dụ, nếu một tiến trình có đến  $2^{31} = 2\text{GB}$  bộ nhớ ảo, mỗi trang có kích thước  $2^9 = 512$  byte, thì tiến trình này phải cần đến  $2^{22}$  phần tử trong bảng trang. Để khắc phục vấn đề này, trong các kỹ thuật bộ nhớ ảo hệ thống lưu trữ bảng trang trong bộ nhớ ảo chứ không lưu trữ trong bộ nhớ thực, và bản thân bảng trang cũng phải được phân trang. Khi tiến trình thực hiện, chỉ có một phần của bản trang được nạp vào bộ nhớ chính, đây là phần chứa các phần tử của các trang đang thực hiện tại thời điểm hiện tại.

Một số processor sử dụng lược đồ hai cấp (two-level) để tổ chức các bảng trang lớn, trong lược đồ này có một thư mục bảng trang (page directory) mà mỗi phần tử trong nó trỏ đến một bảng trang. Trong trường hợp này, nếu chiều dài của thư mục trang là  $X$  và chiều dài tối đa của một bảng trang là  $Y$  thì tiến trình có thể có  $X \times Y$  trang. Chiều dài tối đa của một bảng trang chỉ bằng kích thước của một trang. Chúng ta sẽ xem ví dụ minh họa về bảng trang hai cấp sau đây:



Giả sử có một không gian địa chỉ ảo 32 bit, được chia thành 3 trường: PT1 10 bit, PT2 10 bit và Offset 12 bit. Hình vẽ 3.10.c cho thấy cấu trúc của bảng trang 2 cấp tương ứng với không gian địa chỉ ảo 32 bit. Bên trái là top-level của bảng trang (bảng trang cấp 1), nó gồm có 1024 mục vào (tương ứng với 10 bit của PT1), tức là PT1 của địa chỉ ảo dùng để chỉ mục đến một phần tử trong bảng trang cấp 1. Mỗi mục vào dùng để mô tả 4Mb bộ nhớ, vì toàn bộ 4 GB (32 bit) không gian địa chỉ ảo được chia thành 1024 phần. Entry được chỉ mục trong bảng trang cấp 1 từ PT1 sẽ cho ra địa chỉ hoặc số hiệu khung trang của bản trang thứ hai (second-level). Có 1024 bảng trang cấp 2, đánh số từ 0 đến 1023, bảng trang cấp 2 thứ nhất (0) quản lý không gian nhớ 4Mb từ 0Mb đến 4Mb, bảng trang cấp 2 thứ hai (1) quản lý không gian nhớ 4Mb từ 8Mb,.... Trường PT2 bây giờ được dùng để chỉ mục đến bảng trang cấp 2 để tìm ra số hiệu khung trang của page tương ứng. Giá trị tìm được ở đây sẽ được kết hợp với thành phần Offset để có được địa chỉ vật lý của ô nhớ tương ứng với địa chỉ ảo 32 bit được phát sinh ban đầu.

Chúng ta xem lại ví dụ cụ thể sau đây: Có một địa chỉ ảo 32 bit: 0x00403004, đây là địa chỉ tương ứng với PT1 = 1, PT2 = 3 và Offset = 4. Bộ phận MMU sẽ chuyển địa chỉ này thành địa chỉ vật lý như sau: Đầu tiên MMU dùng PT1 để chỉ mục vào bảng trang cấp 1 và đó là entry 1, tương ứng với không gian đại chỉ từ 4Mb đến 8Mb. Sau đó MMU dùng PT2 để chỉ mục vào bảng trang cấp 2 và đó là entry 3, tương ứng với không gian địa chỉ 12292 đến 16383 trong phạm vi 4Mb. Đây là entry chứa số hiệu khung trang của page chứa địa chỉ ảo 0x00403004. Nếu page này có trong bộ nhớ, thì số hiệu khung trang có được từ bảng trang cấp hai sẽ được kết hợp với thành phần Offset để sinh ra địa chỉ vật lý. Địa chỉ này sẽ được đưa lên a\_bus và gửi đến bộ nhớ.

#### □ **Kích thước của trang:**

Kích thước của một trang do phần cứng quy định, đây là một trong những quyết định quan trọng trong việc thiết kế processor. Nếu kích thước của trang nhỏ thì sự phân mảnh bên trong sẽ nhỏ hơn, vì thế việc sử dụng bộ nhớ chính sẽ được hiệu quả hơn. Nhưng nếu kích thước trang nhỏ thì số lượng trang trên một tiến trình sẽ lớn hơn, bảng trang của tiến trình sẽ lớn, sẽ chiếm nhiều bộ nhớ hơn, và như thế

việc sử dụng bộ nhớ chính sẽ kém hiệu quả hơn. Các vi xử lý họ Intel 486 và họ Motorola 68040 chọn kích thước của một trang là 4096 byte.

Ngoài ra kích thước của trang còn ảnh hưởng đến tỉ lệ xảy ra lỗi trang. Ví dụ: khi kích thước của trang là rất nhỏ thì sẽ có một lượng lớn các trang của tiến trình trên bộ nhớ chính, sau một thời gian thì tất cả các trang của bộ nhớ sẽ chứa các tiến trình được tham chiếu gần đây, vì thế tốc độ xảy ra lỗi trang được giảm xuống.

### **III.3.2.b. Sự phân đoạn:**

Sự phân đoạn cho phép người lập trình xem bộ nhớ như bao gồm một tập các không gian nhớ hoặc các đoạn (segment) có địa chỉ được xác định. Với bộ nhớ ảo người lập trình không cần quan tâm đến giới hạn bộ nhớ được đưa ra bởi bộ nhớ chính. Các segment có thể có kích thước không bằng nhau và được ấn định một cách động. Địa chỉ tham chiếu bộ nhớ trong trường hợp này bao gồm: Segment Number và Offset.

Đối với người lập trình thì sự phân đoạn không gian địa chỉ có một số thuận lợi sau đây so với trường hợp không phân đoạn không gian địa chỉ:

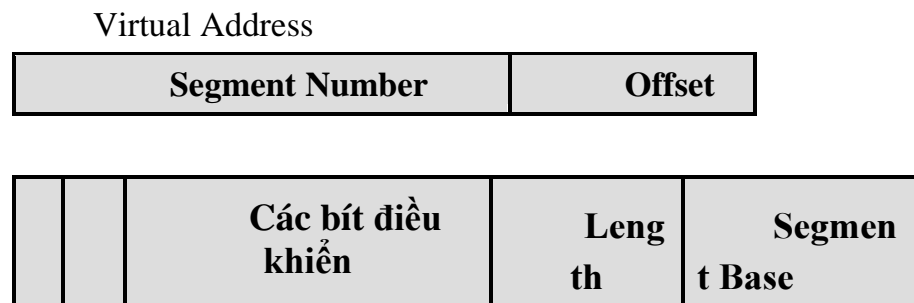
1. Nó đơn giản để điều khiển các cấu trúc dữ liệu lớn dần (growing) trong quá trình hoạt động của hệ thống. Nếu người lập trình không biết trước dữ liệu sẽ lớn đến chừng nào tại thời điểm chạy thì việc ấn định kích thước của động cho segment mang lại nhiều thuận lợi cho người lập trình.
2. Nó cho phép các chương trình không phụ thuộc vào sự thay đổi vào sự biên dịch lại. Nó không yêu cầu thiết lập lại toàn bộ chương trình khi chương trình được liên kết hoặc được nạp trở lại. Việc này chỉ có thể thực hiện bằng cách sử dụng nhiều phân đoạn (Multiple Segment).
3. Nó thích hợp với chiến lược chia sẻ segment giữa các tiến trình. Người lập trình có thể đặt một chương trình tiện ích hoặc một bảng dữ liệu thường sử dụng vào một segment mà có thể được tham chiếu bởi nhiều tiến trình khác nhau.
4. Nó thích hợp với chiến lược bảo vệ bộ nhớ. Bởi vì một segment có thể được sinh ra để chứa một tập xác định các thủ tục hoặc dữ liệu, sau đó người lập trình hoặc người quản trị hệ thống có thể gán quyền truy cập với các độ ưu tiên thích hợp nào đó.

#### **□ Tổ chức của hệ thống phân đoạn:**

Trong kỹ thuật phân đoạn đơn, mỗi tiến trình sở hữu một bảng đoạn riêng, khi tất cả các đoạn của tiến trình được nạp vào bộ nhớ chính thì bảng đoạn của tiến trình được tạo ra và cũng được nạp vào bộ nhớ, mỗi phần tử trong bảng đoạn chứa địa chỉ bắt đầu của đoạn tương ứng trong bộ nhớ chính và độ dài của đoạn. Trong kỹ thuật bộ nhớ ảo cũng vậy, nhưng một phần tử trong bảng đoạn sẽ chứa nhiều thông tin phức tạp hơn. Bởi vì trong kỹ thuật bộ nhớ ảo chỉ có một vài segment của

tiến trình được nạp vào bộ nhớ chính, do đó cần phải có một bit để cho biết một đoạn tương ứng của tiến trình là có hay không trên bộ nhớ chính và một bit cho biết đoạn có bị thay đổi hay không so với lần nạp gần đây nhất. Cụ thể là nó phải có thêm các bit điều khiển:

- Bít M (Modify): Cho biết nội dung của đoạn tương ứng có bị thay đổi hay không so với lần nạp gần đây nhất. Nếu nó không bị thay đổi thì việc phải ghi lại nội dung của một đoạn khi cần phải đưa một đoạn ra lại bộ nhớ ngoài là không cần thiết, điều này giúp tăng tốc độ trong các thao tác thay thế đoạn.

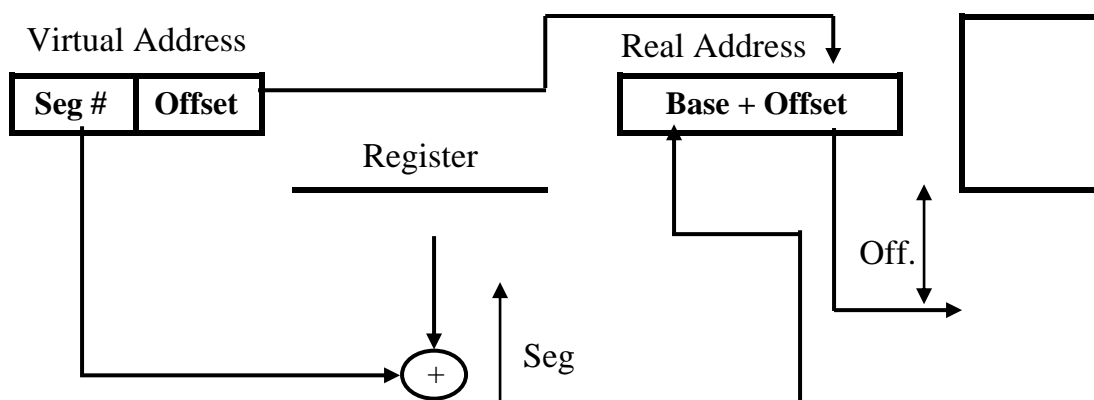


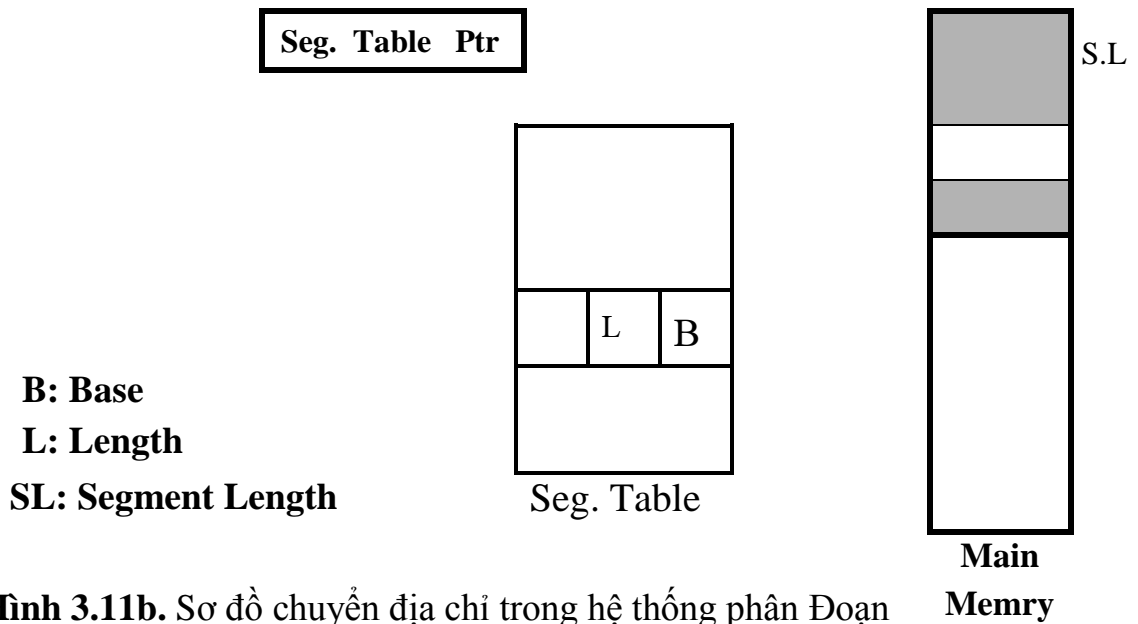
**Hình 3.11a.** Một phần tử trong bảng Đoạn

- Bít P (Present): Cho biết đoạn tương ứng đang ở trên bộ nhớ chính (= 1) hay ở trên bộ nhớ phụ (= 0).
- Các bit điều khiển khác: Các bit này phục vụ cho các mục đích bảo vệ trang và chia sẻ các khung trang.

#### ❑ **Chuyển đổi địa chỉ trong hệ thống phân đoạn:**

Chương trình của người sử dụng sử dụng địa chỉ logic hoặc virtual gồm: segment number và offset để truy xuất dữ liệu trên bộ nhớ chính. Bộ phận quản lý bộ nhớ phải chuyển địa chỉ virtual này thành địa chỉ vật lý tương ứng bao gồm: segment number và offset. Để thực hiện việc này bộ phận quản lý bộ nhớ phải dựa vào bảng đoạn (SCT). Vì kích thước của SCT có thể lớn và thay đổi theo kích thước của tiến trình do đó trong kỹ thuật bộ nhớ ảo hệ điều hành thường chứa SCT trong bộ nhớ chính và dùng một thanh ghi để ghi địa chỉ bắt đầu của bộ nhớ nơi lưu trữ SCT của tiến trình khi tiến trình được nạp vào bộ nhớ chính để chạy. Thành phần segment number của địa chỉ ảo được dùng để chỉ mục đến bảng đoạn và tìm địa chỉ bắt đầu của segment tương ứng trong bộ nhớ chính. Giá trị này sẽ được cộng với thành phần Offset có trong địa chỉ ảo để có được địa chỉ vật lý thực cần tìm.





**Hình 3.11b.** Sơ đồ chuyển địa chỉ trong hệ thống phân Đoạn

#### □ Bảo vệ và chia sẻ trong phân đoạn:

Sự phân đoạn dùng chính nó để cài đặt các chính sách bảo vệ và chia sẻ bộ nhớ. Bởi vì mỗi phần tử trong bảng trang bao gồm một trường length và một trường base address, nên một tiến trình trong segment không thể truy cập đến một vị trí trong bộ nhớ chính mà vị trí này vượt qua giới hạn (length) của segment, ngoại trừ đó là một truy cập dữ liệu đến một segment dữ liệu nào đó.

Để thực hiện việc chia sẻ segment, ví dụ segment A, cho nhiều tiến trình, hệ điều hành cho phép nhiều tiến trình cùng tham chiếu đến segment A, khi đó các thông số (length và base address) của segment A xuất hiện đồng thời ở các bảng segment của các tiến trình cùng chia sẻ segment A. Chiến lược chia sẻ này cũng được áp dụng trong hệ thống phân trang.

Các hệ điều hành cũng có thể sử dụng một chiến lược bảo vệ phức tạp hơn để cài đặt sự bảo vệ các segment, đó là sử dụng cấu trúc vòng bảo vệ (ring protection). Như đã biết trong hệ thống ring, bao gồm: ring 0, ring 1, ring 2, ... thì mỗi ring có một mức đặc quyền truy cập riêng, ring 0 có mức đặc quyền truy cập cao hơn so với ring 1, ring 1 có mức đặc quyền truy cập cao hơn so với ring 2, ..., ring thấp nhất được sử dụng cho thành phần kernel của hệ điều hành, các ring cao hơn được sử dụng cho các ứng dụng của người sử dụng. Nguyên lý cơ bản của hệ thống ring là:

- Chương trình chỉ có thể truy cập đến dữ liệu trong cùng một ring hoặc dữ liệu ở ring có mức đặc quyền truy cập thấp hơn.
- Chương trình có thể gọi các dịch vụ trong cùng một ring hoặc ở các ring có mức đặc quyền truy cập cao hơn.

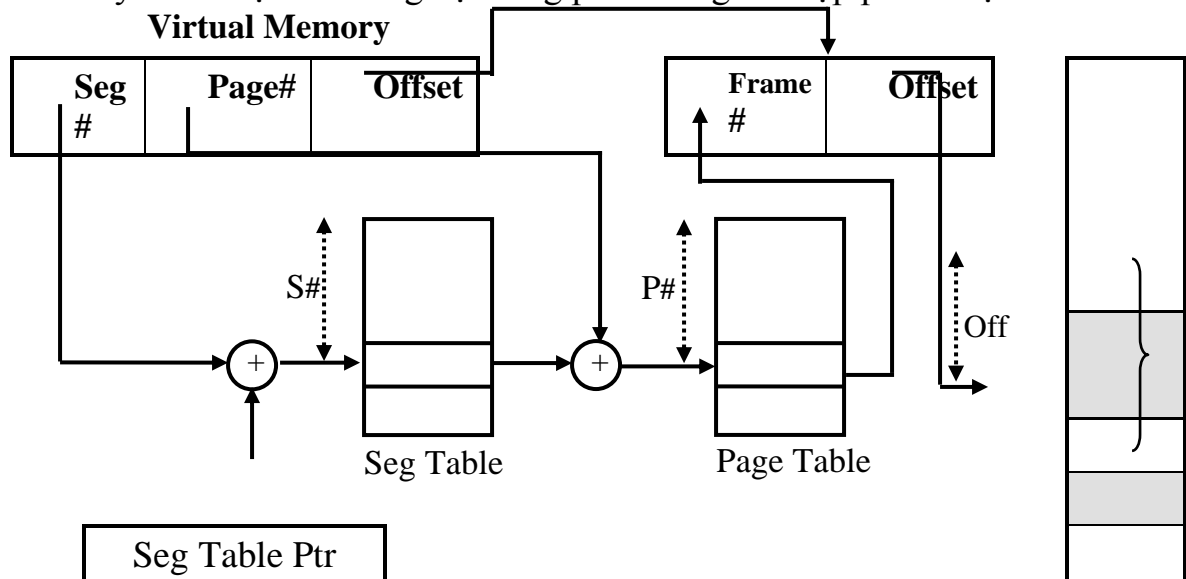
### III.3.2.c. Kết hợp phân trang với phân đoạn:

Cả hai kỹ thuật phân trang và phân đoạn đều có những thế mạnh của nó. Sự phân trang, là trong suốt (transparent) đối với người lập trình, loại bỏ được hiện tượng phân mảnh nội vi. Sự phân đoạn, là thấy được đối với người lập trình, có khả năng điều khiển các cấu trúc dữ liệu lớn dần và hỗ trợ chia sẻ và bảo vệ bộ nhớ. Để kết hợp những thuận lợi của cả hai hệ thống phân trang và phân đoạn, một số hệ thống được trang bị sự hỗ trợ của cả phần cứng processor và phần mềm hệ điều hành để cài đặt kết hợp cả hai kỹ thuật phân trang và phân đoạn.

Trong các hệ thống kết hợp phân trang và phân đoạn, không gian địa chỉ bộ nhớ của người sử dụng được chia thành các đoạn theo ý muốn của người lập trình, sau đó mỗi đoạn lại được chia thành các trang có kích thước cố định bằng nhau. Theo cách nhìn của người lập trình thì địa chỉ logic bao gồm một segment number và một segment offset. Theo cách nhìn của hệ thống thì segment offset được xem như một page number và page offset cho một trang trong phạm vi một segment được chỉ ra.

Trong hệ thống phân trang kết hợp phân đoạn nay, hệ điều hành thiết kế cả bảng trang và bảng đoạn. Hệ điều hành kết hợp với mỗi tiến trình có một bảng đoạn và nhiều bảng trang, mỗi phần tử trong bảng đoạn chỉ đến một bảng trang, bảng trang này quản lý các trang của đoạn tương ứng. Khi một tiến trình riêng biệt chạy, một thanh ghi giữ địa chỉ bắt đầu của bảng đoạn của tiến trình đó. Trong hệ thống này địa chỉ ảo do processor đưa ra phải gồm 3 thành phần: Segment Number, Page Number và Offset. Segment number chỉ vào bảng đoạn tiến trình để tìm bảng trang của segment đó. Sau đó page number được sử dụng để chỉ mục đến bảng trang và tìm số hiệu khung trang tương ứng, giá trị này sẽ được kết hợp với thành phần Offset trong địa chỉ ảo để có được địa chỉ vật lý thực mong muốn.

Sơ đồ chuyển đổi địa chỉ trong hệ thống phân trang kết hợp phân đoạn:



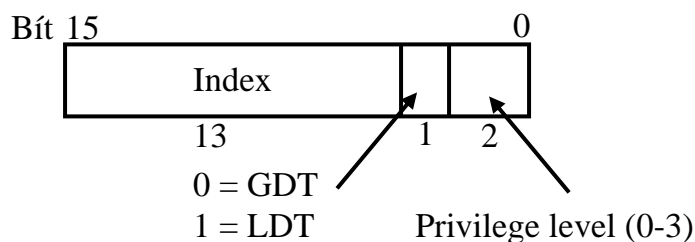
**Hình 3.12.** Sơ đồ chuyển địa chỉ trong hệ thống Trang - Đoạn

- Phân đoạn với phân trang trong Intel 386:

Trong chế độ bảo vệ của 80286 và trong chế độ ảo của 80386 không gian bộ nhớ của hệ thống được chia thành hai loại: không gian bộ nhớ toàn cục và không gian bộ nhớ cục bộ. Không gian nhớ toàn cục được dành cho dữ liệu hệ thống và các tiến trình của hệ điều hành. Mọi chương trình của người sử dụng đều có thể truy cập dữ liệu và các tiến trình ở không gian nhớ toàn cục này. Không gian nhớ cục bộ được dành riêng cho các tiến trình, các tác vụ riêng biệt. Vì vậy, các đoạn mã lệnh và dữ liệu của một tiến trình, một tác vụ nằm trong không gian nhớ cục bộ sẽ được bảo vệ tránh sự truy xuất bất hợp lệ của các tiến trình, các tác vụ khác trong hệ thống.

Trong kỹ thuật bộ nhớ ảo Intel 80386 sử dụng 2 bảng mô tả: Bảng mô tả cục bộ (LDT: Local Descriptor Table), để theo dõi không gian nhớ cục bộ và bảng mô tả toàn cục (GDT: Global Descriptor Table), để theo dõi không gian nhớ toàn cục. Mỗi chương trình sở hữu một LDT riêng, nhưng có một GDT được chia sẻ cho tất cả các chương trình trên hệ thống. LDT mô tả các segment cục bộ cho mỗi chương trình, bao gồm code, data, stack, ..., trong khi đó GDT mô tả các segment hệ thống, của chính hệ điều hành. Các LDT, GDT được nạp vào bộ nhớ trong quá trình hoạt động của hệ thống, Intel 80386 dùng thanh ghi GDTR để ghi địa chỉ cơ sở và giới hạn kích thước của GDT và thanh ghi LDTR để ghi địa chỉ cơ sở và giới hạn kích thước của LDT của tác vụ hiện tại.

Để truy cập một segment, đầu tiên một chương trình chạy trên Intel 386 phải nạp một selector của segment đó vào 1 trong 6 thanh ghi đoạn của Intel 386. Trong quá trình thực hiện chương trình thanh ghi CS giữ selector cho code segment và thanh ghi DS giữ selector cho data segment. Mỗi selector dài 16 bit và được mô tả như sau:



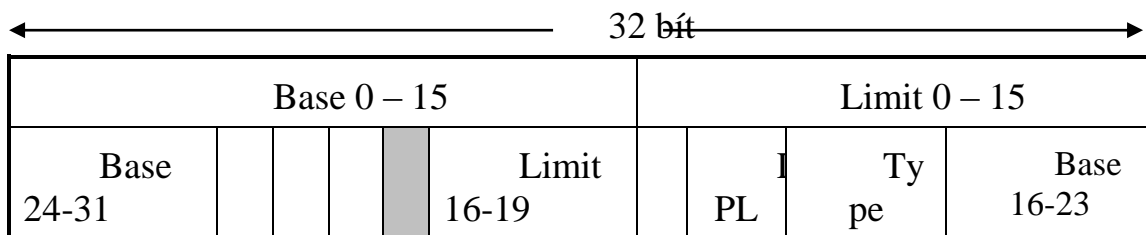
**Hình 3.13.a:** Một Selector (bộ chọn đoạn) Intel 386

Trong đó:

- Hai bit đầu tiên cho biết mức đặc quyền truy cập của bộ chọn đoạn, các bit này phục vụ cho công tác bảo vệ bộ nhớ (segment).

- Một bit tiếp theo cho biết segment là cục bộ hay toàn cục.
- Mười ba bit còn lại chỉ đến mục vào (entry) trong LDT hoặc GDT, vì thế mỗi bảng mô tả (Descriptor Table) chỉ lưu giữ được 8k ( $2^{13}$ ) các bộ mô tả đoạn (segment descriptor). Tức là LDT/GDT có  $2^{13}$  mục vào/ phần tử.

Tại thời điểm một selector được nạp vào một thanh ghi segment, một descriptor tương ứng được nhận từ bảng LDT hoặc GDT và được lưu trữ trong các thanh ghi microprogram, do đó có thể được truy cập nhanh. Một descriptor gồm có 8 byte, gồm có địa chỉ, kích thước, và các thông tin khác của segment. Hình sau đây mô tả một descriptor trong Intel 386:



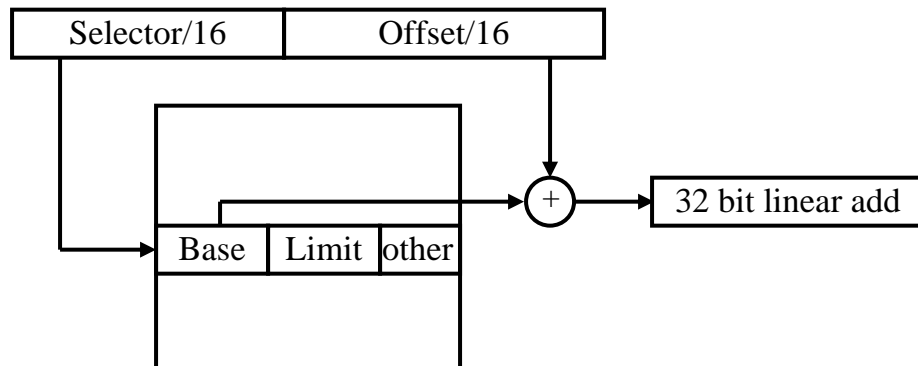
**Hình 3.13.b:** Một descriptor Code segment (bộ mô tả đoạn code) Intel 386

Trong đó:

- Base (24 bit): cho biết vị trí đầu tiên của segment trong không gian địa chỉ tuyến tính 4GB. Bộ xử lý ghép 3 trường địa chỉ cơ sở thành một giá trị địa chỉ 32 bit duy nhất. Trong thực tế trường Base cho phép mỗi segment bắt đầu tại một vị trí bất kỳ trong không gian địa chỉ tuyến tính 32 bit.
- Limit (20 bit): cho biết kích thước của segment. Bộ xử lý ghép hai trường kích thước thành một giá trị 20 bit. Bộ xử lý tính kích thước theo hai cách dựa vào giá trị của cờ G: G = 0: kích thước đoạn nằm giữa 1B và 1MB, tính theo đơn vị byte. G = 1: kích thước đoạn nằm giữa 4KB và 4GB, tính theo đơn vị 4Kbyte ( $= 2^{12} = 1\text{page}$ ). Như vậy với 20 bit limit thì một segment có thể có kích thước lên đến  $2^{32}$  byte ( $2^{12} \times 2^{20}$ ).
- Type (5 bit): định nghĩa dạng của đoạn và kiểu truy cập đoạn.
- DPL: Descriptor Privilege Level (2 bit): cho biết mức đặc quyền truy cập của mô tả segment (có 4 mức đặc quyền truy cập: 0-3).
- P: Present (1 bit): cho biết segment này đã được nạp vào bộ nhớ chính (P = 1) hay chưa được nạp vào bộ nhớ chính (P = 0).
- G: Granularity (1 bit): định nghĩa hằng số để nhân với trường kích thước. G = 0: kích thước tính theo đơn vị 1byte. G = 1: kích thước tính theo đơn vị 1page (Một page của Intel 386 có kích thước cố định là 4Kbyte).
- D: Default Operation Sizze (1 bit): cho biết chiều dài của dòng lệnh. D = 1: vi xử lý mặc định 32 bit địa chỉ, 32/8 bit mã lệnh. D = 0: vi xử lý mặc định 16 bit địa chỉ, 32/8 bit mã lệnh.



Sau đây là sơ đồ chuyển địa chỉ gồm 2 thành phần selector và offset thành địa chỉ tuyến tính (linear address) dựa vào bảng mô tả đoạn.



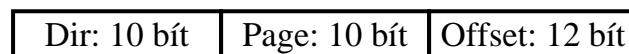
Bảng mô tả đoạn

**Hình 3.13.c:** Chuyển địa chỉ logic (selector:offset) thành địa chỉ tuyến tính

Nếu sự phân trang (paging) bị cấm thì địa chỉ tuyến tính được biên dịch thành địa chỉ vật lý và gửi đến bộ nhớ để truy xuất dữ liệu. Như vậy khi sự phân trang bị cấm thì trong trường hợp này hệ thống chỉ sử dụng sự phân đoạn (segmentation) đơn thuần, với địa chỉ cơ sở (base address) của segment được cho trong descriptor của nó. Nếu sự phân trang là được phép thì địa chỉ tuyến tính sẽ được biên dịch thành địa chỉ ảo và được ánh xạ thành địa chỉ vật lý bằng cách sử dụng các bảng trang.

Mỗi chương trình có một danh mục bảng trang (page directory) riêng, bao gồm 1024 entry 32 bit, nó được nạp vào bộ nhớ được chỉ bởi một thanh ghi global, mỗi entry trong danh mục bảng trang chỉ đến một bảng trang (page table), bảng trang cũng chứa 1024 entry 32 bit, một mục vào trong bảng trang lại chỉ đến một khung trang (page frame).

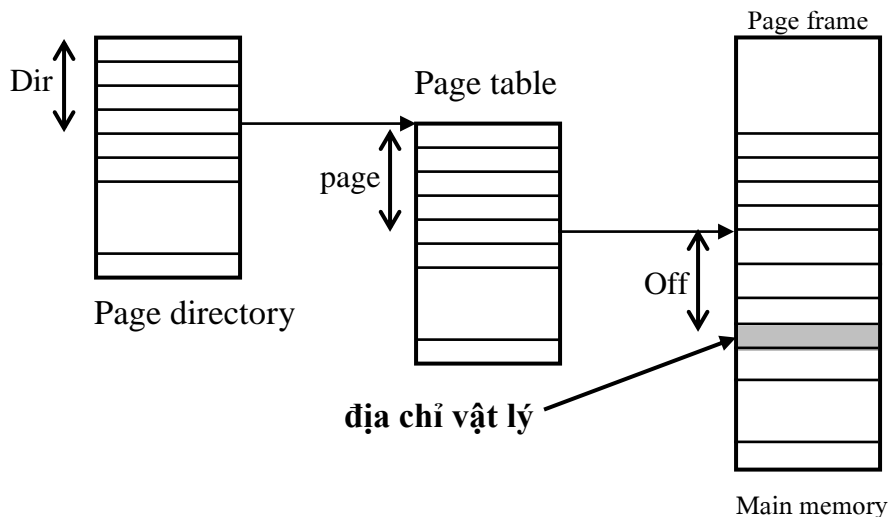
Địa chỉ tuyến tính gồm 3 trường: Dir, Page, Offset. Trường Dir: được sử dụng để chỉ mục vào Page Directory để tìm đến một con trỏ tới Page Table. Trường Page: được sử dụng để chỉ mục vào Page Table để tìm địa chỉ vật lý của Page Frame. Trường Offset được cộng với địa chỉ vật lý của Page Frame để có được địa chỉ vật lý của ô nhớ chứa dữ liệu cần truy xuất.



**Hình 3.13.d :** Địa chỉ tuyến tính 32 bit trong Intel 386

Mỗi entry trong page table dài 32 bit, 20 bit chứa số hiệu của page frame, các bit còn lại là các bit truy cập, được thiết lập bởi phần cứng cho các lợi ích của hệ điều hành các bit bảo vệ và các bit tiện ích khác. Mỗi page table có 1024 entry cho các page frame, mỗi page frame có kích thước là 4Kb, nên một page table đơn quản lý được 4Mb bộ nhớ.

Từ địa chỉ tuyến tính ở trên hệ thống sẽ ánh xạ thành địa chỉ vật lý, dựa vào page directory, page table và page frame. Sau đây là sơ đồ, đơn giản, minh họa sự ánh xạ địa chỉ tuyến tính thành địa chỉ vật lý:



**Hình 3.13.e:** Ánh xạ địa chỉ tuyến tính thành địa chỉ vật lý

Trên đây chúng ta đã tìm hiểu về cơ chế bộ nhớ ảo trong Intel 386, bây giờ chúng ta sẽ tìm hiểu về sự bảo vệ trong cơ chế bộ nhớ ảo của nó.

Công cụ mà 80386 đưa ra để thực hiện nhiệm vụ bảo vệ không gian nhớ chứa các tiến trình và chứa chính hệ điều hành trên bộ nhớ chính là: các mức/ cấp (level) đặc quyền truy cập hay mức ưu tiên được yêu cầu (RPL: Request Privilege Level). Từ vi xử lý 80286 các vi xử lý đã đưa ra 4 mức ưu tiên từ 0 đến 3, *được ghi tại trường Privilege Level của thành phần địa chỉ selector (hình 3.13.a)*. Mức 0 có độ ưu tiên cao nhất, mức 3 có độ ưu tiên thấp nhất. Các segment trên bộ nhớ cũng được gán một mức ưu tiên tương tự, *được ghi tại trường DPL trong bộ mô tả đoạn trong bảng mô tả đoạn (hình 3.13.b)*.

Các ứng dụng có mức ưu tiên cao hơn sẽ được quyền truy cập mã lệnh, dữ liệu tại các đoạn nhớ có mức ưu tiên thấp hơn. Các ứng dụng có mức ưu tiên thấp hơn sẽ không được truy cập mã lệnh, dữ liệu tại các đoạn nhớ có mức ưu tiên cao hơn, trong thực tế thì điều này cũng có thể nếu các ứng dụng biết cách vượt qua các Cổng (Công và nguyên tắc hoạt động của Cổng các bạn có thể tìm đọc ở một tài liệu nào đó viết về các vi xử lý của họ Intel). Bốn mức ưu tiên mà 80386 đưa ra là:

- Mức 0: là mức của thành phần kernel của hệ điều hành. Kernel của hệ điều hành được nạp tại segment có mức đặc quyền truy cập là 0.
- Mức 1: là mức của phần mềm hệ thống quản lý thiết bị và cổng phần

cứng. Segment nhớ được gán mức này chứa các chương trình hệ thống của BIOS và DOS/ Windows.

- Mức 2: chứa các thủ tục thư viện, có thể chia sẻ cho nhiều chương trình đang chạy. Chương trình của người sử dụng có thể gọi các các thủ tục và đọc dữ liệu ở mức này nhưng không thể modify nó.
- Mức 3: chương trình của người sử dụng chạy tại mức này, đây là mức có độ ưu tiên thấp nhất.

Khi chương trình cần truy xuất vào một đoạn nhớ nào đó trên bộ nhớ thì vi xử lý sẽ dựa vào giá trị mức ưu tiên tại RPL và DPL để quyết định có cho phép chương trình truy xuất vào đoạn nhớ hay không. Trong trường hợp này Intel 80386 công nhận ba mức ưu tiên sau đây:

- Mức ưu tiên hiện tại CPL (Current Privilege Level): là mức ưu tiên của chương trình hay tác vụ đang chạy. CPL được lưu trữ tại bit 0 và bit 1 của các thanh ghi đoạn CS và SS. Thông thường giá trị của CPL bằng giá trị mức ưu tiên của đoạn mã lệnh chứa chương trình đang chạy. Giá trị của CPL có thể bị thay đổi nếu điều khiển chương trình được chuyển đến một đoạn mã lệnh có độ ưu tiên khác.
- Mức ưu tiên bộ mô tả DPL (Descriptor Privilege Level): là mức ưu tiên của một đoạn. DPL được lưu trữ tại trường DPL của các bộ mô tả đoạn. Nếu chương trình đang chạy tìm cách truy cập một đoạn nhớ, vi xử lý sẽ so sánh DPL với CPL và RPL của bộ chọn đoạn.
- Mức ưu tiên được yêu cầu RPL (Pequest Privilege Level): là giá trị được ghi tại bit 0 và bit 1 trong bộ chọn đoạn. Vi xử lý sẽ so sánh RPL với CPL để kiểm tra quyền truy cập vào một đoạn. Khi CPL có giá trị cho phép chương trình truy cập một đoạn, nhưng giá trị trong RPL không có mức ưu tiên tương ứng thì chương trình cũng không được phép truy cập đoạn. Điều này có nghĩa là nếu RPL của bộ chọn đoạn có giá trị lớn hơn CPL, thì RPL sẽ ghi chồng lên CPL.

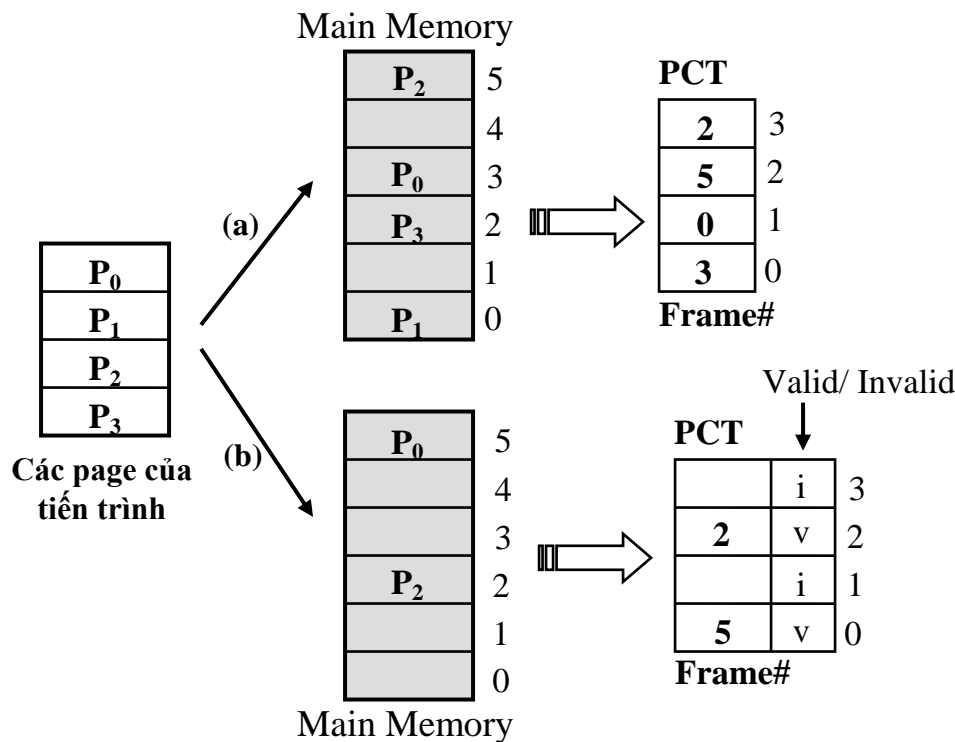
Khi một chương trình có yêu cầu truy cập vào một đoạn nhớ nào đó, thì bộ phận quản lý bộ nhớ của vi xử lý sẽ so sánh mức ưu tiên ghi trong RPL với mức ưu tiên của đoạn được ghi trong DPL, nếu RPL lớn hơn hoặc bằng DPL thì vi xử lý cho phép chương trình truy cập vùng nhớ mà nó yêu cầu, nếu ngược lại thì vi xử lý không trao quyền truy cập vùng nhớ cho chương trình, đồng thời phát ra một ngắt thông báo có sự vi phạm quyền truy cập vùng nhớ. Đây chính là cơ chế bảo vệ bộ nhớ của Intel 80386.

#### **III.3.2.d. Bộ nhớ ảo và lỗi trang (page fault)**

Trước khi tiếp tục tìm hiểu về sự cài đặt bộ nhớ ảo và hiện tượng lỗi trang trong bộ nhớ ảo, chúng ta xem lại sơ đồ sau, hình 3.14, để thấy được sự khác nhau giữa kỹ

thuật phân trang đơn và kỹ thuật bộ nhớ ảo theo kiểu phân trang.

Sơ đồ này cho thấy: có một tiến trình gồm có 4 trang  $P_0, P_1, P_2, P_3$  cần được nạp vào bộ nhớ chính, không gian địa chỉ bộ nhớ chính gồm có 6 phân trang (paging), còn được gọi là khung trang (frame), còn trống:



Hình 3.14: Sự khác nhau giữa phân trang đơn và bộ nhớ ảo phân trang. a: nạp tất cả các page của tiến trình vào bộ nhớ. b: chỉ nạp page 0 và page 2 vào bộ nhớ.

- Trường hợp a, là trường hợp phân trang đơn: trong trường hợp này tất cả 4 page của tiến trình đều được nạp vào bộ nhớ. Rõ ràng sẽ là lãng phí bộ nhớ nếu biết rằng tiến trình này chỉ cần nạp vào bộ nhớ 2 trang  $P_0, P_2$  là tiến trình có thể khởi tạo và bắt đầu hoạt động được. Và trong trường hợp này nếu bộ nhớ chỉ còn 3 frame còn trống thì tiến trình cũng sẽ không nạp vào bộ nhớ được. PCT trong trường hợp này cho biết các page của tiến trình được nạp vào các frame trên bộ nhớ chính.

- Trường hợp b, là trường hợp bộ nhớ ảo sử dụng kỹ thuật phân trang: trong trường hợp này hệ điều hành không nạp tất cả các page của tiến trình vào bộ nhớ mà chỉ nạp 2 page cần thiết ban đầu để tiến trình có thể khởi tạo và bắt đầu hoạt động được, mặc dầu trên bộ nhớ chính còn một vài frame còn trống. Rõ ràng trong trường hợp này hệ điều hành đã tiết kiệm được không gian bộ nhớ chính và

nhớ đó mà hệ điều hành có thể nạp vào bộ nhớ nhiều tiến trình hơn và cho phép các tiến trình này hoạt động đồng thời với nhau. Các page của tiến trình chưa được nạp vào bộ nhớ sẽ được lưu trữ tại một không gian đặc biệt trên đĩa (thường là trên HDD), không gian đĩa này được gọi là không gian bộ nhớ ảo, một cách chính xác thì không gian bộ nhớ ảo này chứa tất cả các page của một tiến trình. Như vậy PCT trong trường hợp này phải có thêm một trường mới, trường này thường được gọi là trường Present. Trường Present chỉ cần một bit, nó cho biết page tương ứng là đã được nạp vào bộ nhớ chính (= 1), hay còn nằm trên đĩa (= 0).

Trong mô hình bộ nhớ ảo khi cần truy xuất đến một page của tiến trình thì trước hết hệ thống phải kiểm tra bit present tại phần tử tương ứng với page cần truy xuất trong PCT, để biết được page cần truy xuất đã được nạp vào bộ nhớ hay chưa. Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó đã được nạp vào bộ nhớ chính, được gọi là truy xuất hợp lệ (v: valid). Trường hợp hệ thống cần truy xuất đến một page của tiến trình mà page đó chưa được nạp vào bộ nhớ chính, được gọi là truy xuất bất hợp lệ (i: invalid). Khi hệ thống truy xuất đến một trang của tiến trình mà trang đó không thuộc phạm vi không gian địa chỉ của tiến trình cũng được gọi là truy xuất bất hợp lệ.

Khi hệ thống truy xuất đến một page được đánh dấu là bất hợp lệ thì sẽ phát sinh một lỗi trang. Như vậy lỗi trang là hiện tượng hệ thống cần truy xuất đến một page của tiến trình mà trang này chưa được nạp vào bộ nhớ, hay không thuộc không gian địa chỉ của tiến trình. Ở đây ta chỉ xét lỗi trang của trường hợp: Page cần truy xuất chưa được nạp vào bộ nhớ chính.

Khi nhận được tín hiệu lỗi trang, hệ điều hành phải tạm dừng tiến trình hiện tại để tiến hành việc xử lý lỗi trang. Khi xử lý lỗi trang hệ điều hành có thể gặp một trong hai tình huống sau:

- Hệ thống còn frame trống (a): Hệ điều hành sẽ thực hiện các bước sau:
  1. Tìm vị trí của page cần truy xuất trên đĩa.
  2. Nạp page vừa tìm thấy vào bộ nhớ chính.
  3. Cập nhật lại bảng trang (PCT) tiến trình.
  4. Tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động.
- Hệ thống không còn frame trống (b):
  1. Tìm vị trí của page cần truy xuất trên đĩa.
  2. Tìm một page không hoạt động hoặc không thực sự cần thiết tại thời điểm hiện tại để swap out nó ra đĩa, lấy frame trống đó để nạp page mà hệ thống vừa cần truy xuất. Page bị swap out sẽ được hệ điều hành swap in trở lại bộ nhớ tại một thời điểm thích hợp sau này.
  3. Cập nhật PCT của tiến trình có page vừa bị swap out.

4. Nạp trang vừa tìm thấy ở trên (bước 1) vào frame trống ở trên (bước 2).
5. Cập nhật lại bảng trang (PCT) của tiến trình.
6. Tái kích hoạt tiến trình để tiến trình tiếp tục hoạt động.

Xử lý lỗi trang là một trong những nhiệm vụ quan trọng và phức tạp của hệ thống và hệ điều hành. Để xử lý lỗi trang hệ thống phải tạm dừng các thao tác hiện tại, trong trường hợp này hệ thống phải lưu lại các thông tin cần thiết như: con trỏ lệnh, nội dung của các thanh ghi, các không gian địa chỉ bộ nhớ, ..., các thông tin này là cơ sở để hệ thống tái kích hoạt tiến trình bị tạm dừng trước đó khi nó đã hoàn thành việc xử lý lỗi trang.

➤ Khi xử lý lỗi trang, trong trường hợp hệ thống không còn frame trống hệ điều hành phải chú ý đến các vấn đề sau:

- **Nên chọn page nào trong số các page trên bộ nhớ chính để swap out:**  
Về vấn đề này chúng ta đã biết hệ điều hành sẽ áp dụng một thuật toán thay page cụ thể nào đó, nhưng ở đây cần chú ý thêm rằng đối tượng của các thuật toán thay page là chỉ các page của tiến trình xảy ra lỗi page, hay tất cả các page của các tiến trình đang có trên bộ nhớ chính. Tức là, nên chọn page của tiến trình xảy ra lỗi trang để thay thế (*thay thế cục bộ*), hay chọn một page của tiến trình khác để thay thế (*thay thế toàn cục*). Nếu chọn page của tiến trình xảy ra lỗi trang thì sẽ đơn giản hơn với hệ điều hành và không ảnh hưởng đến các tiến trình khác, nhưng cách này có thể làm cho tiến trình hiện tại lại tiếp tục xảy ra lỗi trang ngay sau khi hệ điều hành vừa xử lý lỗi trang cho nó, vì page mà hệ điều hành vừa chọn để đưa ra (swap out) lại là page cần truy xuất ở thời điểm tiếp theo. Nếu chọn page của tiến trình khác thì tiến trình hiện tại sẽ ít có nguy cơ xảy ra lỗi trang ngay sau đó hơn, nhưng cách này sẽ phức tạp hơn cho hệ điều hành, vì hệ điều hành phải kiểm soát lỗi trang của nhiều tiến trình khác trong hệ thống, và hệ điều hành khó có thể dự đoán được nguy cơ xảy ra lỗi trang của các tiến trình trong hệ thống. Trong trường hợp này có thể lỗi trang sẽ lan truyền đến nhiều tiến trình khác trong hệ thống, khi đó việc xử lý lỗi trang của hệ điều hành sẽ phức tạp hơn rất nhiều. Đa số các hệ điều hành đều chọn cách thứ nhất vì nó đơn giản và không ảnh hưởng đến các tiến trình khác trong hệ thống.

- **“Neo” một số page:** Trên bộ nhớ chính tồn tại các page của các tiến trình đặc biệt quan trọng đối với hệ thống, nếu các tiến trình này bị tạm dừng thì sẽ ảnh hưởng rất lớn đến hệ thống và có thể làm cho hệ thống ngừng hoạt động, nên hệ điều hành không được đưa các page này ra đĩa trong bất kỳ trường hợp nào. Để tránh các thuật toán thay trang chọn các page này hệ điều hành tổ chức đánh dấu các page này, bằng cách đưa thêm một bit mới vào các phần tử trong các PCT, bit này được gọi là bit neo. Như vậy các thuật toán thay trang sẽ không xem xét đến các page được đánh dấu neo khi cần phải đưa một trang nào đó ra đĩa.

- **Phải tránh được trường hợp hệ thống xảy ra hiện tượng “trì trệ hệ thống”:** Trì trệ hệ thống là hiện tượng mà hệ thống luôn ở trong tình trạng xử lý lỗi trang, tức là đa phần thời gian xử lý của processor đều dành cho việc xử lý lỗi trang của hệ điều hành. Hiện tượng này có thể được mô tả như sau: khi xử lý lỗi trang trong trường hợp trên bộ nhớ chính không còn frame trống, trong trường hợp này hệ điều hành phải chọn một page nào đó, ví dụ  $P_3$ , để swap out nó, để lấy frame trống đó, để nạp page vừa có yêu cầu nạp, để khắc phục lỗi trang. Nhưng khi vừa khắc phục lỗi trang này thì hệ thống lại xảy ra lỗi trang mới do hệ thống cần truy xuất dữ liệu ở trang  $P_3$ , hệ điều hành lại phải khắc phục lỗi trang này, và hệ điều hành phải swap out một page nào đó, ví dụ  $P_5$ . Nhưng ngay sau đó hệ thống lại xảy ra lỗi trang mới do không tìm thấy page  $P_5$  trên bộ nhớ chính và hệ điều hành lại phải xử lý lỗi trang, và cứ như thế có thể hệ điều hành phải kéo dài việc xử lý lỗi trang mà không thể kết thúc được. Trong trường hợp này ta nói rằng: hệ thống đã rơi vào tình trạng “trì trệ hệ thống”. Như vậy hệ thống có thể xảy ra hiện tượng “trì trệ hệ thống” khi: trên bộ nhớ không còn frame trống, page mà thuật toán thay trang chọn để swap out là một page không được “tốt”, xét về khía cạnh dự báo lỗi trang của hệ điều hành.

- **Đánh dấu các trang bị thay đổi:** Khi xử lý lỗi trang, ta thấy hệ điều hành thường phải thực hiện thao tác swap out. Hệ điều hành phải mang một page của một tiến trình tại một khung trang nào đó ra lưu tạm trên đĩa cứng, tại không gian swap. Tức là, hệ điều hành phải tốn thời gian cho thao tác swap out, điều này sẽ làm giảm tốc độ của hệ thống và có thể gây lãng phí thời gian xử lý của processor. Hệ điều hành có thể hạn chế được điều này bằng cách: *không phải lúc nào hệ điều hành cũng phải thực hiện swap out một page để lấy khung trang trống mà hệ điều hành chỉ thực sự swap out một page khi page đó đã bị thay đổi kể từ lần nó được nạp vào bộ nhớ gần đây nhất*. Khi đã quyết định swap out một page để lấy khung trang trống để nạp một page mới vào bộ nhớ, mà page cần swap này không bị thay đổi kể từ lần nạp gần đây nhất, hệ điều hành sẽ không swap out nó mà hệ điều hành chỉ nạp page mới vào bộ nhớ và ghi đè lên nó, điều này có nghĩa là hệ điều hành đã tiết kiệm được thời gian swap out một page tiến trình ra đĩa. Để làm được điều này hệ điều hành phải giải quyết hai vấn đề sau: Thứ nhất, làm thế nào để xác định được một page là đã bị thay đổi hay chưa kể từ lần nạp vào bộ nhớ gần đây nhất. Thứ hai, nếu không swap out một page thì khi cần hệ điều hành sẽ swap in nó từ đâu.

Đối với vấn đề thứ nhất: hệ điều hành chỉ cần thêm một bit, bit modify chẳng hạn, vào phần tử trong bảng trang. Khi một page vừa được nạp vào bộ nhớ thì bit modify bằng 0, nếu sau đó nội dung của page bị thay đổi thì bit modify được đổi thành 1. Hệ điều hành sẽ dựa vào bit modify này để biết được một page có bị thay đổi hay không kể từ lần nạp vào bộ nhớ gần đây nhất.

Đối với vấn đề thứ hai: hệ điều hành có thể swap in một page tại vị trí ban đầu của nó trên đĩa, hoặc tại không gian swap của nó. Trong một số hệ điều hành khi một tiến trình được tạo thì lập tức hệ điều hành sẽ cấp cho nó một không gian swap trên đĩa, bất kỳ khi nào tiến trình bị swap out nó đều được swap đến không gian swap của nó, khi tiến trình kết thúc thì không gian swap của nó sẽ được giải phóng. Như vậy để chuẩn bị cho việc swap in sau này, khi nạp một page của tiến trình vào bộ nhớ hệ điều hành sẽ ghi nội dung của page này vào không gian swap của nó.

### **III.12. Quản lý bộ nhớ RAM của DOS**

#### **III.5.a. Program Segment Prefix (PSP):**

MS\_DOS dùng hàm EXEC để nạp các chương trình EXE và COM vào bộ nhớ. EXEC phải chuẩn bị một vùng nhớ mà chương trình sẽ nạp vào, và đặt vào đầu vùng nhớ này một cấu trúc dữ liệu được gọi là khối tiền tố chương trình PSP. Chương trình sẽ được nạp vào ngay sau PSP, sau đó hệ điều hành sẽ khởi tạo các thành phần Segment và Stack để chương trình bắt đầu hoạt động. Khi chương trình kết thúc thì hệ điều hành giải phóng cả khối nhớ cấp cho chương trình và khối nhớ dùng làm PSP cho chương trình.

PSP dài 256 byte, chứa các thông tin cần cho cả DOS và chương trình chạy. PSP cho biết địa chỉ đoạn của ô nhớ cuối cùng dành cho chương trình, địa chỉ đoạn của khối biến môi trường, độ dài của dòng lệnh, các tham số dòng lệnh, vv.

*Các bạn có thể tìm hiểu rõ hơn về cấu trúc của một PSP ở một tài liệu khác viết về hệ điều hành MS\_DOS.*

#### **III.5.b. Chương trình COM và EXE:**

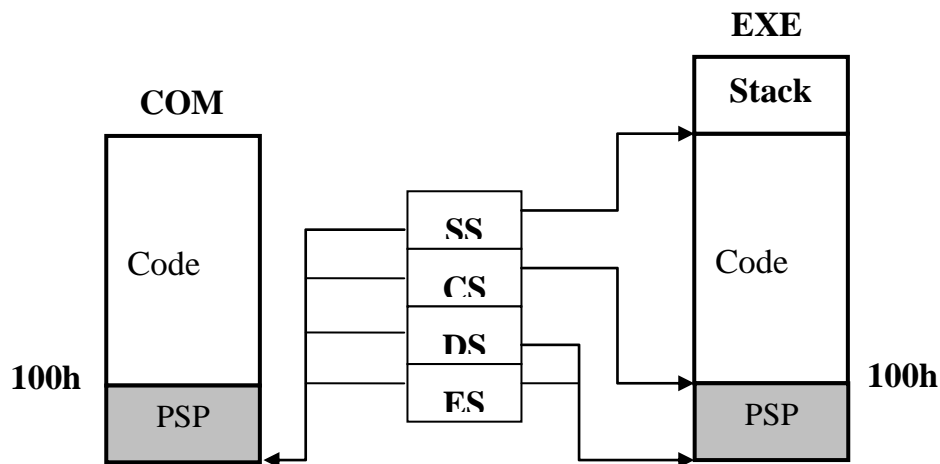
Trong phần này chúng ta xem xét đến sự hoạt động của hai loại file chương trình EXE và COM. Các chương trình dạng COM có kích thước giới hạn trong 64 KB, trong khi đó kích thước của chương trình dạng EXE có thể lớn ngang bằng kích thước của RAM. Hệ điều hành có thể nạp một chương trình dạng EXE có kích thước lớn hơn vào RAM nếu chương trình được thiết kế với cấu trúc thích hợp (cấu trúc động chẳng hạn).

Đối với các chương trình dạng COM, cả ba đoạn của chương trình là code, data và stack đều chứa trong cùng một đoạn 64 KB. Chương trình COM trên RAM là hình ảnh chính xác của nó trên đĩa. Do đó ngay sau khi chương trình COM được nạp vào bộ nhớ RAM thì tất cả các thanh ghi đoạn đều chứa cùng một giá trị, đó chính là địa chỉ đoạn của một đoạn nhớ (64 KB) chứa các đoạn của chương trình, trong quá trình hoạt động của chương trình giá trị các thanh ghi đoạn không thay đổi, ngoại trừ thanh ghi đoạn ES có thể thay đổi. Đối với các chương trình dạng EXE ba đoạn của chương trình là code, data và stack có thể được nạp vào ba đoạn bộ nhớ khác nhau, và có thể một đoạn của chương trình, tùy theo kích thước của



nó, mà nó có thể được nạp vào nhiều hơn một đoạn bộ nhớ. Do đó ngay sau khi chương trình được nạp vào bộ nhớ các thanh ghi đoạn đều được khởi tạo bằng các giá trị khác nhau (có thể  $DS = ES$ ), đó chính là địa chỉ đoạn của các đoạn bộ nhớ chứa các đoạn chương trình tương ứng. Trong quá trình hoạt động của chương trình có thể giá trị của thanh ghi CS bị thay đổi. Chương trình dạng EXE trên bộ nhớ RAM không phải là hình ảnh trung thực của nó ở trên đĩa mà nó được mã hoá theo một cách nào đó. Hình 3.15 sau đây minh họa cho điều này.

Một chương trình dạng COM không thể vượt quá giới hạn 64 Kb kể cả 256 byte PSP và 1 word khởi tạo stack. Mặc dù vậy nhưng khi nạp một chương trình COM vào bộ nhớ DOS vẫn dành hết bộ nhớ còn lại cho chương trình, điều này gây lãng phí bộ nhớ, do đó chương trình COM không thể gọi một chương trình khác thông qua hàm EXEC, nếu trước đó chương trình không giải phóng khối nhớ thừa mà hệ điều hành đã cấp cho chương trình. Vì chương trình COM được nạp ngay sau PSP của nó trong bộ nhớ nên ngay sau khi điều khiển được chuyển cho chương trình COM thì các thanh ghi đoạn đều hướng tới đầu PSP, thanh ghi con trỏ lệnh (IP) nhận giá trị offset 100h và thanh ghi con trỏ stack (SP) nhận giá trị offset FFFEh (vì stack ở vùng cao của đoạn 64 Kb và luôn được khởi tạo bởi một phần tử 2 byte). DOS không nạp chương trình COM vào một địa chỉ xác định trước và không cho phép thực hiện một lời gọi xa (FAR) trong chương trình.



**Hình 3.15:** Các thanh ghi con trỏ đoạn ngay sau khi khởi tạo chương trình: chương trình EXE so với chương trình COM.

Chương trình dạng EXE không bị giới hạn kích thước một segment (64 Kb) mà nó có thể vượt quá giới hạn của 3 segment, nhờ đó mà người lập trình có thể thiết kế được các chương trình với nhiều chức năng hơn. Tuy nhiên điều này làm cho cấu trúc của chương trình sẽ phức tạp hơn, nó phải chứa nhiều thông tin liên quan đến chương trình hơn, các thông tin này là cần thiết để hệ điều hành nạp chương trình vào bộ nhớ và điều khiển việc thực hiện của chương trình. Các

chương trình EXE không được nạp vào các đoạn bộ nhớ xác định trước mà có thể được nạp vào một vị trí bất kỳ (có địa chỉ là bội nguyên của 16). Trong chương trình dạng EXE có thể gọi một chương trình khác thông qua hàm EXEC và có thể thực hiện các lời gọi xa để gọi một chương trình con ở đoạn nhớ khác.

Trong giai đoạn biên dịch và liên kết chương trình, chương trình liên kết LINK đặt vào đầu các file chương trình EXE một cấu trúc dữ liệu đặc biệt được gọi là Header của chương trình. Header chứa nhiều thông tin liên quan đến chương trình như kích thước header, kích thước chương trình, giá trị khởi tạo của IP và SP, địa chỉ bắt đầu của đoạn code trong file EXE, ... và đặc biệt là *địa chỉ tương đối của các đoạn*. Địa chỉ đoạn thực tế trong bộ nhớ RAM nhận được bằng cách cộng *địa chỉ tương đối* này với địa chỉ của đoạn mà chương trình được nạp vào (địa chỉ đoạn bắt đầu). Địa chỉ đoạn bắt đầu thường là địa chỉ đoạn của PSP cộng với 10h.

Khi hệ điều hành nạp một chương trình EXE vào bộ nhớ nó biết được địa chỉ các ô nhớ chứa các địa chỉ đoạn cần thay đổi cho phù hợp, hệ điều hành viết lại các giá trị này bằng cách cộng các giá trị trong đó với địa chỉ đoạn bắt đầu. Thao tác này có thể làm chậm tốc độ nạp các chương trình EXE. Sau khi các địa chỉ đoạn được sửa thành các địa chỉ có hiệu lực thì hệ điều hành cố định các thanh ghi đoạn DS và ES theo đầu của PSP ( $DS = ES = PSP$ ). Do đó chương trình EXE có thể truy cập dễ dàng đến các thông tin trong PSP.

*Các bạn có thể tìm hiểu rõ hơn về cấu trúc và lợi ích của header ở một tài liệu khác viết về các file EXE trong hệ điều hành MS\_DOS.*

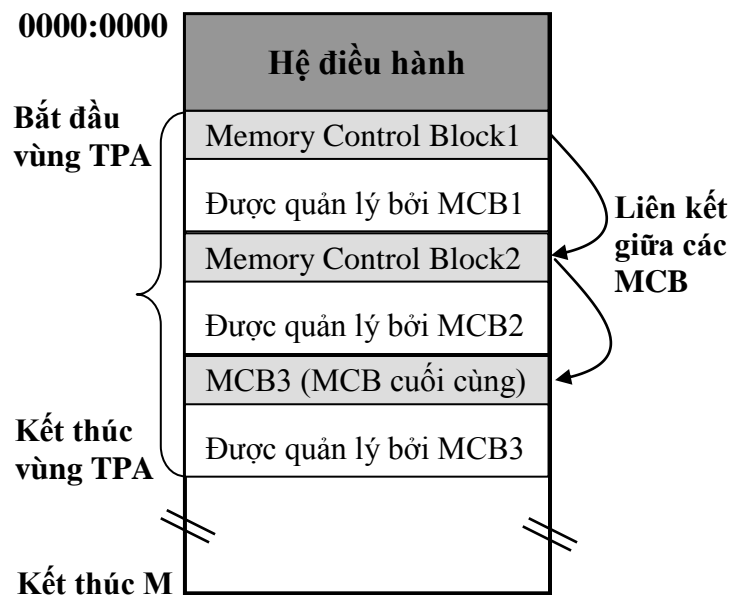
### **III.5.c. Memory Control Block (MCB):**

Hệ điều hành MS\_DOS là hệ điều hành đơn nhiệm đa chương, nên nó cho phép nạp nhiều hơn một chương trình vào bộ nhớ chính (RAM), mặc dù tại một thời điểm nhất định chỉ có duy nhất một chương trình thực hiện.

Theo MS\_DOS thì không gian bộ nhớ chính được chia thành 2 phần: phần thấp nhất (640B đầu tiên) được gọi là vùng bộ nhớ quy ước (conventional area). Phần còn lại được gọi là vùng nhớ trên (upper area). Các chương trình của người sử dụng phải nạp vào vùng nhớ quy ước để hoạt động. Vùng nhớ quy ước được chia thành hai phần: phần thấp nhất chứa các thành phần chính của hệ điều hành như là bảng vector ngắt, io.sys, msdos.sys, các chương trình điều khiển thiết bị, phần thường trực của command.com, vv. Phần còn lại của vùng quy ước được gọi là vùng TPA (Transient Program Area), tất cả các chương trình của người sử dụng phải được chạy tại đây. Như vậy, không phụ thuộc vào kích thước của bộ nhớ chính, MS\_DOS chỉ dành chưa đầy 640KB cho các chương trình của người sử dụng và các khối tham số môi trường tương ứng. Vùng nhớ trên được chia thành nhiều vùng nhỏ khác nhau như là video RAM, ROM BIOS, các điều khiển I/O, vv.

Để quản lý các khối nhớ đã được chia trên vùng nhớ cũng như các chương trình đã được nạp vào vùng TPA hệ điều hành MS\_DOS sử dụng các khối điều

khởi tạo bộ nhớ MCB. Mỗi MCB dài 16 byte (1 paragraphe), quản lý một vùng nhớ nằm ngay sau nó. Khi có một chương trình cần được nạp vào bộ nhớ, tùy theo yêu cầu bộ nhớ của chương trình mà DOS cung cấp cho chương trình một vùng nhớ, vùng nhớ này sẽ được quản lý bởi một MCB đứng ngay trước nó. Hình 3.16 sau đây minh họa cho điều này.



**Hình 3.16:** Quản lý bộ nhớ bằng MCB của DOS  
Cấu trúc của một MCB được mô tả ở hình dưới. Trong đó:

- Trường ID: định danh MCB, ID = 'Z': đây là MCB cuối cùng, ID = 'M': chưa phải là MCB cuối cùng.
- Trường địa chỉ PSP: đây là địa chỉ đoạn của PSP tương ứng của chương trình. Nếu vùng nhớ được cấp là khối môi trường của một chương trình thì trường này chỉ ra địa chỉ PSP của chính chương trình. Ngược lại nếu vùng nhớ được cấp là một PSP thì trong đa số trường hợp trường này chỉ ra chính vùng nhớ của chương trình.

1 byte	2 byte	2 byte	11 byte
ID	Địa chỉ PSP	Số lượng byte	Chưa sử dụng

**Hình 3.17:** Cấu trúc của một MCB

- Trường số lượng byte: trường này chỉ ra số lượng byte của vùng nhớ được cấp (tính theo đơn vị paragraphe), tức là nó cho biết khoảng cách từ một MCB thấp đến MCB kế tiếp cao hơn. Nhờ vậy mà các MCB trên bộ nhớ được kết nối như một danh sách liên kết.

Để nạp một chương trình vào bộ nhớ DOS cấp cho chương trình hai vùng

nhớ, vùng thứ nhất đủ để chứa khối tham số môi trường của chương trình, vùng thứ hai đủ để chứa chính chương trình và PSP tương ứng của nó. Việc xác định kích thước của chương trình dạng COM sẽ rất khó vì chương trình dạng COM được lưu trữ trên đĩa là hoàn toàn giống như nó ở trên bộ nhớ, ngoài ra không có thêm bất cứ thông tin nào. Đối với các chương trình dạng EXE thì dễ hơn vì kích thước của chương trình EXE được ghi trong Header, header đứng trước mọi chương trình EXE và được tạo ra trong quá trình biên dịch chương trình. Thông thường khi nạp một chương trình dạng COM vào bộ nhớ DOS sẽ cấp cho chương trình toàn bộ không gian nhớ còn lại của bộ nhớ.

### **III.13. Sự phân trang/đoạn trong hệ điều hành Windows NT**

Windows NT được thiết kế để cài đặt trên nhiều họ processor khác nhau, đặc biệt là trên các processor Intel 486. Trên các hệ thống PC IBM và tương thích sử dụng Intel 486, Windows chấp nhận kích thước của một page là 4KB, đây là cơ sở để thực hiện chiến lược bộ nhớ ảo của Windows NT.

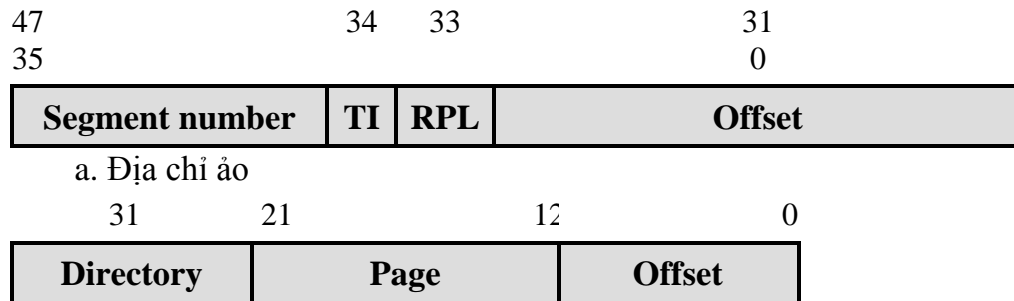
#### **III.5.a.Segmentation:**

Khi sự phân đoạn được sử dụng, mỗi địa chỉ ảo (trong các tài liệu 80386 được gọi là địa chỉ logic) bao gồm 16 bit segment và 32 bit offset. Hai bit đầu tiên của segment được sử dụng trong chiến lược bảo vệ bộ nhớ, 14 bit còn lại dùng để chỉ đến một segment cụ thể. Do đó với bộ nhớ không được phân đoạn, bộ nhớ ảo của chương trình người sử dụng là  $2^{32} = 4\text{GB}$ . Với bộ nhớ được phân đoạn, tổng không gian bộ nhớ ảo được nhìn bởi chương trình người sử dụng là  $2^{46} = 64\text{TB}$ . Không gian địa chỉ vật lý 32 bit cho kích thước tối đa là 4GB. Lượng bộ nhớ ảo thực tế có thể lớn hơn 64TB: Đối với 80386 địa chỉ ảo phụ thuộc vào processor đang sử dụng. Một nửa không gian địa chỉ ảo (8K segment x 4GB) là không gian chung, được chia sẻ cho tất cả các tiến trình, phần còn lại là địa chỉ cục bộ và dành riêng cho mỗi tiến trình.

Gắn liền với mỗi đoạn là 2 dạng thức của sự bảo vệ: mức đặc quyền (privilege level) và thuộc tính truy cập (access attribute). Có 4 mức đặc quyền, từ được bảo vệ cao nhất đến được bảo vệ thấp nhất: level 0, level 1, level 2, level 3. Mức đặc quyền có thể được gắn với segment dữ liệu hoặc với program segment. Một chương trình thực hiện chỉ có thể truy cập dữ liệu trong segment khi mức đặc quyền của nó thấp hơn hoặc bằng mức đặc quyền gắn với segment dữ liệu. Phần cứng không bắt buộc sử dụng cấp đặc quyền nào, việc này do các nhà thiết kế hệ điều hành quyết định. Thuộc tính truy cập của một segment dữ liệu chỉ ra có hay không sự cho phép truy cập read/write hoặc read-only. Đối với các segment program, thuộc tính truy cập chỉ ra có hay không truy cập read/execute hoặc read-only.

Chiến lược chuyển đổi địa chỉ cho sự phân đoạn gồm ánh xạ một địa chỉ ảo

thành địa chỉ tuyến tính và địa chỉ tuyến tính thành địa chỉ thực. Dạng thức của địa chỉ ảo trong NT được mô tả trong hình sau:

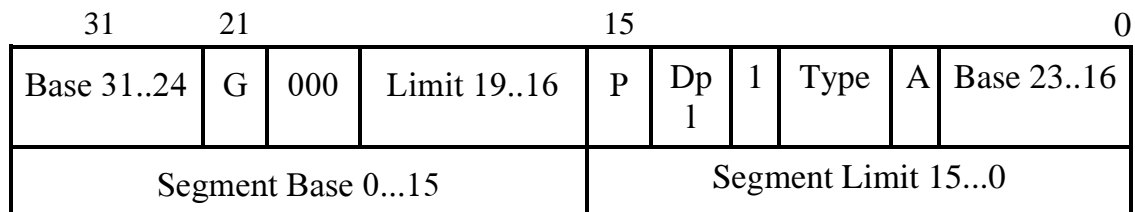


b. Địa chỉ tuyến tính

**Hình 3.18.a:** Địa chỉ ảo và địa chỉ tuyến tính của Intel 80486

- *Table indicator (TI)*: 1 bit, cho biết có hay không một global segment table hay local segment table là được sử dụng cho việc chuyển đổi địa chỉ.
- *Segment number*: 13 bit, là số hiệu của segment, nó được xem như là một chỉ mục vào segment table.
- *Offset*: 32 bit, khoảng cách từ một byte được đánh địa chỉ so với đầu segment.
- *Requested privilege level (RPL)*: 2 bit, mức đặc quyền truy cập được yêu cầu cho truy cập này.

Mỗi phần tử trong segment table bao gồm 64bit, được mô tả như hình sau:



**Hình 3.18.b:** Một phần tử trong segment table

- *Limit*: đây là kích thước của segment. Kích thước tối đa của một segment có thể là 1Mb (1 đơn vị = 1byte) hoặc 4Gb (1 đơn vị = 4Kb, điều này phụ thuộc vào bit Granularity).
- *Base*: cho biết địa chỉ bắt đầu của segment trong không gian tuyến tính 4Gb.
- *Accessed bit (A)*: khi segment tương ứng được truy cập thì bit này bằng 1. Các hệ điều hành sử dụng hệ thống bộ nhớ segmented – nonpaged dùng bit này để theo dõi việc sử dụng các segment. Đối với các hệ thống paged thì bit này bỏ qua.
- *Type*: cho biết đặc tính của các loại segment khác nhau và chỉ ra các thuộc tính truy cập.
- *Descriptor privilege level (DPL)*: chỉ ra mức đặc quyền của segment

(0-3).

- *Segment present bit (P)*: trong các hệ thống không được phân trang, bit này cho biết segment có trong bộ nhớ chính hay không. Trong các hệ thống phân trang bit này luôn luôn bằng a.
- *Granularity bit (G)*: cho biết một đơn vị cấp phát là 1 byte hay 4Kb (page). Bit này kết hợp với trường limit để tính kích thước của một segment.

### III.5.b.Paging:

Sự phân đoạn là sự lựa chọn cho tương lai và có thể khó cài đặt được. Khi sự phân đoạn được sử dụng, địa chỉ được sử dụng trong chương trình là địa chỉ ảo và được chuyển thành địa chỉ tuyến tính. Khi sự phân đoạn không được sử dụng, địa chỉ được sử dụng trong chương trình là địa chỉ tuyến tính. Trong cả hai trường hợp địa chỉ tuyến tính đều được chuyển thành địa chỉ thực 32bit.

Trong chiến lược phân trang của 80386, 80386 sử dụng bản trang 2 cấp. Cấp đầu tiên là thư mục bản trang (page directory), cấp thứ hai là bản trang (page table). Page directory có thể gồm 1024 phần tử. Tức là nó chia 4Gb không gian địa chỉ bộ nhớ thành 1024 nhóm trang, mỗi nhóm trang gồm 4Mb và sở hữu một bản trang riêng. Mỗi bản trang gồm 1024 phần tử, mỗi phần tử tương ứng với một trang đơn 4KB. Các hệ điều hành có thể lựa chọn sử dụng một page directory cho tất cả các tiến trình hoặc mỗi tiến trình có một page directory riêng hoặc kết hợp cả hai. Page directory của tác vụ hiện tại nằm trong bộ nhớ chính, page table có thể được chứa trong bộ nhớ ảo.

Sau đây là dạng thức của các phần tử trong page directory và page table:

31	11	0
Page <u>Table</u> Address 31..12	Avail 00	D A 00 US RW P

a. Một phần tử trong Page table directory

31	11	0
Page <u>Frame</u> Address 31..12	Avail 00	D A 00 US RW P

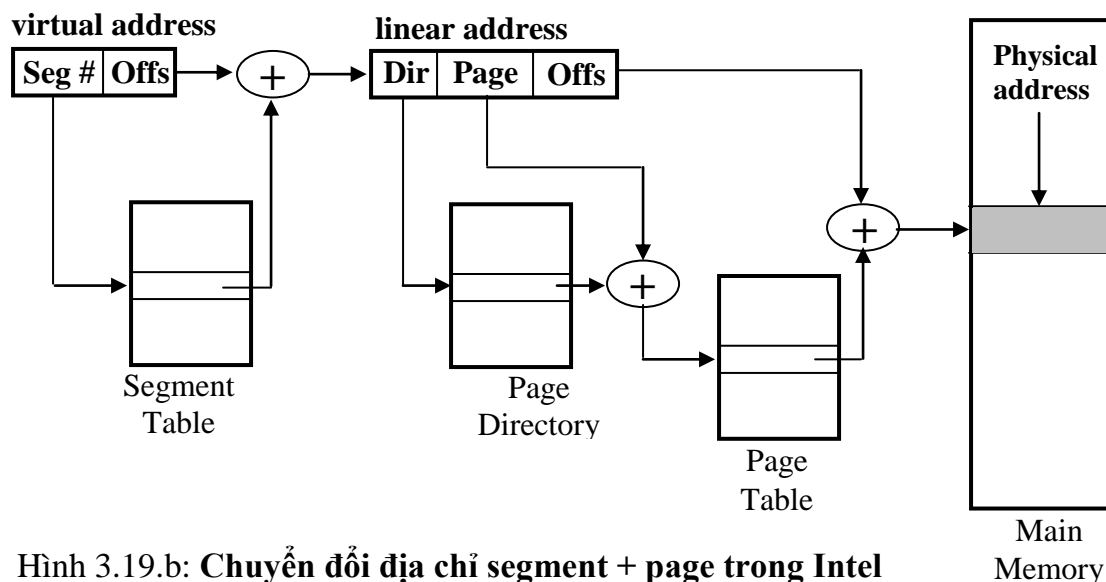
b. Một phần tử trong Page table

**Hình 3.19.a:** phần tử trong page directory và page table

- *Page frame address*: Nếu  $P = 1$  thì các bit này chỉ ra địa chỉ vật lý của một trang trong bộ nhớ.
- *Page table address*: Nếu  $P = 1$  thì các bit này chỉ ra địa chỉ vật lý của một bản trang trong bộ nhớ.
- *Present bit (P)*: bit này cho biết trang tương ứng đang ở trong bộ nhớ chính hay còn ở trên đĩa.
- *Accessed bit (A)*: bit này được processor bật lên 1 trong cả hai cấp của bản trang khi có một thao tác write/read được thực hiện ở trang tương ứng.

- *Dirty bit (D)*: bit này được processor bật lên 1 khi có một thao tác write được thực hiện ở trang tương ứng.
- *User/Supervisor bit (US)*: Bit này cho biết trang tương ứng dành riêng cho hệ điều hành hay cho cả hệ điều hành (cấp supervisor) và các chương trình ứng dụng (cấp user)
- *Read/write bit (RW)*: đối với các page cấp user, bit này cho biết trang tương ứng là có hay không sự truy cập read only. Đối với các page cấp program, bit này cho biết trang tương ứng là có hay không sự truy cập read Read/write.
- *Available bits (AVAIL)*: sẵn sàng cho người lập trình hệ thống sử dụng.

Chuyển địa chỉ trong kỹ thuật segmentation kết hợp paging của Windows NT trong hệ thống vi xử lý Intel 486.



Hình 3.19.b: Chuyển đổi địa chỉ segment + page trong Intel

Như vậy Intel 80486 hỗ trợ Windows NT cài đặt bộ nhớ ảo theo kỹ thuật phân đoạn kết hợp phân trang. Cần nhắc lại rằng trong kỹ thuật bộ nhớ ảo cần phải có sự hỗ trợ của cả phần cứng (processor) và phần mềm. Processor thực hiện 2 nhiệm vụ chính là thực hiện việc chuyển đổi động từ địa chỉ ảo thành địa chỉ vật lý và phát sinh ngắt khi có một sự tham chiếu đến một trang hoặc đoạn mà trang đoạn này không có trên bộ nhớ chính (lỗi trang).

### III.14. Các thuật toán thay trang

Như đã biết, để xử lý lỗi trang, trong trường hợp trên bộ nhớ không còn frame trống, hệ điều hành phải tìm một page nào đó trên bộ nhớ chính để đưa ra đĩa, để lấy frame trống đó để phục vụ cho việc xử lý lỗi trang. Khi quyết định chọn một

page nào đó để đưa ra đĩa thì hệ điều hành phải đảm bảo rằng việc chọn này là: không ảnh hưởng đến các tiến trình khác, ít có nguy cơ xảy ra lỗi trang ngay sau đó nhất và đặc biệt hệ thống khó có thể rơi vào tình trạng “trì trệ hệ thống” nhất. Trong trường hợp này hệ điều hành đã đưa vào sử dụng các thuật toán thay trang cụ thể như: Optimal, LRU, FIFO, Clock.

Các thuật toán thay trang khác nhau có các tiêu chí để chọn trang swap out khác nhau, nhưng tất cả đều hướng tới mục tiêu là: đơn giản và ít xảy ra lỗi trang nhất. Nó không quan tâm đến việc page được chọn để swap out là trang của tiến trình gây ra lỗi trang hay trang của một tiến trình nào đó trong hệ thống. Các thuật toán thay trang không xem xét đến các trang bị đánh dấu “neo”.

Để so sánh hiệu suất xử lý lỗi trang của các thuật toán thay trang, chúng ta phải áp dụng các thuật toán này trong cùng một điều kiện: có cùng số lượng frame còn trống ban đầu và cần phải nạp một danh sách các trang như nhau vào bộ nhớ. Thuật toán được gọi là có hiệu suất cao hơn khi nó xảy ra ít lỗi trang hơn.

Trong các thuật toán sau đây chúng ta xem xét trong trường hợp: ban đầu hệ thống có 3 frame còn trống và hệ điều hành cần phải nạp một danh sách các trang sau đây vào bộ nhớ: **2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2**.

Trong các thuật toán sau đây chúng ta chỉ xét đến trường hợp b của lỗi trang, đó là hệ điều hành phải xử lý lỗi trang khi trên bộ nhớ chính không còn khung trang trống.

### ➤ Thuật toán FIFO (First In First Out)

Thuật toán FIFO là thuật toán đơn giản và dễ cài đặt nhất. Với thuật toán này thì trang mà hệ điều hành chọn để swap out là trang được đưa vào bộ nhớ sớm nhất, hay ở trong bộ nhớ lâu nhất. Bảng sau đây minh họa cho việc chọn trang để swap out và thay thế của thuật toán FIFO:

	2	3	2	1	5	2	4	5	3	2	5	2
Frame 1	2	2	2	2	5	5	5	5	3	3	3	3
Frame 2		3	3	3	3	2	2	2	2	2	5	5
Frame 3				1	1	1	4	4	4	4	4	2
						F	F	F		F		F

Theo bảng trên thì trong trường hợp này đã xảy ra 6 lỗi trang, khi hệ điều hành cần nạp trang 5 vào bộ nhớ thì nó phải đưa trang 2 ra ngoài để lấy frame1 nạp trang 5, khi hệ điều hành cần nạp lại trang 2 vào bộ nhớ thì nó phải đưa trang 3 ra ngoài để lấy frame2 nạp trang 2, khi hệ điều hành cần nạp trang 4 vào bộ nhớ thì nó phải đưa trang 1 ra ngoài để lấy frame3 nạp trang 4, ...



Thuật toán này không phải lúc nào cũng mang lại hiệu quả tốt. Thứ nhất, có thể trang được đưa vào bộ nhớ lâu nhất lại là trang cần được sử dụng ngay sau đó, tức là hệ điều hành vừa swap out nó thì phải swap in nó trở lại bộ nhớ ngay và rõ ràng trong trường hợp này hệ điều hành lại phải tiếp tục việc xử lý lỗi trang, trường hợp của trang 2 ở trên là một ví dụ. Thứ hai, có thể lỗi trang sẽ tăng lên khi số lượng khung trang được sử dụng tăng lên, trường hợp này được gọi là nghịch lý Belady. Khi hệ điều hành cần nạp các trang sau đây theo thứ tự vào bộ nhớ: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 thì sẽ xảy ra 9 lỗi trang nếu sử dụng 3 khung trang, và sẽ xảy ra 10 lỗi trang nếu sử dụng 4 khung trang.

Với thuật toán này hệ điều hành cần phải có cơ chế thích hợp để ghi nhận thời điểm một trang được nạp vào bộ nhớ để làm cơ sở thay thế sau này. Trong trường hợp này hệ điều hành thường sử dụng một danh sách liên kết để ghi nhận các trang được nạp vào bộ nhớ, trang được nạp vào sớm nhất được ghi nhận ở đầu danh sách, trang được nạp vào muộn nhất được ghi nhận ở cuối danh sách và trang được chọn để thay thế là trang ở đầu danh sách. Hệ điều hành sẽ xóa phần tử ở đầu danh sách này ngay sau khi một trang đã được thay thế.

#### ➤ Thuật toán LRU (Least Recently Used)

Theo thuật toán này thì trang được hệ điều hành chọn để thay thế là trang có khoảng thời gian từ lúc nó được truy xuất gần đây nhất đến thời điểm hiện tại là dài nhất, so với các trang đang ở trên bộ nhớ chính. Như vậy trong trường hợp này hệ điều hành phải ghi nhận thời điểm cuối cùng trang được truy xuất. Bảng sau đây minh họa cho việc chọn trang để swap out và thay thế của thuật toán LRU:

	2	3	2	1	5	2	4	5	3	2	5	2
Frame 1	2	2	2	2	2	2	2	2	3	3	3	3
Frame 2		3	3	3	5	5	5	5	5	5	5	5
Frame 3				1	1	1	4	4	4	2	2	2
							F	F		F	F	

Theo bảng trên thì trong trường hợp này xảy ra 4 lỗi trang, khi hệ điều hành cần nạp trang 5 vào bộ nhớ thì nó phải đưa trang 3 ra ngoài để lấy frame2 nạp trang 5, vì hệ điều hành thấy rằng trang 3, chứ không phải là trang 2, là trang vừa được truy xuất cách đây lâu nhất.

Thuật toán này cần phải có sự hỗ trợ của phần cứng để xác định thời điểm gần đây nhất trang được truy xuất của tất cả các trang trên bộ nhớ. Có hai cách được áp dụng:

- Sử dụng bộ đếm: trong cách này, các processor thêm vào cấu trúc của

các phần tử bảng trang một trường mới, tạm gọi là trường LRU, trường này ghi nhận thời điểm trang tương ứng được truy xuất gần đây nhất. Và thêm vào cấu trúc của CPU một bộ đếm (Counter). Mỗi khi có sự truy xuất bộ nhớ thì Counter tăng lên một đơn vị. Mỗi khi một trang trên bộ nhớ được truy xuất thì giá trị của Counter sẽ được ghi vào trường LRU tại phần tử trong bảng trang tương ứng với trang này. Như vậy trang được chọn để thay thế là trang có LRU là nhỏ nhất.

- Sử dụng Stack: trong cách này hệ điều hành sử dụng một Stack để lưu trữ số hiệu của các trang đã được nạp vào bộ nhớ chính. Khi một trang được truy xuất thì số hiệu của trang này sẽ được xóa khỏi Stack tại vị trí hiện tại và được đưa lên lại đỉnh Stack. Như vậy trang có số hiệu nằm ở đỉnh stack là trang được sử dụng gần đây nhất, trang có số hiệu nằm ở đáy stack là trang lâu nay ít được sử dụng nhất. Và trang được chọn để thay thế là các trang có số hiệu nằm ở đáy stack.

### ➤ Thuật toán Optimal (tối ưu)

Theo thuật toán này thì trang được hệ điều hành chọn để thay thế là trang sẽ lâu được sử dụng nhất trong tương lai. Bảng sau đây minh họa cho việc chọn trang để swap out và thay thế của thuật toán Optimal:

	2	3	2	1	5	2	4	5	3	2	5	2
Frame 1	2	2	2	2	2	2	4	4	4	2	2	2
Frame 2		3	3	3	3	3	3	3	3	3	3	3
Frame 3			1	1	5	5	5	5	5	5	5	5
				F		F			F			

Theo bảng trên thì trong trường hợp này chỉ xảy ra 3 lỗi trang, khi hệ điều hành cần nạp trang 5 vào bộ nhớ thì nó phải đưa trang 1 ra ngoài để lấy frame3 nạp trang 5, vì hệ điều hành cho rằng trang 1 là trang sẽ lâu được sử dụng trong tương lai.

Mặc dầu thuật toán này ít xảy ra lỗi trang hơn, nhưng trong thực tế khó có thể cài đặt được vì hệ điều hành khó có thể đoán trước được khi nào thì một trang được truy xuất trở lại. Thuật toán này không chịu tác động của nghịch lý Belady.

**Chú ý:** Các tài liệu về hệ điều hành đã đưa ra rất nhiều thuật toán thay trang, nhưng chúng tôi không trình bày ở đây, các bạn có thể tìm đọc ở tài liệu tham khảo [1] và [2].

## III.15. Cấp phát khung trang

Với kỹ thuật bộ nhớ ảo phân trang thì hệ điều hành không cần và cũng không thể mang tất cả các page của một tiến trình nạp vào bộ nhớ chính để chuẩn bị thực

hiện. Vì vậy hệ điều hành cần phải quyết định nạp bao nhiêu page, bao nhiêu tiến trình vào bộ nhớ. Hay chính xác hơn là nạp bao nhiêu tiến trình và mỗi tiến trình được nạp bao nhiêu page vào bộ nhớ (được cấp bao nhiêu khung trang). Hệ điều hành có thể quyết định vấn đề này theo các chọn lựa sau đây:

- Chỉ có một lượng nhỏ, có thể là tối thiểu, các page của tiến trình được nạp vào bộ nhớ. Như vậy hệ điều hành sẽ nạp được nhiều tiến trình vào bộ nhớ tại bất kỳ thời điểm nào. Điều này làm tăng khả năng đa chương của hệ điều hành và khả năng tìm thấy một tiến trình Ready của hệ điều hành là rất lớn nhờ vậy mà hiệu quả điều phối của hệ điều hành tăng lên. Nhưng trong trường hợp này hệ điều hành phải luôn chú ý đến việc nạp thêm các page của tiến trình vào bộ nhớ và hệ điều hành khó có thể xác định được số lượng khung trang tối thiểu mà mỗi tiến trình cần khi khởi tạo.

- Nếu có một lượng vừa phải các page của một tiến trình trong bộ nhớ chính thì có ít hơn số tiến trình được nạp vào bộ nhớ. Như vậy sự đa chương sẽ giảm xuống nhưng tốc độ thực hiện của tiến trình có thể được cải thiện vì khi một chỉ thị của các page trong bộ nhớ chính cần truy xuất đến một page khác thì nhiều khả năng page này đã có trên bộ nhớ chính. Nhưng lý thuyết hệ điều hành đã chứng minh được rằng trong trường hợp này tỉ lệ xảy ra lỗi trang là rất lớn.

- Nếu có một lượng lớn các page của một tiến trình trong bộ nhớ chính, thì sự đa chương sẽ giảm xuống đáng kể. Nhưng lý thuyết hệ điều hành đã chứng minh được rằng trong trường hợp này tỉ lệ xảy ra lỗi trang là rất thấp. Mặt khác điều này có thể gây lãng phí bộ nhớ vì có thể có các page của một tiến trình rất ít được sử dụng khi nó ở trên bộ nhớ chính.

Theo trên thì mỗi chọn lựa đều có những điểm thuận lợi và những điểm chưa thuận lợi riêng, do đó tùy trường hợp cụ thể mà hệ điều hành thực hiện cấp phát khung trang cho tiến trình theo một chọn lựa nào đó, để đảm bảo có nhiều tiến trình được nạp vào bộ nhớ chính, nhưng khả năng và tỉ lệ lỗi trang là thấp nhất và sự lãng phí bộ nhớ là thấp nhất. Để đáp ứng điều này các hệ điều hành thường thực hiện việc cấp phát khung trang cho các tiến trình theo hai chính sách: *Cấp phát tĩnh* và *Cấp phát động*

- Chính sách cấp phát tĩnh (*fixed – allocation*): Với chính sách này hệ điều hành sẽ cấp cho mỗi tiến trình một số lượng khung trang cố định, để nạp đủ các page của tiến trình vào bộ nhớ để nó có thể hoạt động được. Số lượng khung trang này được quyết định tại thời điểm khởi tạo/tạo tiến trình. Hệ điều hành cũng có thể quyết định số lượng khung trang tối thiểu cho tiến trình dựa vào loại của tiến trình, đó là tiến trình tương tác, tiến trình xử lý theo lô hay tiến trình theo hướng ứng dụng. Với cấp phát tĩnh, khi có lỗi trang xảy ra trong quá trình thực hiện tiến trình thì hệ điều hành phải swap out một page của tiến trình đó để thực hiện việc xử lý lỗi trang.

- Chính sách cấp phát động (*variable - allocation*): Với chính sách này hệ điều hành chỉ cấp một lượng vừa đủ khung trang, để nạp đủ các trang cần thiết nhất của tiến trình, để tiến trình có thể khởi tạo và hoạt động được, sau đó tùy theo yêu cầu của tiến trình mà hệ điều hành có thể cấp phát thêm khung trang cho nó, để nạp thêm các trang cần thiết khác. Hệ điều hành thường cấp thêm khung trang cho tiến trình khi tiến trình bị rơi vào tình trạng lỗi trang, với các tiến trình có tần suất xảy ra lỗi trang lớn thì hệ điều hành phải cung cấp một lượng khung trang lớn, một cách vượt bậc, đủ để tiến trình thoát ra khỏi lỗi trang và nguy cơ lỗi trang tiếp theo là thấp nhất.

### **III.16. Một số vấn đề về quản lý bộ nhớ của Windows 2000**

#### **III.8.1. Nhiệm vụ quản lý bộ nhớ của Windows 2000**

Thành phần quản lý bộ nhớ của Windows 2000 thực hiện hai nhiệm vụ chính sau đây:

- Chuyển đổi, hay ánh xạ, không gian địa chỉ ảo của một tiến trình vào bộ nhớ vật lý để khi một tiến trình thực thi trong một ngữ cảnh của tiến trình đó, đọc hay ghi vào không gian địa chỉ ảo thì địa chỉ vật lý chính xác sẽ được tham chiếu.

- Phân trang một vài nội dung bộ nhớ ra đĩa (swap out) khi nó trở nên vượt quá sự đáp ứng bộ nhớ của hệ thống. Có nghĩa là, khi việc thực thi các tiến trình hay mã hệ thống cố gắng sử dụng nhiều bộ nhớ vật lý hơn khả năng hiện thời – và mang nội dung trở lại vào bộ nhớ vật lý (swap in) khi cần.

Hệ điều hành Windows 2000 Professional và Server hỗ trợ lên đến 4GB bộ nhớ vật lý, Windows 2000 Advanced Server thì hỗ trợ lên đến 8 GB, và Windows 2000 Datacenter Server thì lên đến 64 GB. Thực tế bộ nhớ lớn nhất cho Windows 2000 Datacenter Server phụ thuộc vào khả năng phần cứng. Bởi vì Windows 2000 là một hệ điều hành 32-bit, nên các tiến trình người sử dụng có một không gian địa chỉ ảo 32-bit, 4GB bộ nhớ phẳng.

Ngoài việc cung cấp sự quản lý bộ nhớ ảo, trình quản lý bộ nhớ cung cấp một tập lỗi các dịch vụ mà trong đó các hệ thống con môi trường Windows 2000 khác nhau được xây dựng. Các dịch vụ này bao gồm các tập tin ánh xạ bộ nhớ, bộ nhớ copy-on-write, và hỗ trợ cho các ứng dụng sử dụng các không gian địa chỉ lớn, không liên tiếp.

Cũng như tất cả các thành phần của windows 2000 executive, trình quản lý bộ nhớ hỗ trợ sự thực thi đồng thời trên các hệ thống đa xử lý. Nó cho phép hai tiến trình thu được các tài nguyên theo cách mà sau này chúng không làm cho hỏng dữ liệu của mỗi tiến trình khác. Để đạt được mục tiêu này, trình quản lý bộ nhớ sử dụng một vài cơ chế đồng bộ nội tại khác nhau để điều khiển sự truy xuất vào các cấu trúc dữ liệu nội tại của riêng nó.

### III.8.2. Các dịch vụ trình quản lý bộ nhớ cung cấp

Trình quản lý bộ nhớ cung cấp một tập các dịch vụ hệ thống để định vị và giải phóng bộ nhớ ảo, chia sẻ bộ nhớ giữa các tiến trình, ánh xạ các tập tin vào bộ nhớ, flush các trang ảo vào đĩa, truy lục thông tin về một vùng các trang ảo, thay đổi sự bảo vệ của các trang ảo, và khoá các trang ảo vào bộ nhớ.

Trình quản lý bộ nhớ cũng cung cấp một lượng các dịch vụ, như định vị và bỏ định vị bộ nhớ vật lý và khoá các trang trong bộ nhớ vật lý cho các trao đổi truy xuất bộ nhớ trực tiếp (DMA), đến các thành phần chế độ kernel khác bên trong Executive cũng như các device driver. Các hàm này bắt đầu với tiền tố *Mm*. Ngoài ra, mặc dù không hoàn toàn là một phần của trình quản lý bộ nhớ, Executive hỗ trợ các thường trình bắt đầu với *Ex* mà nó được sử dụng để định vị và bỏ định vị từ các heap hệ thống (vùng phân trang và không phân trang) cũng như để vận dụng các danh sách look-aside.

**Sau đây chúng ta sẽ xem xét một vài trong nhiều dịch vụ mà trình quản lý bộ nhớ của Windows 2000 cung cấp:**

#### ➤ Bảo vệ bộ nhớ

Windows 2000 cung cấp sự quản lý bộ nhớ để không một tiến trình người sử dụng nào, có thể không cố ý hay cố ý, làm hỏng không gian địa chỉ của các tiến trình khác hoặc của chính hệ điều hành. Windows 2000 cung cấp sự bảo vệ này theo bốn cách chính sau đây:

- Thứ nhất, tất cả các cấu trúc dữ liệu và các vùng bộ nhớ được sử dụng bởi các thành phần hệ thống kernel-mode chỉ thể được truy xuất trong kernel-mode. Các tiểu trình user-mode không thể truy xuất các page này. Nếu các tiểu trình này cố gắng thực hiện sự truy xuất này thì phần cứng phát sinh một lỗi, và trình quản lý bộ nhớ sẽ gửi thông báo vi phạm truy xuất đến cho tiểu trình.

- Thứ hai, mỗi tiến trình có một không gian địa chỉ riêng, tách biệt, được bảo vệ khỏi bị truy xuất bởi bất kỳ tiểu trình nào thuộc một tiến trình khác. Chỉ các ngoại lệ là nếu một tiến trình đang chia sẻ các trang với các tiến trình khác hay nếu một tiến trình khác có truy xuất đọc hay ghi bộ nhớ ảo vào đối tượng tiến trình và do đó có thể sử dụng các hàm *ReadProcessMemory* hay *WriteProcessMemory*. Mỗi khi một tiểu trình tham chiếu một địa chỉ, phần cứng bộ nhớ ảo, phối hợp với trình quản lý bộ nhớ, can thiệp và chuyển đổi địa chỉ ảo thành một địa chỉ vật lý. Bằng cách điều khiển các địa chỉ ảo được chuyển đổi, Windows 2000 có thể đảm bảo các tiểu trình đang thực thi trong một tiến trình không truy xuất bất hợp lệ một trang thuộc một tiến trình khác.

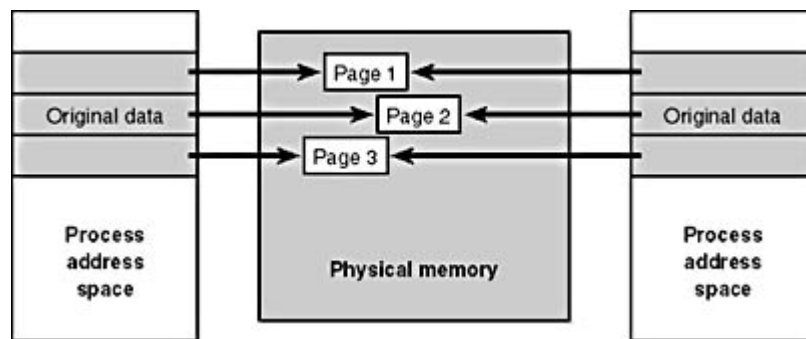
- Thứ ba, ngoài các cung cấp sự bảo vệ mặc nhiên cho việc chuyển đổi địa chỉ ảo thành đại chỉ vật lý, tất cả các processor được hỗ trợ bởi Windows 2000 cung cấp một số hình thức bảo vệ bộ nhớ được điều khiển bởi phần cứng (như đọc/ghi; chỉ đọc, ...); chi tiết chính xác của sự bảo vệ như vậy thay đổi theo

processor. Ví dụ, các page mã trong không gian địa chỉ của một tiến trình được đánh dấu chỉ đọc và do đó được bảo vệ khỏi sự sửa đổi bởi các tiến trình người sử dụng. Các page mã cho các tiến trình điều khiển thiết bị cũng được đánh dấu chỉ đọc như vậy.

- Và cuối cùng, các section object bộ nhớ chia sẻ có các danh sách điều khiển truy xuất Windows 2000 chuẩn, mà nó được kiểm tra khi các tiến trình cố gắng mở chúng, do đó việc giới hạn truy xuất của bộ nhớ chia sẻ đến các tiến trình này với các quyền thích hợp. Bảo mật cũng thừa hưởng cách hoạt động khi một tiến trình tạo một section để chứa một tập tin ảnh xạ. Để tạo một section, tiến trình phải có ít nhất truy xuất đọc đến đối tượng tập tin cơ sở hay thao tác sẽ lỗi.

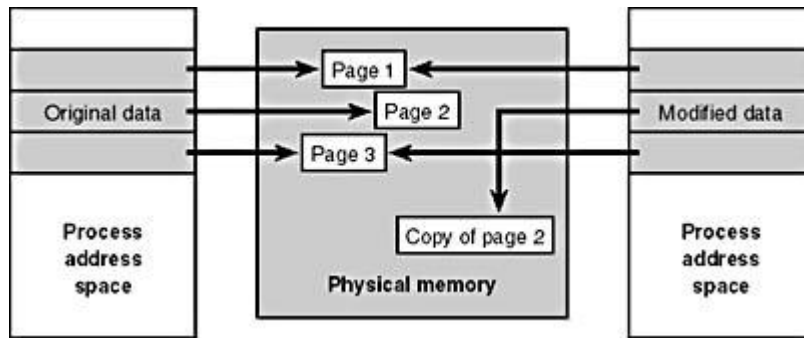
### ➤ Copy-On-Write

Sự bảo vệ các trang copy-on-write là một sự tối ưu trong việc quản lý bộ nhớ của Windows 2000. Để thấy được ý nghĩa của việc sử dụng các trang copy-on-write chúng ta hãy xem ví dụ sau đây: Có hai tiến trình đang chia sẻ ba trang (page1, page2, page3), mỗi trang được đánh dấu là copy-on-write, nhưng cả hai tiến trình đều không sửa đổi bất kỳ dữ liệu nào trên các trang.



**Hình 3.20.a:** “Trước” copy-on-write

Nếu một tiến trình của một trong hai tiến trình này ghi vào một trang, một lỗi quản lý bộ nhớ được phát sinh. Trình quản lý bộ nhớ xem việc ghi đó là vào trang copy-on-write, nên thay vì báo lỗi như một vi phạm truy xuất, thì nó định vị một trang read/write mới trong bộ nhớ vật lý, sau đó sao chép nội dung của trang ban đầu vào trang mới, cập nhật thông tin bảng trang tương ứng của tiến trình này để trở đến một vị trí mới, và thao tác ghi ở trên sẽ được hệ thống chuyển hướng để thực hiện ở trang mới này. Lần này, thao tác ghi hoàn thành, nhưng như trình bày trong hình sau, trang được sao chép mới bây giờ là sở hữu của tiến trình thực hiện ghi và không thấy được từ các tiến trình khác, vẫn đang chia sẻ trang copy-on-write. Mỗi tiến trình mới ghi vào cùng trang được chia sẻ này cũng sẽ nhận bản sao riêng của nó.



**Hình 3.20.b:** "Sau" copy-on-write

Một ứng dụng của copy-on-write là để cài đặt điểm ngắt hỗ trợ trong các trình gỡ rối. Ví dụ, mặc định, các trang mã bắt đầu chỉ thực thi. Tuy nhiên, nếu một lập trình viên thiết đặt một điểm ngắt trong khi gỡ rối một chương trình, thì trình gỡ rối phải thêm một chỉ thị điểm ngắt vào mã. Nó thực hiện điều đó bằng cách đầu tiên thay đổi sự bảo vệ trang thành `PAGE_EXECUTE_READWRITE` và sau đó thay đổi luồng chỉ thị. Bởi vì trang mã là một phần của một mapped section, nên trình quản lý bộ nhớ tạo một bản sao riêng cho tiến trình với tập điểm ngắt, trong khi các tiến trình khác tiếp tục sử dụng trang mã chưa sửa đổi.

Hệ thống con POSIX lợi dụng copy-on-write để cài đặt chức năng *fork* (phân nhánh). Điển hình, khi một ứng dụng UNIX gọi một hàm *fork* để tạo một tiến trình khác, điều đầu tiên mà tiến trình mới thực hiện là gọi hàm *exec* để khởi tạo lại không gian địa chỉ với một ứng dụng có thể thực thi. Thay vì sao chép toàn bộ không gian địa chỉ trên *fork*, tiến trình mới chia sẻ các trang trong tiến trình cha bằng cách đánh dấu chúng là copy-on-write. Nếu một tiến trình con ghi lên dữ liệu, một bản sao riêng tiến trình được thực hiện. Nếu không, hai tiến trình tiếp tục chia sẻ và không có việc sao chép nào được thực hiện. Một cách hay một cách khác, trình quản lý bộ nhớ chỉ sao chép các trang tiến trình cố gắng ghi thay vì sao chép toàn bộ không gian địa chỉ.

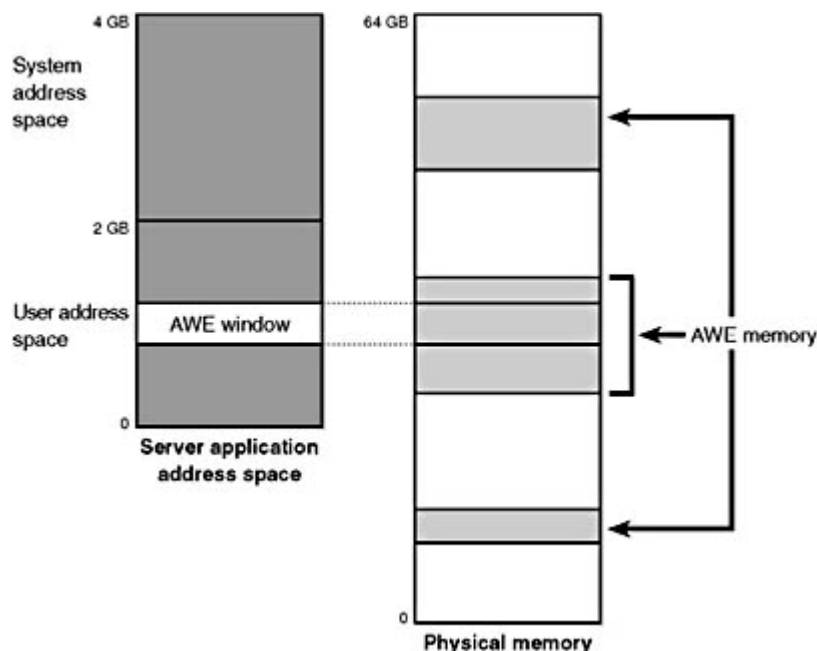
### ➤ **AWE: Address Windowing Extension**

Mặc dù hệ điều hành Windows 2000 có thể hỗ trợ trên 64 GB bộ nhớ vật lý, nhưng mỗi tiến trình người sử dụng 32-bit chỉ có một không gian địa chỉ ảo 2 GB hoặc 3 GB. Để cho phép một tiến trình 32-bit định vị và truy xuất nhiều bộ nhớ vật lý hơn, có thể được thể hiện trong không gian địa chỉ bị giới hạn của nó, Windows 2000 cung cấp một tập các hàm được gọi là Address Windowing Extensions (AWE). Ví dụ, trên hệ thống Windows 2000 Advanced Server với 8 GB bộ nhớ vật lý, một ứng dụng cơ sở dữ liệu server có thể sử dụng AWE để định vị và sử dụng gần 8 GB bộ nhớ như một cache cơ sở dữ liệu.

Việc định vị và sử dụng bộ nhớ thông qua các hàm AWE được thực hiện qua ba bước:

1. Định vị bộ nhớ vật lý để được sử dụng.

2. Tạo một vùng không gian địa chỉ ảo để hoạt động như một cửa sổ để ánh xạ các khung nhìn của bộ nhớ vật lý.
3. Ánh xạ các khung nhìn của bộ nhớ vật lý vào cửa sổ.



**Hình 3.21:** Sử dụng AWE để ánh xạ bộ nhớ vật lý

Để định vị bộ nhớ vật lý, một ứng dụng gọi hàm `Win32 AllocateUserPhysicalPages`. Ứng dụng sau đó sử dụng hàm `Win32 VirtualAlloc` với cờ `MEM_PHYSICAL` để tạo một cửa sổ trong phần riêng của không gian địa chỉ của tiến trình mà nó được ánh xạ đến một số hoặc tất cả bộ nhớ vật lý được định vị trước đây. Bộ nhớ được AWE định vị có thể sau đó với gần tất cả các hàm Win32 API.

Nếu một ứng dụng tạo một cửa sổ 256Mb trong không gian địa chỉ của nó và định vị 4Gb bộ nhớ vật lý (trên một hệ thống với hơn 4 GB bộ nhớ vật lý), ứng dụng có thể sử dụng các hàm `Win32 MapUserPhysicalPages` hay `MapUserPhysicalPagesScatter` để truy xuất bất kỳ phần nào của cửa sổ không gian địa chỉ ảo xác định lượng bộ nhớ vật lý mà ứng dụng có thể truy xuất với một ánh xạ nhất định. Hình 3.21 trên đây trình bày một cửa sổ AWE trong một không gian địa chỉ ứng dụng phục vụ được ánh xạ đến một vùng bộ nhớ vật lý được định vị trước đó bằng `AllocateUserPhysicalPages`.

Các hàm AWE tồn tại trên tất cả các ấn bản của Windows 2000 và có thể được sử dụng bất chấp hệ thống có bao nhiêu bộ nhớ vật lý. Tuy nhiên, AWE hữu ích nhất trên các hệ thống với nhiều hơn 2 GB bộ nhớ vật lý, bởi vì nó chỉ là cách cho tiến trình 32-bit trực tiếp sử dụng nhiều hơn 2 GB bộ nhớ.

Cuối cùng, có một số hạn chế trong việc định vị bộ nhớ và định xạ bằng các hàm



AWE:

- Các trang không thể chia sẻ giữa các tiến trình.
- Cùng một trang vật lý không thể được ánh xạ nhiều hơn một địa chỉ ảo trong cùng tiến trình.
- Sự bảo vệ trang chỉ giới hạn đến read/write.

### III.8.3. Address Space Layout

Theo mặc định, mỗi tiến trình người sử dụng trên phiên bản 32-bit của Windows 2000 có thể có trên 2Gb không gian địa chỉ riêng; hệ điều hành giữ 2Gb. Windows 2000 Advanced Server và Windows 2000 Datacenter Server hỗ trợ một tùy chọn tại thời điểm khởi động nó cho phép không gian địa chỉ tiến trình/chương trình người sử dụng lên đến 3Gb.

Tùy chọn không gian địa chỉ 3Gb mang lại cho các tiến trình một không gian địa chỉ 3Gb (dành 1Gb cho không gian hệ thống). Đặc tính này được thêm vào như một giải pháp tình thế để đáp ứng sự cần thiết cho các ứng dụng server cơ sở dữ liệu để giữ nhiều dữ liệu hơn trong bộ nhớ so với khi thực hiện với không gian địa chỉ 2Gb.

Không gian địa ảo của các hệ điều hành windows trước được tổ chức khác hơn so với Windows 2000. Nó cũng cung cấp một không gian địa chỉ ảo 32 bit 4Gb và cấp phát không gian địa chỉ 2Gb riêng cho mỗi tiến trình người sử dụng, nhưng nó chia 2Gb còn lại thành 2 phần, 1Gb cho không gian hệ thống, 1Gb dùng làm không gian chia sẻ cho tất cả các tiến trình người sử dụng.

➤ **Không gian địa chỉ hệ thống:** Trong các kiến trúc Intel x86, không gian địa chỉ 2Gb của hệ thống được phân thành các vùng khác nhau, được mô tả ở hình 3.22 sau:

Trong đó:

- **System code:** Chứa chính hệ điều hành, HAL và các điều khiển thiết bị được sử dụng để boot hệ thống.
- **System mapped views:** Sử dụng để ánh xạ Win32k.Sys, có thể nạp một phần của hệ thống con Win32 trong chế độ kernel mode và các điều khiển đồ họa.
- **Session space:** Được sử dụng để ánh xạ thông tin đến một session người sử dụng cụ thể.
- **Process page tables and page directory:** Được dùng để chứa các bảng trang tiến trình và các danh mục bảng trang.
- **Hyperspace:** Đây là một vùng đặc biệt, được sử dụng để ánh xạ danh sách working set của tiến trình và để ánh xạ tạm các trang vật lý khác.
- **System working set list:** Chứa các trúc dữ liệu danh sách working set mà nó mô tả working set của hệ thống.

- **System cache:** Không gian địa chỉ ảo được sử dụng để ánh xạ các file mở trong hệ thống cache.
- **Paged pool:** Chứa các pool được phân trang.
- **System page table entries (PTEs):** Pool của các PTEs hệ thống được sử dụng để ánh xạ các trang hệ thống như: không gian I/O, các stack kernel và các danh sách mô tả bộ nhớ.
- **Nonpaged pool:** Chứa các pool không được phân trang.
- **Crash dump information:** Được dự trữ để ghi thông tin về trạng thái của một hệ thống Crash.
- **HAL usage:** Hệ thống bộ nhớ dự trữ cho kiến trúc HAL đặc biệt.

x86	
80000000	System code (Ntoskrnl, HAL) and initial nonpaged pool on some systems
A0000000	System mapped views (e.g., Win32k.sys) or session space
A4000000	Additional system PTEs (Cache can extend here)
C0000000	Process page tables and page directory
C0400000	Hyperspace and process working set list
C0800000	Unused – no access
C0C00000	System working set list
C1000000	System cache
E1000000	Paged pool
EB000000 (min)	System PTEs
	Nonpaged pool expansion
FFBE0000	Crash dump information
FFC00000	HAL usage

**Hình 3.22:** Phân lớp không gian địa chỉ trong x86

**Chú ý:** Khi khởi tạo hệ thống, trình quản lý bộ nhớ tạo hai kiểu vùng nhớ pool định kích thước tự động mà các thành phần chế độ kernel sử dụng để định vị bộ nhớ hệ thống:

- **Vùng Pool không phân trang:** bao gồm các vùng địa chỉ ảo hệ thống được bảo đảm tồn tại **trong** bộ nhớ vật lý tại tất cả các thời điểm và do đó có thể được truy xuất bất cứ khi nào mà không mắc phải một lỗi trang.

- **Vùng pool phân trang:** một vùng bộ nhớ ảo trong không gian hệ thống có thể được phân trang trong và ngoài hệ thống. Các trình điều khiển thiết bị có thể sử dụng vùng phân trang này.

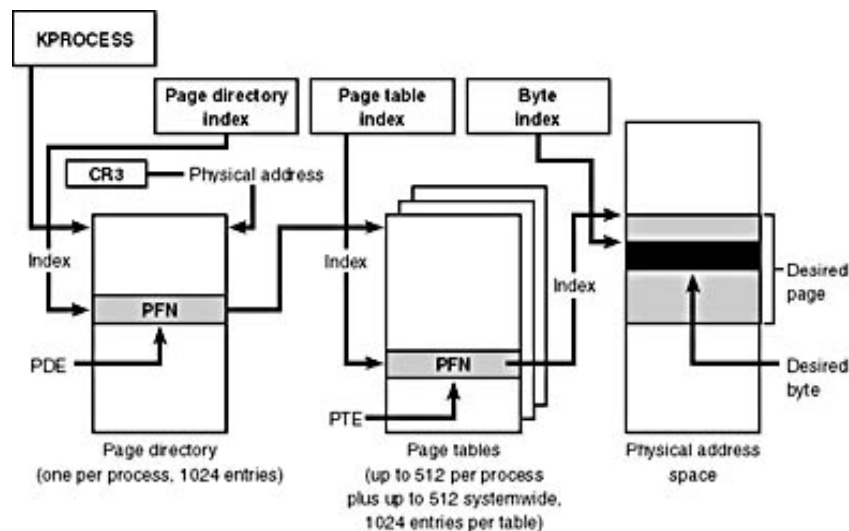
Cả hai vùng bộ nhớ đều được định vị trong phân hệ thống của không gian địa chỉ và được ánh xạ vào không gian địa chỉ ảo của mỗi tiến trình. Trình **Executive** cung cấp các thường trình để định vị và giải phóng từ các vùng này.

### III.8.4. Chuyển đổi địa chỉ

#### ➤ Sơ đồ chuyển đổi một địa chỉ ảo thành địa chỉ vật lý

Theo mặc định hệ điều hành Windows 2000 chạy trên hệ thống x86 sử dụng cấu trúc bảng trang 2 cấp (two-level) để chuyển đổi địa chỉ ảo thành địa chỉ vật lý. 32 bit không gian địa chỉ ảo được chia thành 3 thành phần: 10 bit cao nhất là *Page Directory Index*, 10 bit tiếp theo là *Page Table Index*, 12 bit thấp nhất là *Byte Index* (Byte Index rộng 12 bit vì trong x86 kích thước 1 page là 4096 byte ( $2^{12} = 4096$ )).

Hình vẽ 3.23 sau đây cho thấy ý nghĩa sử dụng của ba thành phần trên và cách chuyển đổi từ địa chỉ ảo 32 bit thành địa chỉ vật lý trên hệ thống x86\_Windows 2000 có thể xem lại ở mục III.2.3.c ở trên).



**Hình 3.23:** Sơ đồ chuyển địa chỉ ảo thành vật lý trên hệ thống x86

Sau đây là các bước thực hiện việc chuyển đổi địa chỉ ảo theo sơ đồ ở trên:

1. Bộ phận phần cứng quản lý bộ nhớ tìm đến danh mục bảng trang (page directory) của tiến trình hiện tại.
2. Thành phần *Page Directory Index* được sử dụng để chỉ mục vào page directory để tìm một mục vào danh mục bảng trang (PDE: page directory entry), mà nó mô tả vị trí của bảng trang (page table) cần để ánh xạ địa chỉ ảo. PDE chứa số hiệu khung trang (PFN: page frame number) của bảng trang (nếu nó đang thường trú trong bộ nhớ. Vì các bảng trang có thể được

phân trang ra ngoài).

3. Thành phần *Page Table Index* được sử dụng để chỉ mục vào page table để tìm một mục vào bảng trang (PTE: page table entry), mà nó mô tả vị trí vật lý của trang ảo trong địa chỉ ảo.
4. PTE được sử dụng để định vị trang. Nếu là trang hợp lệ, nó chứa PFN của trang trong bộ nhớ vật lý chứa trang ảo. Nếu PTE chỉ báo rằng trang là không hợp lệ, trình quản lý bộ nhớ sẽ điều khiển lỗi trang và cố gắng làm cho nó trở thành hợp lệ.
5. Khi PTE trỏ đến một trang hợp lệ, *Byte Index* được sử dụng để tìm đến địa chỉ chính xác của ô nhớ trong phạm vi trang vật lý tương ứng với địa chỉ ảo 32 bit ban đầu mà tiến trình phát ra.

Sau đây chúng ta sẽ xem xét một cách chi tiết hơn về cấu trúc của **page directory**, **page table** và **page table entry** để thấy được đặc thù của nó trong Windows 2000 so với những gì mà ta đã khảo sát một cách tổng quát về nó ở các mục trước:

#### ➤ **Danh mục bảng trang (page directory)**

Mỗi tiến trình có một có một page directory đơn, trình quản lý bộ nhớ dùng một trang để tạo bản đồ định vị của tất cả các bảng trang của tiến trình đó. Địa chỉ vật lý của page directory của tiến trình được lưu trữ trong block KPROCESS.

Địa chỉ vật lý (cơ sở) của page directory được chỉ ra ở thanh ghi điều khiển CR3 trên các hệ thống x86. Mỗi khi có một sự chuyển đổi ngữ cảnh xuất hiện với một tiểu trình nằm trong tiến trình khác tiến trình hiện tại thì giá trị của thanh ghi CR3 này sẽ được nạp vào block KPROCESS của tiến trình khác đó. Việc chuyển đổi ngữ cảnh giữa các tiểu trình trong cùng một tiến trình sẽ không được nạp lại địa chỉ vật lý của page directory bởi vì tất cả các tiểu trình trong cùng một tiến trình chia sẻ cùng một không gian địa chỉ tiến trình.

Page directory bao gồm các các mục vào danh mục bảng trang (PDE: page Directory Entry). Mỗi entry dài 4 byte (8 byte trên các hệ thống chạy ở chế độ PAE), để mô tả trạng thái và vị trí của tất cả các bảng trang của tiến trình. Các bit trong PDE tương tự như các bit của PTE.

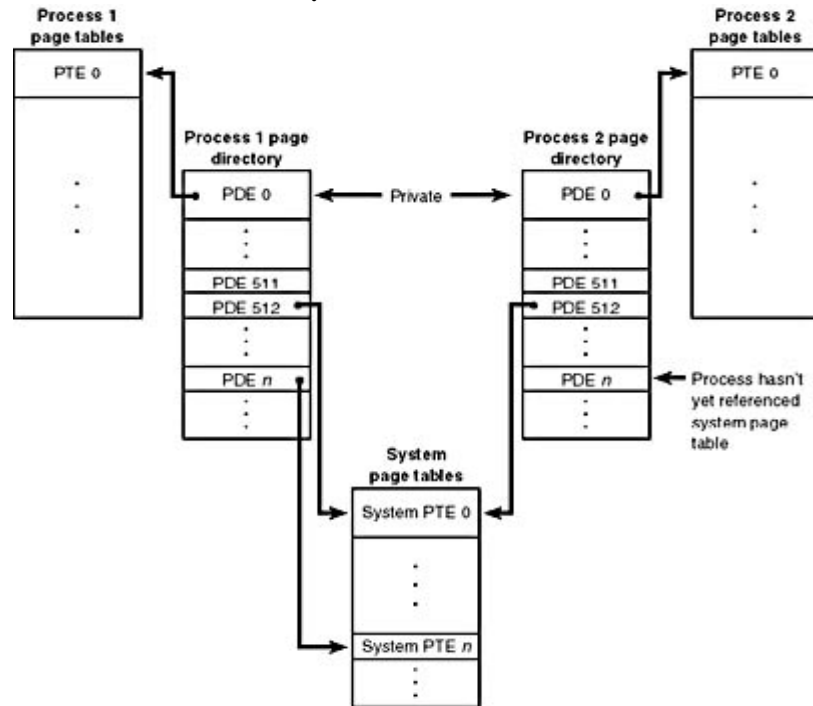
Trên các hệ thống x86, 1024 (2048 trên hệ thống PAE) được yêu cầu để mô tả đầy đủ 4Gb không gian địa chỉ ảo. Page directory tiến trình ánh xạ đến các bảng trang chứa 1024 PDE. Do đó, page directory index cần phải rộng 10 bit ( $2^{10} = 1024$ ).

#### ➤ **Bảng trang tiến trình và bảng trang hệ thống**

Trước khi tham chiếu đến byte trong phạm vi trang bằng *byte offset*, đầu tiên CPU cần phải tìm đến trang mà nó chứa byte yêu cầu của dữ liệu. Để tìm đến trang này, hệ điều hành xây dựng một trang khác của bộ nhớ, trang này chứa các thông tin ánh xạ cần thiết để tìm đến trang mong muốn chứa dữ liệu. Trang thông tin ánh xạ này được gọi là *page table*. Vì Windows 2000 cung cấp một không gian địa chỉ riêng cho mỗi tiến trình nên mỗi tiến trình sở hữu một tập các bảng trang tiến trình để

ánh xạ đến không gian địa chỉ riêng đó, sự ánh xạ sẽ khác nhau ở mỗi tiến trình.

Các bảng trang mô tả không gian hệ thống được chia sẻ cho tất cả các tiến trình. Khi một tiến trình được tạo, các PDE không gian hệ thống được khởi tạo để chỉ đến các bảng trang hệ thống đang tồn tại. Nhưng không phải tất cả các tiến trình đều có cùng điểm nhìn của không gian hệ thống. Khi bảng trang hệ thống thay đổi đến vị trí cấp phát mới thì bộ phận quản lý bộ nhớ không cập nhật tất cả các page directory tiến trình, trình quản lý bộ nhớ chỉ cập nhật các page directory tiến trình khi các tiến trình tham chiếu đến địa chỉ ảo mới.



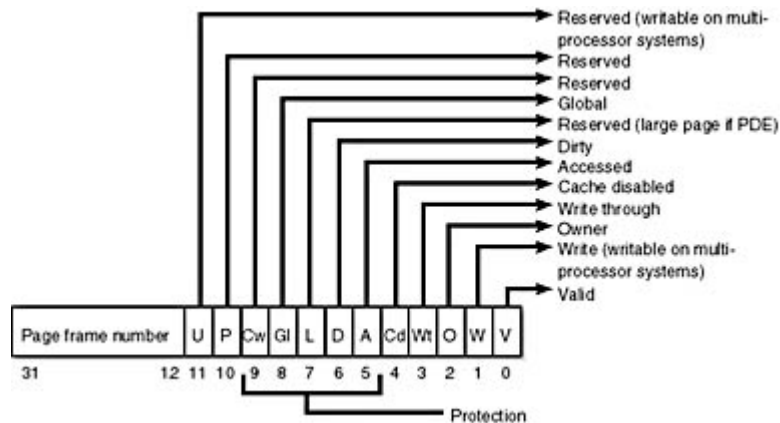
**Hình 3.26:** Bảng trang hệ thống và bảng trang riêng của tiến trình

Số lượng PTE được Windows 2000 tính toán dựa vào kích thước của bộ nhớ. Ta cũng có thể quy định số lượng này bằng cách thay đổi trong Registry, nhưng giá trị lớn nhất mà hệ thống x86 chấp nhận là 128.000 PTE.

### ➤ Các mục vào bảng trang (PTE)

Một PTE dài 32 bit, gồm 13 trường được mô tả ở hình dưới đây:

Sau đây chúng ta sẽ mô tả về các bit trạng thái và các bit bảo vệ trong PTE:



- **Accessed:** Trang đã được đọc.
- **Cache disabled:** Cấm cache trang này
- **Dirty:** Trang đã được ghi đến
- **Global:** Sự chuyển đổi áp dụng đến tất cả các tiến trình.
- **Large page:** Chỉ báo rằng PDE ánh xạ đến trang 4Mb trên hệ thống với 128Mb (hoặc hơn) bộ nhớ.
- **Owner:** Chỉ báo rằng có hay không code user-mode các thẻ truy cập trang hoặc có hay không trang là được giới hạn chỉ truy cập ở kernel-mode.
- **Valid:** Chỉ báo có hay không sự chuyển đổi ánh xạ đến trang trong bộ nhớ vật lý.
- **Write through:** Cấm cache cho việc ghi đến trang với mục đích sự thay đổi ngay lập tức được ghi đến đĩa.
- **Write:** Trên các hệ thống uniprocessor, đây là chỉ báo có hay không trang là read/write hoặc read-only. Trên các hệ thống multiprocessor, đây là chỉ báo có hay không trang là có thể write. (bit Write được lưu trữ trong bit dự trữ trong PTE).

Trên các hệ thống x86, phần cứng PTE chứa một bit Dirty và một bit Accessed. Bit Accessed bị xóa (= 0) nếu trang vật lý được trình bày bởi một PTE không thể đọc hoặc ghi, Processor thiết lập bit (= 1) này khi trang được đọc hoặc ghi lần đầu tiên. Processor thiết lập bit Dirty chỉ khi trang lần đầu tiên được ghi. Kiến trúc x86 cũng thêm vào bit Write để cung cấp sự bảo vệ trang, khi bit này bị xóa thì trang trở thành read-only, khi bit này được thiết lập thì trang có thể là write/read. Nếu một tiểu trình cố gắng ghi đến một trang mà bit Write = 0 thì trình quản lý bộ nhớ sẽ phát sinh một ngoại lệ truy cập, và bộ phận điều khiển lỗi truy cập phải xác định có hay không một tiểu trình có thể ghi đến trang (trong trường hợp copy-on-write) hoặc có hay không một sự vi phạm truy cập phải được sinh ra.

Trên nền phần cứng x86, các PTE luôn luôn rộng 4 byte (32 bit), 8 byte trên các hệ thống cho phép chế độ PAE, vì thế mỗi bảng trang chứa 1024 PTE, 512 trên các hệ thống PAE (4096 byte trên một page, 4 byte trên một PTE) và vì thế có thể ánh xạ 1024 trang (512 page PAE) cho tổng số 4Mb (2 Mb trên PAE) của các trang

dữ liệu.

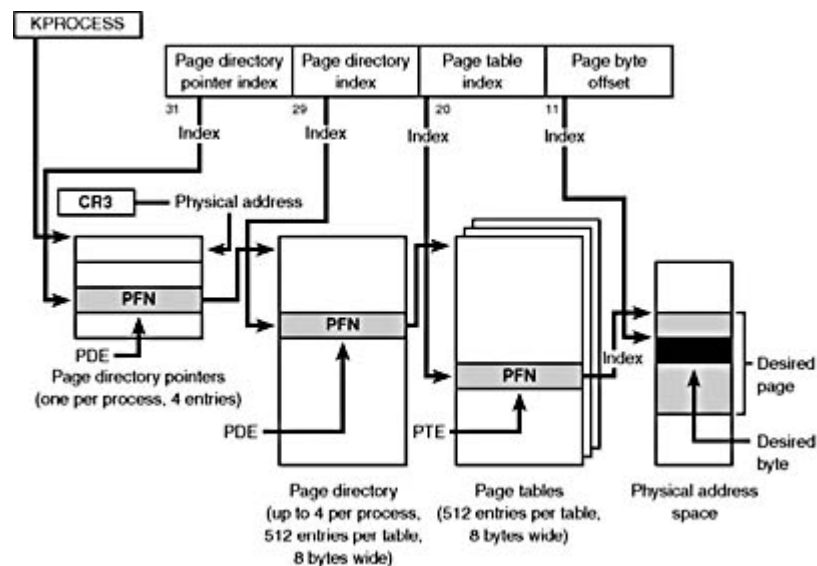
Trường *page table index* của địa chỉ ảo chỉ đến một PTE trong page table, để từ đó ánh xạ đến trang dữ liệu mà tiến trình yêu cầu. Trên các hệ thống x86, *page table index* rộng 10 bit (9 bit trên PAE), cho phép tham chiếu đến 1024 PTE (512 trên PAE). Tuy nhiên, vì windows 2000 cung cấp 4Gb không gian địa chỉ ảo riêng, nên cần nhiều hơn một page table để ánh xạ toàn bộ không gian địa chỉ. Ta có thể tính được số lượng page table được yêu cầu để ánh xạ toàn bộ không gian địa chỉ 4Gb của tiến trình như sau:  $4\text{Gb}/4\text{Mb} = 1024$  page table, hoặc  $2048$  page table  $4\text{Gb}/2\text{Mb} = 1028$  page table trên PAE (mỗi bảng trang trên hệ thống x86 ánh xạ 4Mb (2 Mb trên PAE) của các trang dữ liệu).

### ➤ Byte trong phạm vi trang (byte within page)

Mỗi khi trình quản lý bộ nhớ tìm thấy trang vật lý tương ứng với địa chỉ ảo mà tiến trình đưa ra để truy xuất dữ liệu trên bộ nhớ, nó phải tìm đến đúng dữ liệu được yêu cầu trong phạm vi trang này. Đây là nơi thành phần Byte Index chỉ vào. Byte Index chỉ cho CPU biết byte dữ liệu trong trang mà tiến trình muốn tham chiếu đến. Trên hệ thống x86, byte index rộng 12 bit, cho phép tiến trình tham chiếu đến 4096 byte dữ liệu (đây cũng chính là kích thước trang).

### ➤ Mở rộng địa chỉ vật lý

Tất cả các processor thuộc họ Intel x86 đều bao gồm một chế độ ánh xạ bộ nhớ được gọi là PAE (Physical Address Extension). Với một chipset thích hợp chế độ PAE cho phép truy cập đến 64GB bộ nhớ vật lý. Khi thực thi x86 trong chế độ PAE, thành phần quản lý bộ nhớ (MMU) của processor chia địa chỉ ảo thành 4 thành phần. Trong trường hợp này hệ thống sử dụng bảng trang ba cấp (three-level) để thực hiện việc chuyển đổi địa chỉ.



Hình 3.24: Ánh xạ trang với PAE

MMU vẫn cài đặt page directory và page table nhưng cấp thứ 3 là page directory pointer table. PAE có thể đánh địa chỉ bộ nhớ nhiều hơn chế độ chuẩn không những là do mở rộng cấp bảng trang mà còn do các PDE và PTE rộng 64 bit chứ không phải 32 bit. Với địa chỉ vật lý bên trong là 24 bit, nên x86 có khả năng quản lý được 64Gb ( $2^{24+12}$  byte) bộ nhớ.

Để chọn Windows 2000 hoạt động trong chế độ PAE ta phải chọn boot với tham số khoá chuyển /PAE trong Boot.ini. Chế độ này được hỗ trợ trong tập tin Ntkrpamp.exe.

## Chương IV

# QUẢN LÝ TẬP TIN VÀ ĐĨA

---

*Tất cả các ứng dụng trên máy tính đều cần lưu trữ và đọc lại thông tin mà nó nhận vào và xử lý. Trong khi một tiến trình đang chạy nó có thể lưu trữ một lượng giới hạn thông tin trong phạm vi không gian địa chỉ sở hữu của nó. Tuy nhiên khả năng lưu trữ này bị giới hạn bởi kích thước không gian địa chỉ ảo của hệ thống. Đối với một vài ứng dụng thì không gian này là vừa đủ, nhưng đối với một số ứng dụng khác thì nó là quá nhỏ. Mặt khác nếu lưu giữ thông tin trong không gian địa chỉ của tiến trình thì thông tin này sẽ bị mất khi tiến trình kết thúc. Vấn đề thứ ba là phải đáp ứng việc truy cập thông tin đồng thời giữa các tiến trình trong môi trường hệ điều hành đa nhiệm. Những vấn đề trên chúng ta đã biết trong các chương Quản lý tiến trình và Quản lý bộ nhớ của tài liệu này. Để giải quyết những vấn đề trên hệ điều hành phải thiết kế một hệ thống lưu trữ thông tin sao cho: Thứ nhất là phải lưu trữ được một khối lượng lớn thông tin. Thứ hai là thông tin phải được bảo toàn khi tiến trình sử dụng nó kết thúc. Và cuối cùng là có thể có nhiều tiến trình truy xuất thông tin đồng thời.*

*Giải pháp cho tất cả vấn đề trên là lưu trữ thông tin trên đĩa và các thiết bị media khác trong các đơn vị dữ liệu, được gọi là các file (tập tin). Các tiến trình có thể đọc thông tin của file và rồi ghi mới thông tin vào file nếu cần thiết. Thông tin được lưu trữ trong file phải không bị tác động bởi việc tạo và kết thúc tiến trình.*

*Các file được quản lý bởi hệ điều hành. Thành phần hệ điều hành tham gia trực tiếp vào quá trình quản lý các file trên đĩa được gọi là hệ thống file. Hệ điều hành phải xây dựng cấu trúc và tổ chức*



*hoạt động của hệ thống file. Một trong những nhiệm vụ quan trọng của hệ thống file là theo dõi việc lưu trữ file trên đĩa, theo dõi và điều hành việc truy cập file của các tiến trình, bảo vệ file và nội dung của file, ... Cấu trúc, tổ chức hoạt động và những nhiệm vụ của hệ thống file của hệ điều hành, của các hệ điều hành cụ thể, sẽ được chúng ta xem xét trong chương này.*

## **Tổng quan về quản lý tập tin và đĩa**

### **IV.1.1. Tập tin và hệ thống quản lý tập tin**

**Tập tin (File):** Tập tin là đơn vị logic được lưu trữ và xử lý bởi thành phần quản lý tập tin của hệ điều hành. Hệ điều hành cung cấp các công cụ để người sử dụng và chương trình của người sử dụng có thể lưu trữ tập tin trên thiết bị lưu trữ (đĩa và các thiết bị media khác) và có thể đọc lại tập tin này nhanh nhất. Mỗi tập tin được hệ điều hành tạo ra một sự tương ứng với một tên cụ thể nào đó, tên tập tin là một khái niệm trừu tượng, nó tạo ra sự đồng nhất giữa tập tin với các thiết bị lưu trữ khác nhau. Nhờ đó, mà người sử dụng dễ dàng truy xuất tập tin thông qua tên của nó. Đa số các hệ điều hành đều cho phép tên tập tin là một dãy kí tự ASCII hoặc Unicode.

Nội dung của tập tin có thể là một chương trình, một tập các thủ tục hoặc một khối dữ liệu. Nó có thể là một dãy tuần tự các byte không cấu trúc, hệ điều hành không biết nội dung của tập tin. Một dãy các record có chiều dài cố định. Hay là một cấu trúc cây, gồm cây của những record có thể không có cùng độ dài, mỗi record có một trường khoá để giúp cho việc tìm kiếm nó được nhanh hơn.

Các hệ điều hành hỗ trợ nhiều kiểu tập tin khác nhau như: tập tin thường, tập tin thư mục, tập tin có ký tự đặc biệt, tập tin khối. Tập tin thường là tập tin text hay tập tin nhị phân chứa thông tin của người sử dụng. Tập tin thư mục là những tập tin hệ thống dùng để lưu giữ cấu trúc của hệ thống tập tin. Tập tin có ký tự đặc biệt, liên quan đến nhập xuất thông qua các thiết bị nhập xuất tuần tự như màn hình, máy in, mạng. Tập tin khối dùng để truy xuất trên các thiết bị lưu trữ khối (đĩa là thiết bị lưu trữ khối).

Thiết bị lưu trữ tập tin thường được chia thành các block có kích thước cố định bằng nhau, các block được đánh địa chỉ để phân biệt. Thành phần quản lý tập tin của hệ điều hành có nhiệm vụ cấp phát và thu hồi các block cho các tập tin khi cần thiết. Vì kích thước tập tin có thể thay đổi, nên các hệ điều hành thường tổ chức cấp phát động các block cho các tập tin. Hệ điều hành có thể tổ chức cấp phát tĩnh block cho các tập tin có kích thước không thay đổi như các tập tin thực thi, các tập tin thư viện, ... Cấp phát tĩnh sẽ nhanh và đơn giản hơn nhiều so với cấp phát động.

Các hệ điều hành cho phép truy xuất tập tin theo 2 cách tuần tự và ngẫu nhiên. Trong các hệ thống truy xuất tuần tự, các tiến trình có thể đọc tất cả các byte

hoặc các record trong tập tin, theo thứ tự, từ một vị trí bắt đầu nào đó mà không thể bỏ qua một byte hay một record nào. Truy cập ngẫu nhiên thì ngược lại, các tiến trình có thể truy xuất tại bất kỳ một byte hay một record nào đó trong file. Trong cả hai cách trên đều phải chỉ ra vị trí bắt đầu đọc. Trong cách thứ nhất, mỗi thao tác đọc cần phải xác định vị trí bắt đầu đọc trong file. Trong cách thứ 2, trước khi đọc hệ thống phải tìm đến (SEEK) vị trí bắt đầu đọc, sau đó tiến hành đọc tuần tự như cách thứ nhất.

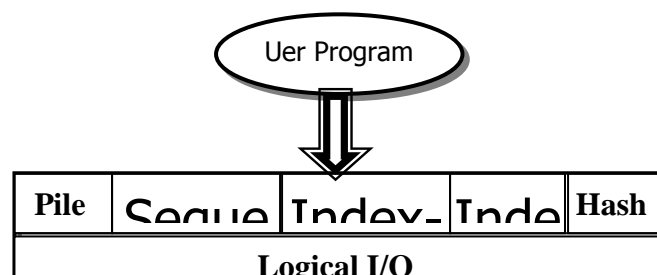
**Hệ thống quản lý tập tin (File management System):** Hệ thống quản lý tập tin, hay gọi ngắn gọn là hệ thống tập tin, là một tập các dịch vụ mà hệ điều hành cung cấp cho người sử dụng và chương trình người sử dụng để các đối tượng này sử dụng các tập tin trên hệ thống. Người sử dụng và chương trình của người sử dụng chỉ có thể truy xuất các tập tin thông qua hệ thống tập tin. Hệ thống quản lý tập tin của hệ điều hành phải đáp ứng các mục tiêu cơ bản sau đây:

- Đáp ứng các yêu cầu về lưu trữ dữ liệu của người sử dụng, bao gồm: khả năng lưu trữ, độ tin cậy và hiệu suất.
- Cự tiểu hay loại bỏ các nguy cơ có thể dẫn đến hỏng hoặc mất dữ liệu.
- Cung cấp sự hỗ trợ vào/ra cho nhiều loại thiết bị lưu trữ khác nhau.
- Cung cấp sự hỗ trợ vào/ra cho nhiều người sử dụng trong các hệ thống đa người sử dụng.
- Cung cấp một tập chuẩn các thủ tục giao diện vào/ra.

Đối với người sử dụng thì hệ thống quản lý tập tin của một hệ điều hành phải đáp ứng các yêu cầu tối thiểu sau đây:

- Mỗi người sử dụng phải có thể tạo (create), xóa (delete) và thay đổi (change) các tập tin.
- Mỗi người sử dụng có thể được điều khiển để truy cập đến các tập tin của người sử dụng khác.
- Mỗi người sử dụng phải có thể di chuyển dữ liệu giữa các tập tin.
- Mỗi người sử dụng phải có thể truy cập đến các tập tin của họ thông qua tên tượng trưng của tập tin.
- Mỗi người sử dụng phải có thể dự phòng và khôi phục lại các tập tin của họ trong trường hợp hệ thống bị hỏng.

**Kiến trúc hệ thống tập tin (File System Architecture):** Các hệ điều hành khác nhau có cách tổ chức hay kiến trúc của hệ thống tập tin khác nhau. Hình vẽ sau đây trình bày một kiến trúc hệ thống tập tin chung nhất mà các hệ điều hành thường sử dụng.



- Cấp thấp nhất trong kiến trúc này là các điều khiển thiết bị (device driver) truyền thông trực tiếp với các thiết bị ngoại vi. Device driver chịu trách nhiệm khởi tạo một thao tác vào/ra trên thiết bị và xử lý các yêu cầu vào/ra. Các device driver trong hệ thống tập tin thường là các điều khiển đĩa.

- Cấp kế trên device driver, được xem như là hệ thống tập tin cơ sở (basic file system), hoặc cấp vào/ra vật lý, đó là giao diện chính giữa môi trường bên ngoài với hệ thống máy tính. Nó giao tiếp với các block dữ liệu trao đổi giữa các đĩa với hệ thống. Vì thế nó được kết nối với các block trên đĩa và các buffer trên bộ nhớ chính. Nó không hiểu các dữ liệu cũng như các cấu trúc file phức tạp.

- Cấp basic I/O supervisor chịu trách nhiệm khởi tạo và kết thúc tất cả các thao tác vào/ra tập tin. Tại cấp này, các cấu trúc điều khiển được duy trì, các cấu trúc điều khiển này giao tiếp với thiết bị vào/ra, bộ phận lập lịch đọc đĩa và bộ phận quản lý trạng thái tập tin. Basic I/O supervisor kết hợp với các bộ phận lập lịch đọc đĩa để tối ưu các thao tác đọc đĩa, nhằm góp phần tăng tốc độ truy xuất tập tin của các chương trình người sử dụng.

- Cấp vào/ra logic (Logical I/O) là thành phần quan trọng của hệ thống tập tin, nó cho phép người sử dụng và chương trình người sử dụng truy cập đến các record. Trong khi hệ thống tập tin cơ sở giao tiếp với các block dữ liệu, thì logical I/O giao tiếp với các record file. Logical I/O cung cấp các công cụ chung nhất để thực hiện các thao tác vào/ra file dựa trên record.

- Cấp trên cùng của kiến trúc hệ thống tập tin kết hợp chặt chẽ với người sử dụng. Nó cung cấp một giao diện chuẩn giữa chương trình người sử dụng, hệ thống tập tin và thiết bị lưu trữ dữ liệu. Các phương pháp truy cập dữ liệu khác nhau phản ánh các cấu trúc tập tin khác nhau và các cách khác nhau để truy cập và xử lý dữ liệu. Các phương pháp truy cập đó là: Pile, Sequential file, Indexed-sequential file, Indexed file, Hashed, vv. Xem cụ thể ở [6].

#### IV.1.2. Bảng danh mục và tập tin chia sẻ

**Bảng danh mục (Directory Table):** Các hệ điều hành phải tổ chức bảng danh mục, để lưu trữ các thông tin liên quan đến các tập tin và các thư mục đang tồn tại trên đĩa (hoặc thiết bị lưu trữ khác), đặc biệt là thông tin cho biết vị trí lưu trữ nội dung của một tập tin trên đĩa. Để truy xuất đến một tập tin hệ điều hành cần phải thông qua bảng danh mục này.

Bảng danh mục gồm nhiều entry (phần tử/mục vào), mỗi phần tử dùng để chứa thông tin của một tập tin hay thư mục trên đĩa. Khi có một tập tin/ thư mục được tạo ra thì hệ điều hành sẽ dùng một phần tử trong bảng danh mục để chứa các thông tin của nó. Khi một tập tin/ thư mục bị xóa khỏi đĩa thì hệ điều hành sẽ giải phóng phần tử của nó trong bảng danh mục. Có thể xem một phần tử trong bảng danh mục là một sự tương ứng giữa tập tin và vị trí lưu trữ của tập tin trên đĩa.

Số lượng phần tử trong bảng danh mục có thể bị giới hạn cố định trước hoặc không có giới hạn và có thể tăng/ giảm nếu cần. Bảng danh mục có thể được chứa tại một không gian đặc biệt nào đó trên đĩa, hoặc có thể chứa trong một file metadata nào đó trên đĩa. Trong quá trình hoạt động của hệ thống bảng danh mục thường được hệ điều hành nạp từ đĩa vào bộ nhớ, để sẵn sàng cho việc truy xuất file của hệ điều hành sau này.

Một phần tử trong danh mục phải chứa các thông tin tối thiểu sau đây: Tên của tập tin; Kiểu của tập tin; Địa chỉ vật lý của tập tin trên đĩa. Các thông tin kiểm tra truy nhập tập tin; Các thông tin quản trị tập tin; vv.

Các hệ điều hành thường thiết kế và sử dụng bảng danh mục hai mức. Mức 1, được gọi là bảng danh mục chủ, bao gồm các con trỏ trỏ tới bảng danh mục người sử dụng. Mức 2, được gọi là bảng danh mục người sử dụng, bao gồm tên tập tin và địa chỉ vật lý của tập tin trên đĩa,... Tổ chức bảng thư mục gốc và bảng thư mục con là sự cài đặt cụ thể cấu trúc bảng danh mục hai mức của hệ điều hành MS\_DOS. Muốn truy xuất đến tập tin thì người sử dụng và chương trình của người sử dụng phải thông qua danh mục chủ và danh mục người sử dụng hay thông qua thư mục gốc và thư mục con trong hệ điều hành MS\_DOS.

Để thực hiện bất kỳ một thao tác nào trên nội dung của tập tin thì trước hết tập tin phải được mở. Khi nhận được yêu cầu mở tập tin thì hệ điều hành sử dụng đường dẫn được chỉ ra bởi người sử dụng hay chương trình của người sử dụng để tìm đến một mục vào tương ứng với tập tin cần mở trong bảng danh mục. Phần tử trong bảng danh mục sẽ cung cấp các thông tin cần thiết để hệ điều hành tìm đến các block đĩa chứa nội dung của tập tin. Tùy vào từng hệ điều hành mà thông tin này có thể là địa chỉ của tất cả block đĩa chứa nội dung tập tin (trong chiến lược cấp phát liên tục), địa chỉ của block đĩa đầu tiên chứa nội dung tập tin (trong chiến lược danh sách liên kết và danh sách liên kết chỉ mục), hoặc số hiệu của I-node (trong

chiến lược I-node). Các chiến lược này được trình bày trong phần *quản lý các block chứa file trên đĩa* ngay sau đây.

Tổ chức bảng thư mục gốc của MS\_DOS, windows98 và MFT của windowsNT/2000 là các sự cài đặt cụ thể về cấu trúc của bảng danh mục của các hệ điều hành. Tổ chức của bảng thư mục gốc của MS\_DOS, windows98, windowsNT/2000 sẽ được xem xét ở phần sau của chương này.

**Tập tin chia sẻ (Shared File):** Tập tin chia sẻ xuất hiện trong các môi trường nhiều người sử dụng, đây là một kỹ thuật của hệ điều hành, nhằm giúp nhiều người sử dụng trên hệ thống có thể cùng nhau sử dụng một tập tin nào đó. Đối với người sử dụng, tập tin chia sẻ là tập tin được xuất hiện đồng thời trong các thư mục khác nhau của các người sử dụng khác nhau.

Kỹ thuật chia sẻ tập tin thường được các hệ điều hành sử dụng nhất là, cho phép các phần tử trong các bảng danh mục người sử dụng khác nhau chứa thông tin của cùng một tập tin chia sẻ nào đó, đặc biệt là thông tin về địa chỉ của các block đĩa chứa nội dung của tập tin chia sẻ. Khi có một liên kết chia sẻ mới được thiết lập đến một người sử dụng nào đó, hệ điều hành chỉ cần sao chép danh sách các block đĩa của file chia sẻ đến phần tử tương ứng trong bảng danh mục người sử dụng của người sử dụng đó. Kỹ thuật này đơn giản dễ cài đặt nhưng cũng xuất hiện vấn đề: *nếu tập tin được cập nhật bởi một người sử dụng nào đó thì sự cập nhật này sẽ không được nhìn thấy bởi các người sử dụng khác (điều này sẽ vi phạm mục đích của việc chia sẻ tập tin)*. Vì khi tập tin được cập nhật thì hệ điều hành phải cung cấp thêm một vài block đĩa cho nó, địa chỉ của các block đĩa mới này chỉ được liệt kê thêm trong phần tử tương ứng trong bảng danh mục của người sử dụng thực hiện sự cập nhật tập tin mà không được liệt kê trong các bảng danh mục của người sử dụng khác.

Vấn đề trên có thể được giải quyết như sau: danh sách địa chỉ các block đĩa chứa tập tin chia sẻ không được liệt kê trong phần tử bảng danh mục, mà được chứa trong một khối dữ liệu có cấu trúc nào đó, tạm gọi là khối dữ liệu mô tả lưu trữ tập tin hay nói gọn hơn là khối mô tả lưu trữ. Khối mô tả lưu trữ này có thể được gắn vào chính tập tin chia sẻ nếu kích thước nhỏ, hoặc được đặt ở một vị trí nào đó trên đĩa, nếu kích thước lớn (trường hợp này có thể dùng chung cho nhiều tập tin chia sẻ). Mọi sự thay đổi về danh sách địa chỉ các block đĩa chứa tập tin chia sẻ đều được phản ánh ở khối mô tả lưu trữ của nó. Các phần tử trong bảng danh mục bây giờ chỉ đóng vai trò như một con trỏ trỏ đến khối mô tả lưu trữ của các tập tin chia sẻ, nhờ vậy mà một sự thay đổi tập tin chia sẻ từ bất kỳ một người sử dụng nào trong số những người sử dụng được chia sẻ tập tin đều được nhìn thấy từ tất cả những người sử dụng còn lại.

Trong môi trường nhiều người sử dụng, việc chia sẻ một tập tin cho nhiều người sử dụng là rất cần thiết và nó đã mang lại nhiều thuận lợi. Nhưng nó cũng

phát sinh nhiều lỗi trong quá trình sử dụng tập tin chia sẻ giữa nhiều người sử dụng và chương trình người sử dụng, mà nếu hệ điều hành không tổ chức giám sát tốt thì có thể dẫn đến tình trạng hỏng tập tin chia sẻ hoặc nội dung của tập tin chia sẻ. Chúng ta đã biết hệ điều hành giải quyết vấn đề này như thế nào trong chương *Quản lý tiến trình* của tài liệu này. Đây là một vấn đề lớn đối với các hệ điều hành đa nhiệm đặc biệt là các hệ điều hành mạng. Các hệ điều hành này cung cấp đầy đủ các công cụ để người sử dụng và chương trình của người sử dụng kết hợp cùng với hệ điều hành khai thác, sử dụng tốt các tập tin chia sẻ nhưng hạn chế thấp nhất các lỗi có thể xảy ra. Trong phần sau của chương này chúng ta sẽ xem xét những thao tác mà hệ điều hành phải thực hiện để đáp ứng yêu cầu mở file từ người sử dụng trong môi trường nhiều người sử dụng.

#### IV.1.3. Quản lý không gian đĩa

**Kích thước block:** Để tổ chức lưu trữ nội dung các file trên đĩa, các hệ điều hành đều chia không gian lưu trữ của đĩa thành các phần có kích thước bằng nhau được gọi là khối (block) lưu trữ. Nội dung của file cũng được chia thành các block có kích thước bằng nhau, trừ block cuối cùng, và bằng với kích thước các block đĩa. Khi cần lưu trữ file trên đĩa hệ điều hành cấp cho mỗi tập tin một số lượng block vừa đủ để chứa hết nội dung của tập tin. Kích thước của một block phụ thuộc vào qui định của vi xử lý và hệ điều hành, thường là 128 byte, 256 byte, hoặc 512 byte, vv.

Khi chọn kích thước của block hệ điều hành phải xem xét các vấn đề sau:

- Nếu kích thước block lớn thì dễ lãng phí đĩa, trong trường hợp kích thước của tập tin không phải là bội số của kích thước block.
- Nếu kích thước block nhỏ thì đĩa được chia thành nhiều block, dẫn đến kích thước danh sách quản lý block của đĩa, danh sách quản lý block của một tập tin, bảng các block, vv, sẽ tăng lên do đó dung lượng bộ nhớ chứa nó sẽ tăng lên.
- Kích thước của block phải là bội của kích thước khối dữ liệu mà hệ thống dùng khi thực hiện truyền dữ liệu giữa bộ nhớ chính và bộ nhớ phụ.

**Theo dõi các block tự do:** Khi cần lưu trữ nội dung của các file lên đĩa, hệ điều hành cấp cho file một số lượng block đĩa nhất định để chứa hết nội dung của nó, các block đĩa này có thể nằm tại các vị trí bất kỳ trên đĩa. Trong quá trình sử dụng file kích thước của file có thể thay đổi, tăng lên hay giảm xuống, do đó hệ điều hành phải tổ chức cấp phát động các block đĩa cho các file. Khi kích thước của file tăng lên thì hệ điều hành phải cấp phát thêm block cho nó, khi kích thước file giảm xuống hoặc khi file bị xoá khỏi đĩa thì hệ điều hành phải thu hồi lại các block đĩa đã cấp cho nó để có thể cấp cho các file khác sau này.

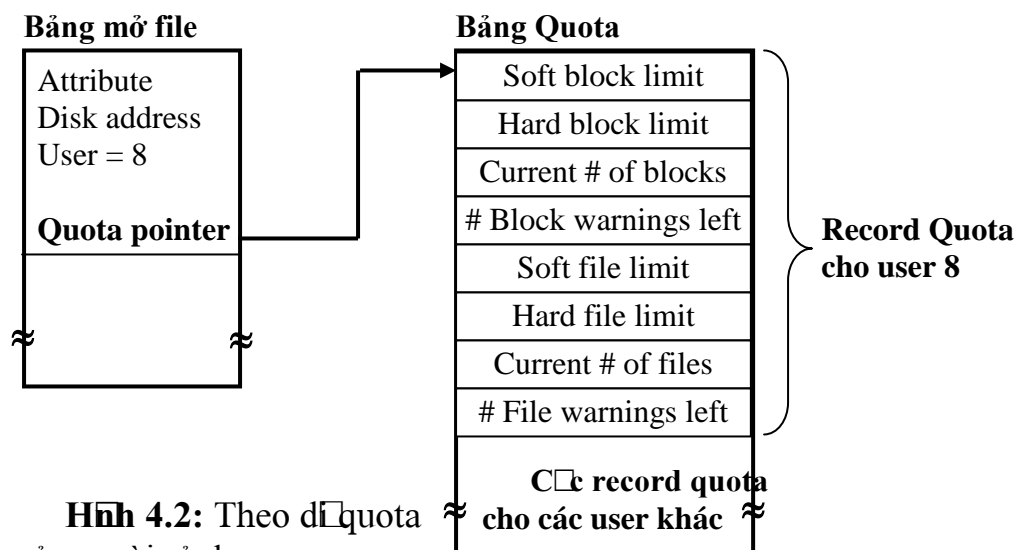
Để tổ chức cấp phát động các block đĩa cho file hệ điều hành phải quản

lý được trạng thái của các block, còn tự do hay đã cấp phát, trên đĩa. Trong trường hợp này các hệ điều hành có thể sử dụng 2 kỹ thuật: **Dùng bảng bit** và/hoặc **dùng danh sách liên kết**. Trong bảng bit, mỗi bit cho biết trạng thái của một block tương ứng trên bộ nhớ phụ, = 0 thì block tương ứng còn tự do, = 1 thì block tương ứng đã cấp phát cho một file nào đó. Như vậy, để tìm N block tự do hệ điều hành chỉ cần tìm N bit 0 trong bảng bit, do đó tốc độ tìm và cấp phát block cho các file sẽ tăng lên rất nhiều. Trong danh sách liên kết, để quản lý các block còn tự do hệ điều hành dùng một danh sách liên kết. Mỗi phần tử trong danh sách cho biết địa chỉ của một block tự do trên đĩa. Như vậy khi cần cấp phát block cho một file nào đó thì hệ điều hành sẽ dựa vào danh sách các block tự do này.

Sau khi cấp phát hoặc thu hồi block hệ điều hành phải tiến hành cập nhật lại danh sách liên kết hay bảng bit. Trong trường hợp bảng bit hoặc danh sách liên kết lớn, hệ điều hành sẽ chứa nó ở đĩa và chỉ nạp phần cần thiết vào bộ nhớ chính. Khi lựa chọn các block trong tập các block tự do để cấp phát cho một file hệ điều hành phải chọn sao cho việc cấp phát được thực hiện nhanh và việc đọc sau này là tối ưu với một thuật toán đọc đĩa cụ thể nào đó.

**Cấp hạn ngạch đĩa (Disk Quotas):** Để ngăn chặn người sử dụng sử dụng quá nhiều không gian đĩa, các hệ điều hành đa người sử dụng thường cung cấp một chiến lược để người quản trị hệ thống giới hạn số lượng không gian đĩa tối đa (block) mà mỗi người sử dụng được phép sử dụng và hệ điều hành phải đảm bảo rằng người sử dụng không thể sử dụng quá không gian đĩa mà hệ điều hành cấp cho họ, chiến lược này được gọi là cấp hạn ngạch đĩa.

Khi người sử dụng mở file, thì các thuộc tính và các địa chỉ block đĩa mà hệ điều hành cấp cho file được ghi vào bảng mở file trong bộ nhớ chính, trong đó có cả thuộc tính cho biết người sử dụng nào sở hữu file được mở. Bất kỳ một sự thay đổi nào về kích thước file cũng thay đổi đến hạn ngạch của người sử dụng sở hữu file.



**Hình 4.2:** Theo dõi quota của người sử dụng

Một bảng thứ hai chứa record quota, cho mỗi người sử dụng mở file hiện tại, thậm chí nếu file được mở bởi một người nào đó, bảng này được trình bày ở hình sau. Hình 4.2 cho thấy một phần của file quota trên đĩa, cho biết file của người sử dụng nào là đang được mở. Khi tất cả các file đều được đóng, record sẽ ghi trở lại file quota.

Khi có một entry mới được tạo ra trong bảng mở file thì một con trỏ (quota pointer) trỏ tới record quota của người sở hữu file, là được nhập vào nó. Mỗi khi có một block được thêm vào một file thì tổng số block của người sử dụng được tăng lên và một check được gán đến cả Hard block limit và Soft block limit. Soft limit có thể được vượt quá, nhưng hard limit thì không thể. Một sự cố gắng thêm vào cuối file khi hard block limit bị vượt quá giới hạn sẽ trả về thông báo lỗi.

Khi một người sử dụng cố gắng login, hệ thống sẽ kiểm tra file quota để xem người sử dụng đã vượt quá soft limit của block hoặc file hay chưa (soft block limit hoặc soft file limit). Nếu cả hai limit đều bị vi phạm, thì một cảnh báo sẽ xuất hiện, và bộ đếm (count) tương ứng với cảnh báo sẽ giảm xuống một đơn vị. Nếu bộ đếm nhận được giá trị zero thì người sử dụng sẽ không được phép login.

#### IV.1.4. Quản lý các block chứa file trên đĩa

*Trong phần này chúng ta xem xét các phương pháp khác nhau mà các hệ điều hành sử dụng để theo dõi danh sách các block đĩa mà hệ điều hành đã cấp phát cho một file, để chứa hết các block của một file, của tất cả các file đang được lưu trữ trên đĩa.*

➤ **Cấp phát liên tục (contiguous allocation):** là một chiến lược đơn giản nhất, trong chiến lược này các block file được lưu trữ tại các block đĩa liên tục nhau. Tức là, nếu 1 block đĩa là 1K thì một file 50K sẽ được lưu trữ tại 50 block liên tiếp nhau trên đĩa. Chiến lược này đơn giản, dễ cài đặt và thời gian đọc file giảm xuống đáng kể, vì hệ điều hành chỉ cần biết block đĩa đầu tiên chứa các block file và tổng số block đĩa chứa file là có thể tiến hành đọc nội dung của file mà không cần dò tìm danh sách các block đĩa chứa nội dung của file.

Chiến lược này chỉ có thể được sử dụng với các file có kích thước cố định, không thay đổi so với thời điểm tạo ra file, hoặc với các file mà hệ điều hành biết trước được kích thước tối đa của file, trong trường hợp này hệ điều hành phải dự trữ block đĩa cho file, điều này dễ dẫn đến tình trạng lãng phí trong việc sử dụng block đĩa. Chiến lược này có thể dẫn đến hiện tượng phân mảnh trên đĩa, tức là trên đĩa có thể xuất hiện các đoạn block trống nhỏ, không đủ để chứa một file có kích thước tối thiểu, nằm giữa các đoạn block chứa file, các đoạn block trống này có thể

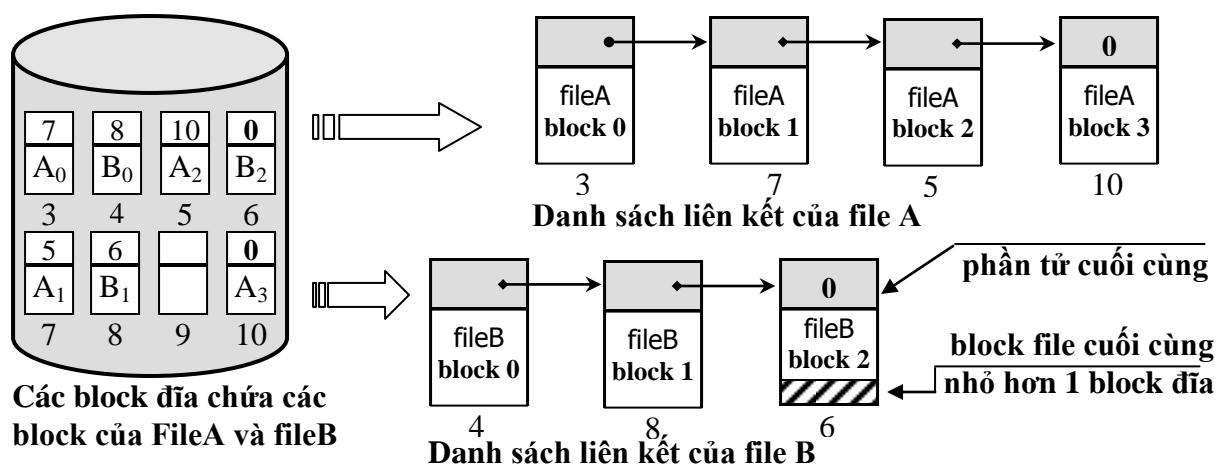


là nơi lưu trữ của một file nào đó mà file này đã bị xoá khỏi đĩa. Hiện tượng phân mảnh đĩa sẽ làm chậm tốc độ đọc file của hệ điều hành.

Các hệ điều hành hiện nay, hệ điều hành windowsNT/2000 chẳng hạn, cải tiến chiến lược này để khắc phục các hạn chế và tận dụng những thuận lợi của nó. Bằng cách, vẫn cấp phát các block đĩa liên tục để chứa vừa đủ kích thước ban đầu của file, và sau đó nếu kích thước của file tăng lên thì hệ điều hành sẽ tìm và cấp phát một đoạn block khác tại một vị trí bất kỳ trên đĩa để chứa vừa đủ phần kích thước tăng lên này. Tức là, nội dung của file được lưu trữ tại các đoạn block đĩa rời rạc nhau trên đĩa. Nếu kích thước file giảm xuống thì hệ điều hành phải tổ chức lại việc lưu trữ file để sao cho có thể giải phóng được một đoạn block đĩa chứa file trước đó. Với việc cải tiến này, hệ điều hành có thể đọc file nhanh hơn, ít xảy ra phân mảnh hơn nhưng việc tổ chức lưu trữ sẽ phức tạp hơn. Chúng ta sẽ thấy cách tổ chức này trong hệ thống file của hệ điều hành windowsNT/2000 trong phần sau của chương này.

➤ **Cấp phát theo danh sách liên kết (linked list allocation):** chiến lược này sử dụng một danh sách liên kết các block đĩa để chứa nội dung của một file. Word đầu tiên của mỗi block đĩa được sử dụng như một con trỏ để trỏ đến block kế tiếp, trừ word của block cuối cùng được sử dụng để chứa tín hiệu báo kết thúc danh sách của một file, phần còn lại của block đĩa dùng để chứa nội dung của file. Trong trường hợp này kích thước của block đĩa phải lớn hơn kích thước của block file 1 word.

Hình sau đây minh họa cho việc lưu trữ file theo chiến lược này, với file A được chia thành 4 block: block 0, block 1, block 2, block 3 được lưu trữ tại các block đĩa, lần lượt là 3, 7, 5, 10. Với file B được chia thành 3 block: block 0, block 1, block 2, được lưu trữ tại các block đĩa, lần lượt là 4, 8, 6.

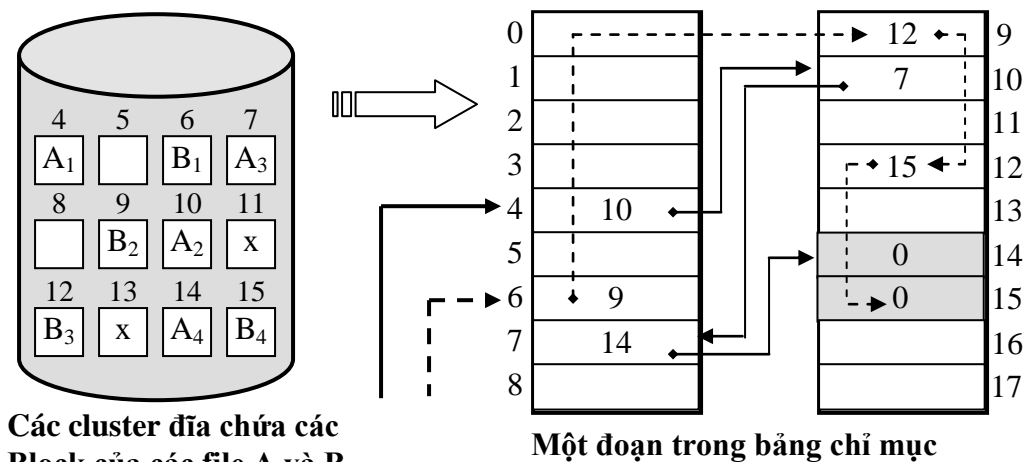


**Hình 4.3:** Cấp phát block theo danh sách liên kết

Không như cấp phát liên tục, mọi block đều có thể được sử dụng trong chiến

lược này, nên sẽ không dẫn đến hiện tượng phân mảnh đĩa và khai thác tối đa không gian đĩa. Và hệ điều hành chỉ cần biết block đĩa đầu tiên chứa file là có thể đọc được toàn bộ nội dung của file, block đầu tiên này được ghi ở phần tử trong bảng danh mục tương ứng với mỗi file. Tốc độ đọc file theo cách truy cập ngẫu nhiên trong chiến lược này sẽ rất chậm so với cách truy cập tuần tự như ở chiến lược cấp phát liên tục ở trên.

➤ **Cấp phát theo danh sách liên kết sử dụng chỉ mục (linked list allocation using an index):** Cấp phát theo danh sách liên kết tồn tại hai hạn chế đó là: chậm và tốn một word để chứa con trỏ đến block kế tiếp. Để khắc phục hai hạn chế này, các hệ điều hành lưu các word con trỏ nói trên vào trong một bảng chỉ mục và nạp bảng chỉ mục này vào bộ nhớ khi hệ điều hành cần đọc nội dung của file trên đĩa.



**Hình 4.4:** Cấp phát block theo danh sách liên kết có chỉ mục

Hình 4.4 minh họa cho việc lưu trữ file theo chiến lược này, với file A được chia thành 4 block: A<sub>1</sub>, A<sub>2</sub>, A<sub>3</sub>, A<sub>4</sub> được lưu trữ tại các block đĩa, lần lượt là 4, 10, 7, **14** (cuối cùng). Với file B được chia thành 4 block: B<sub>1</sub>, B<sub>2</sub>, B<sub>3</sub>, B<sub>4</sub> được lưu trữ tại các block đĩa, lần lượt là 6, 9, 12, **15** (cuối cùng).

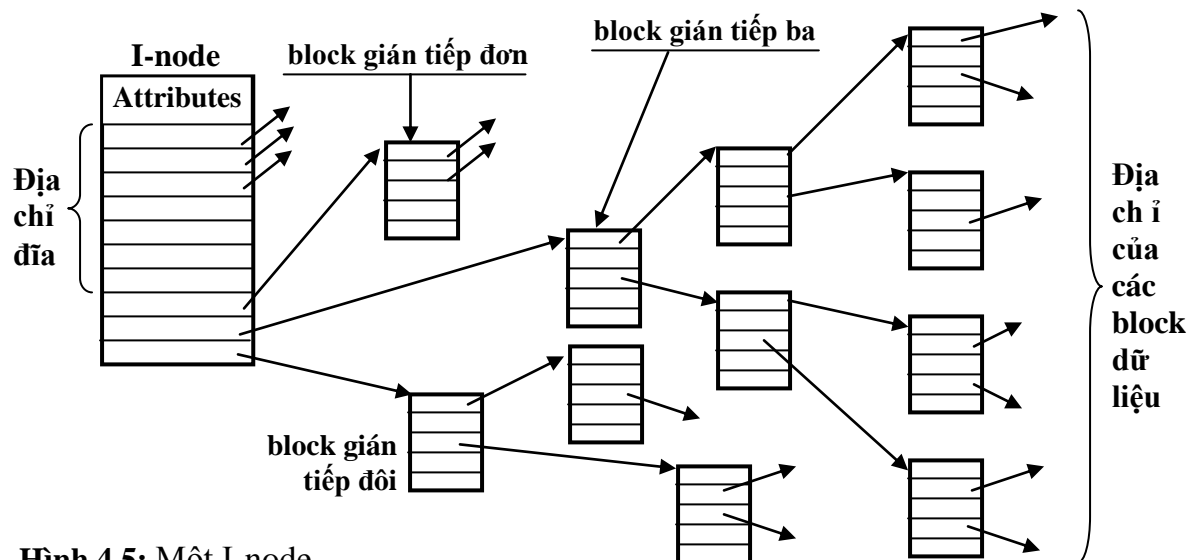
Với cách tổ chức này thì toàn bộ block đĩa được sử dụng để lưu trữ file và việc truy cập ngẫu nhiên trong trường hợp này sẽ dễ dàng hơn. Tuy nhiên cũng phải tồn tại một móc xích để tìm ra tất cả các block đĩa chứa nội dung của một file và móc xích này phải được nạp vào bộ nhớ để hệ điều hành có thể tìm đọc file khi cần. Cũng như chiến lược trên block đầu tiên của một file phải được chứa trong phần tử bảng danh mục tương ứng với mỗi file, trong trường hợp này nó được xem như một con trỏ trỏ đến bảng chỉ mục để bắt đầu dò tìm dãy các block đĩa chứa nội dung của file, mỗi khi hệ điều hành cần đọc file. Hệ điều hành MS\_DOS tổ chức quản lý file trên đĩa dựa theo chiến lược này.

Một hạn chế lớn của chiến lược này là toàn bộ bảng chỉ mục phải nạp vào bộ nhớ trong suốt thời gian làm việc của hệ thống, điều này sẽ làm tốn thời gian nạp

bảng chỉ mục của hệ điều hành và làm lãng phí không gian bộ nhớ của hệ thống, đặc biệt trong trường hợp bảng chỉ mục lớn. Bảng chỉ mục lớn là do đĩa lớn, đĩa có bao nhiêu block thì bảng chỉ mục có bấy nhiêu phần tử, mỗi phần tử trong bảng chỉ mục có thể là 1 word, 1.5 word, 2 word, 4 word, vv phụ thuộc vào kích thước đĩa, kích thước block và cách tổ chức quản lý block đĩa của mỗi hệ điều hành.

Các hệ điều hành hiện nay khắc phục hạn chế trên đây bằng cách, không nạp tất cả bảng chỉ mục vào bộ nhớ mà chỉ nạp phần bảng chỉ mục liên quan đến các file đang mở trên bộ nhớ tại một thời điểm cụ thể nào đó, tức là, phần bảng chỉ mục này luôn thay đổi trong quá trình làm việc của hệ thống. Khái niệm *cửa sổ bảng FAT* trong hệ thống file của hệ điều hành windows98 là một ví dụ của trường hợp này. Chúng ta sẽ được nhắc đến điều này trong phần sau của chương này.

➤ **I-nodes (index-node):** trong chiến lược này, hệ điều hành thiết kế một bảng nhỏ để theo dõi các blocks của một file, được gọi là I-node. I-node liệt kê các thuộc tính và các địa chỉ đĩa của các block của file. Hình sau đây minh họa cho chiến lược này.



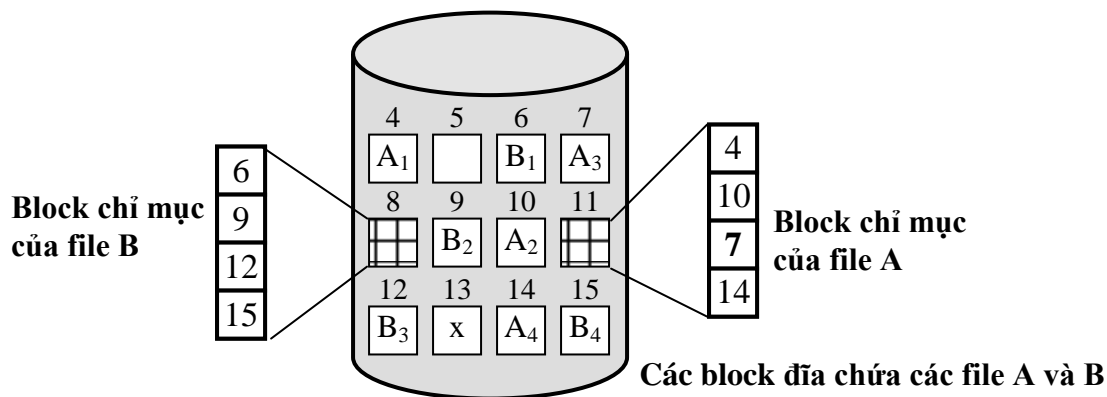
Hình 4.5: Một I-node

Đầu tiên một phần địa chỉ đĩa (các block đĩa) được lưu trữ trong chính I-node. Sau đó, đối với các file nhỏ thì tất cả các thông tin cần thiết là phải chứa trong chính I-node, đó là các thông tin được nhận từ đĩa vào bộ nhớ chính khi file được mở. Đối với các file lớn, gồm nhiều block, thì một trong các địa chỉ trong I-node là địa chỉ của một block đĩa, được gọi là block gián tiếp đơn. Block này chứa các địa chỉ đĩa được thêm vào. Nếu vẫn còn không đủ thì một địa chỉ khác trong I-node, được gọi là block gián tiếp đôi, sẽ chứa địa chỉ của một block mà nó chứa một danh sách các block gián tiếp đơn. Mỗi block gián tiếp đơn trở đến khoảng 100 block dữ liệu. Nếu vẫn còn không đủ thì có thể một block gián tiếp ba được sử dụng. Nhìn hình vẽ trên ta dễ dàng phân biệt được sự khác nhau giữa: block gián

tiếp đơn, block gián tiếp đôi và block gián tiếp ba.

Chiến lược này được windows 2000 cải tiến và sử dụng trong cấu trúc MFT trong hệ thống file của nó. Chúng ta sẽ thấy điều này khi tìm hiểu hệ thống file của windows 2000 trong phần sau của chương này.

➤ **Cấp phát không liên tục với block chỉ mục:** Cả hai chiến lược cấp phát, theo danh sách liên kết và theo liên kết chỉ mục đều tồn tại hạn chế là phải phân tích danh sách liên kết hay bảng chỉ mục để dò tìm ra danh sách các block đĩa chứa nội dung của tập tin cần đọc, khi đọc tập tin, dẫn đến làm chậm tốc độ đọc tập tin trên đĩa.



**Hình 4.6:** Cấp phát không liên tục với block chỉ mục

Để khắc phục điều này các hệ điều hành có thể cài đặt chiến lược cấp phát không liên tục với block chỉ số. Hệ điều hành sử dụng một block đĩa để chứa danh sách các block đĩa chứa nội dung của một tập tin nào đó, block đĩa này được gọi là block chỉ mục. Trong hình trên block 11 là chỉ mục của file A, block 8 là chỉ mục của file A. Như vậy chỉ cần thiết kế một con trỏ, tại phần tử trong bảng chỉ mục, trỏ tới block chỉ mục của tập tin trên đĩa là hệ điều hành có thể quản lý được danh sách các block đĩa chứa nội dung của một tập tin.

Với chiến lược này thì tốc độ đọc file của hệ điều hành sẽ tăng lên, nhưng nó chỉ dụng được đối với các file nhỏ, vì nếu file lớn thì một block có thể không chứa đủ danh sách các block đĩa chứa nội dung của một file. Mặt khác nếu block chỉ mục của file bị hỏng thì hệ điều hành không thể đọc được file, mặc dầu nội dung của file vẫn còn tồn tại trên các block đĩa.

#### IV.1.5. An toàn trong quản lý tập tin

**Bảo toàn dữ liệu tập tin:** Một hệ quản trị file phải cung cấp những cơ chế thích hợp để phục hồi nội dung của file trong trường hợp hệ thống gặp sự cố về phần mềm hoặc phần cứng. Để thực hiện được điều này hệ điều hành phải luôn tạo bản sao của các tập tin đang mở trên hệ thống, để có thể phục hồi lại khi cần thiết. Có hai kỹ thuật được sử dụng trong cơ chế này:

- **DUMP có chu kỳ:** Sau một khoảng thời gian nhất định nội dung của các tập tin đang mở trên bộ nhớ chính sẽ được đổ (Dump/backup) ra lại đĩa. Nếu hệ thống gặp sự cố thì tất cả các tập tin đang mở sẽ được tái tạo lại kể từ trạng thái mà chúng được DUMP ra lần cuối cùng. Rõ ràng việc DUM này sẽ làm tốn thời gian thực hiện của hệ thống.

- **DUMP Incremental:** Trong cách này, hệ thống chỉ lưu trữ các thông tin được sửa đổi kể từ lần Dump sau cùng, tức là chỉ có các tập tin được tạo lập hoặc sửa đổi so với lần đổ ra cuối cùng mới được Dump ra. Với kỹ thuật này thông tin cần lưu trữ ít hơn do đó hệ thống có thể thực hiện Dump thường xuyên hơn.

Để biết được trong số những tập tin đang mở tập tin nào có sự cập nhật dữ liệu hoặc có sự thay đổi so với lần Dump ra trước đó hệ thống đưa thêm vào danh mục người sử dụng một trường mới, dài 2 bit, tạm gọi là trường kiểm tra cập nhật (KTCN). Nếu KTCN = 00: mở không cập nhật; KTCN = 01: mở có cập nhật; KTCN = 10: không có thay đổi so với lần Dump trước. KTCN = 11: có thay đổi so với lần Dump trước.

Với cách này hệ thống phải luôn kiểm tra bảng danh mục và phải cập nhật lại trường KTCN sau mỗi lần Dump, dẫn đến làm chậm tốc độ thực hiện của hệ thống.

Để hệ thống không phải khảo sát tất cả các điểm vào của danh mục, hệ điều hành cài đặt thêm một bảng danh mục mới để ghi nhận thông tin của các tập tin đang được truy xuất (ghi/đọc) trên hệ thống và chỉ có Dump sử dụng bảng danh mục này, do đó hệ thống Dump có thể hoạt động song song với các thao tác khác của hệ thống.

Dump Incremental là một tiến trình có độ ưu tiên thấp, thường trú trong bộ nhớ phân tích các bảng danh mục để tìm ra các tập tin cần phải thực hiện Dump.

**Danh sách các quyền truy cập (Access Right):** Trong phần trình bày về tập tin chia sẻ ở trên, chúng tôi đã trình bày về kỹ thuật tạo ra tập tin chia sẻ của hệ điều hành, kỹ thuật này hoàn toàn trong suốt với người sử dụng. Trong phần này chúng tôi trình giới thiệu một công cụ mà hệ điều hành dùng để bảo vệ các tập tin chia sẻ trong môi trường nhiều người sử dụng. Đó là quyền truy cập, quyền truy cập và quản lý truy cập đồng thời là các công cụ cơ bản mà hệ điều hành dùng để quản lý và bảo vệ các tập tin chia sẻ trong các hệ thống nhiều người sử dụng (multiuser systems).

Quyền truy cập có thể được gán cho một người sử dụng (User) cụ thể, một nhóm người sử dụng (User Group) hay tất cả người sử dụng (All User) có trong các hệ thống multiuser. Một user group chứa nhiều user, khi một group được gán quyền nào đó thì tất cả các user thành viên trong group này đều được cũng được gán quyền truy cập đó.

Sau đây là các quyền truy cập mà hệ điều hành thường dùng để gán cho một người sử dụng cụ thể đến một file cụ thể nào đó:

- **None:** Người sử dụng không biết được là file có tồn tại hay không. Với giới hạn của quyền này, người sử dụng không được phép đọc thư mục chứa file này.
- **Knowledge:** Người sử dụng có thể xác định được là file đang tồn tại và ai là người sở hữu file.
- **Excution:** Người sử dụng có thể nạp và thực hiện một chương trình nhưng không thể copy nó. Các chương trình thuộc dạng độc quyền của một nhà sản xuất nào đó thường được tạo với sự giới hạn với quyền này.
- **Reading:** Người sử dụng có thể đọc file cho bất kỳ mục đích nào, bao gồm cả copy và execution. Một vài hệ thống cho phép có sự khác nhau giữa xem và copy file. Trong trường hợp này nội dung của file có thể được hiển thị để người sử dụng xem, nhưng họ không được cung cấp công cụ để copy nội dung này.
- **Appending:** Người sử dụng có thể thêm dữ liệu vào file, thường là ở cuối file, nhưng không thể thay đổi hoặc xoá bất kỳ một nội dung nào trong file.
- **Updating:** Người sử dụng có thể thay đổi, xoá và thêm dữ liệu vào file.
- **Changing protection:** Người sử dụng có thể thay đổi các quyền truy cập được gán đến người sử dụng khác. Quyền này thường chỉ được gán cho người sở hữu file.
- **Deletion:** Người sử dụng có thể xoá được file từ hệ thống file.

Người sử dụng được gán quyền truy cập đến file, và họ chỉ có thể truy cập file ở mức độ tương ứng với quyền truy cập được gán. Ví dụ, người sử dụng A được gán quyền đọc (read) file **tailieu.doc**, nhưng không được gán quyền xoá (delete) file **tailieu.doc** thì người sử dụng A này chỉ có thể thực hiện thao tác mở file **tailieu.doc** ra để đọc nội dung của file, chứ không thể thay xoá hay thay đổi nội dung của file (vì không được gán quyền thay đổi (modify) nội dung file).

Người sử dụng có thể được gán nhiều quyền truy cập đến một file, khi đó họ sẽ có đầy đủ các sự cho phép và sự giới hạn tương ứng với các quyền đã được gán. Tuy nhiên quyền truy cập có tính kế thừa, nên chỉ cần gán một quyền truy cập cao nhất thì họ có đủ các sự cho phép và sự giới hạn của các quyền khác. Ví dụ, nếu người sử dụng được gán quyền Updating với một file nào đó, thì xem như họ đã được gán các quyền Knowledge, execution, reading và appending đối với file này.

**Mở và đóng tập tin:** Hệ điều hành cho rằng các tập tin được lưu trữ trên đĩa đều ở trạng thái đóng, để thực hiện bất kỳ một thao tác đọc/ghi/thay đổi nội dung của tập tin thì trước hết chương trình, tiến trình của người sử dụng (kể cả người sử dụng) phải thực hiện thao tác mở tập tin. Khi nhận được yêu cầu mở tập tin bộ phận quản

lý tập tin của hệ điều hành sẽ đọc nội dung của tập tin từ đĩa và nạp nó vào bộ nhớ chính, sau đó trả về cho chương trình, tiến trình của người sử dụng một thẻ tập tin/ thẻ file (file handle) hoặc một biến tương ứng với tập tin này để chương trình, tiến trình theo dõi và thao tác trên tập tin này. Sau khi thực hiện xong một thao tác nào đó trên nội dung của tập tin thì chương trình, tiến trình và cả người sử dụng phải thực hiện thao tác đóng tập tin lại. Đối tượng yêu cầu đóng tập tin phải cung cấp đúng thẻ tập tin của tập tin cần đóng cho hệ điều hành.

Một số hệ điều hành cho phép thực các thao tác trên tập tin (mở/cập nhật/ đóng) bằng chính tên của tập tin. Các hệ điều hành đều cung cấp hai thủ tục chính để chương trình của người sử dụng thực hiện các thao tác mở/đóng file: **Open** (*tên file cần mở, chế độ mở*): dùng để mở file (chế độ: Đọc/ Viết/ Tạo lập) và **Close** (*tên file cần đóng*): dùng để đóng file khi mở.

Thao tác mở/đóng file sẽ đơn giản trong môi trường hệ điều hành đơn nhiệm và sẽ phức tạp hơn trong môi trường hệ điều hành đa nhiệm. Trong môi trường đa nhiệm, hệ điều hành chỉ thực sự đóng file theo yêu cầu của một tiến trình từ một người sử dụng nào đó khi tất cả các thao tác ghi/đọc file này từ các tiến trình người sử dụng khác đều đã kết thúc. Trong trường hợp này hệ điều hành phải luôn theo dõi các tiến trình người sử dụng tham gia vào việc mở file này. Để đáp ứng yêu cầu mở file từ một chương trình, tiến trình của người sử dụng trong môi trường đa nhiệm hệ điều hành phải thực hiện các bước cơ bản sau đây để đảm bảo việc truy xuất file sau này là hợp lệ:

1. Kiểm tra tên của file cần mở, tại các entry, trong bảng danh mục file của hệ thống (đó là bảng thư mục trong hệ điều hành DOS và Windows9x).
2. Kiểm tra tiến trình gọi tới từ một người sử dụng hay chương trình người sử dụng có được quyền truy cập file ở chế độ đã được chỉ ra hay không.
3. Kiểm tra nếu file đã được mở để đọc bởi một tiến trình trước đó thì tiến trình hiện tại không thể mở để ghi vào file, mặc dầu tiến trình này được quyền ghi file. Ngược lại tiến trình hiện tại không thể mở file để đọc khi đã có một tiến trình nào đó đang ghi vào file.
4. Đảm bảo sự sẵn sàng của các thiết bị lưu trữ, đĩa chẳng hạn, và vật mang liên quan đến file cần mở.

Để mô tả đầy đủ các thông tin về một file thì một phần tử trong bảng danh mục cần phải chứa các trường sau: Tên file; Mô tả của đơn vị của lưu trữ file; Địa chỉ của Block đầu tiên trong dãy các block (trên đĩa) chứa file; Địa chỉ của các block kế tiếp trong dãy các block chứa file; Chế độ truy cập tập tin; vv.

Trong môi trường hệ điều hành đa nhiệm có thể có các tiến trình song song cùng đọc nội dung của một file, đối với các file chia sẻ, nhưng không thể xảy ra trường hợp có hai tiến trình cùng ghi vào một file hoặc có một tiến trình ghi vào

file trong khi có một hoặc nhiều tiến trình khác đang đọc nội dung của file. Hệ điều hành phải kiểm soát chặt chẽ các trường hợp này. Để tránh hiện tượng này hệ điều hành phải tạo một cơ chế thích hợp để loại trừ lẫn nhau trong thao tác đọc/ghi file giữa các file đồng thời.

Để thực hiện loại trừ lẫn nhau này hệ điều hành đưa thêm hai trường vào các entry trong bảng danh mục người sử dụng: Trường thứ nhất, **Bít ghi**, = 1 đang có một tiến trình ghi vào file, = 0 không có tiến trình nào ghi vào file. Trường thứ hai, **Bộ đếm**, = <số các tiến trình đang mở file để đọc>. Theo đó một tiến trình chỉ có thể mở file để đọc khi **Bít ghi** = 0, mở file để ghi khi **Bít ghi** = 0 và **Bộ đếm** = 0. Như vậy, ngay sau khi chấp nhận yêu cầu mở file để ghi từ một tiến trình thì hệ điều hành phải gán **Bít ghi** = 1, ngay sau khi chấp nhận yêu cầu mở file để đọc từ một tiến trình thì hệ điều hành phải tăng **Bộ đếm** lên 1 đơn vị, **Bộ đếm** = **bộ đếm** + 1. Khi một tiến trình đọc file đóng file thì **Bộ đếm** = **bộ đếm** - 1, khi một tiến trình ghi file đóng file thì **Bít ghi được gán** = 1. Rõ ràng kỹ thuật này có thể dẫn đến lỗi khi hệ thống không giám sát tốt việc thay đổi giá trị trên các trường Bít ghi và Bộ đếm, điều này chúng ta đã thấy trong chương *Quản lý tiến trình* của tài liệu này.

#### IV.1.6. Hiệu suất hệ thống file

Như đã biết, tốc độ truy xuất dữ liệu trên đĩa chậm hơn rất nhiều so với tốc độ truy xuất dữ liệu trên bộ nhớ, tốc độ truy xuất dữ liệu trên đĩa tính bằng đơn vị milliseconds, trong khi đó tốc độ truy xuất dữ liệu trên bộ nhớ chỉ tính bằng đơn vị nanoseconds. Do đó, để tạo ra sự đồng bộ trong việc trao đổi dữ liệu trên bộ nhớ và trên đĩa, cũng như tăng tốc độ truy xuất dữ liệu trên bộ nhớ, các hệ điều hành phải thiết kế hệ thống file của nó sao cho tốc độ đọc dữ liệu là nhanh nhất và giảm số lần truy cập đĩa mỗi khi truy xuất file xuống mức thấp nhất.

Một trong những kỹ thuật được hệ điều hành sử dụng ở đây là tạo ra các block cache hoặc buffer cache. Trong ngữ cảnh này, cache là một tập các block logic trên đĩa, nhưng được tạo ra và được giữ trong bộ nhớ chỉ để phục vụ cho mục đích cải thiện hiệu suất của hệ thống.

Có nhiều thuật toán khác nhau được sử dụng để quản lý cache, nhưng tất cả đều hướng tới mục đích của việc sử dụng cache và nguyên lý hoạt động của cache: Khi nhận được một yêu cầu đọc dữ liệu từ tiến trình của người sử dụng thì bộ phận quản lý cache sẽ kiểm tra block dữ liệu cần đọc đã có trong cache hay chưa, nếu có trong cache thì đọc trực tiếp trong cache mà không cần truy cập đĩa, nếu không có trong cache thì dữ liệu cần đọc sẽ được đọc và ghi vào trong cache trước rồi sau đó được chép đến bất cứ nơi nào cần thiết. Việc ghi vào cache này nhằm chuẩn bị cho các lần đọc dữ liệu sau này. Tức là, nếu sau này có một yêu cầu đọc cùng một block dữ liệu như trên thì nó sẽ được đọc trực tiếp từ cache mà không cần truy cập đĩa.



Khi cache bị đầy các block thì một vài block trong đó phải bị xoá hoặc bị xoá và ghi trở lại về đĩa nếu block này cần thay đổi kể từ khi nó được mang vào bộ nhớ kể từ lần được mang vào gần đây nhất. Trong trường hợp này hệ điều hành cũng sử dụng các thuật toán thay trang trong quản lý bộ nhớ như FIFO, LRU, để chọn một block trong cache để đưa ra đĩa. Tuy nhiên cache được truy xuất thường xuyên hơn, nên hệ điều hành có thể tổ chức một danh sách liên kết để theo dõi việc truy xuất các block trong cache, danh sách liên kết này được sử dụng cho thuật toán thay block: LRU.

### Một số khái niệm dùng trong quản lý đĩa

➤ *Track (tờ đạo):* Là các vòng tròn đồng tâm được tạo ra trên bề mặt đĩa, đây sẽ là nơi chứa dữ liệu sau này. Các track được đánh số bắt đầu từ 0. Số track trên mỗi mặt đĩa phụ thuộc vào từng loại đĩa.

➤ *Sector (cung từ):* Các track được chia thành các khối có kích thước cố định bằng nhau và được đánh địa chỉ, các khối này được gọi là các sector. Các sector được đánh địa chỉ bắt đầu từ 1 trên mỗi track, như vậy trên đĩa sẽ tồn tại nhiều sector có cùng số hiệu địa chỉ, cách đánh địa chỉ này gây khó khăn nhiều người lập trình.

Kích thước của sector, số byte dữ liệu có thể chứa trên một sector, phụ thuộc vào phần cứng. Trên các họ processor x86, kích thước sector trên đĩa cứng thường là 512 byte, kích thước sector trên đĩa CD\_ROM thường là 2048 byte.

- Các sector được đánh địa chỉ theo kiểu trên được gọi là sector vật lý. Trong thực tế lập trình các hệ điều hành chỉ sử dụng sector logic, theo đó thì địa chỉ các sector được đánh bắt đầu từ 0 kể từ track 0 của mặt 0 trên đĩa thứ nhất. Như vậy trên đĩa không có các sector có cùng số hiệu địa chỉ. Bảng sau đây cho thấy sự tương ứng giữa các sector vật lý với sector logic trên một đĩa mềm:

Mặt đĩa	Track	Sector	Sector logic	Thông tin lưu trữ
0	0	1	0	Boot record
0	0	2 - 5	1 - 4	FAT
0	0	6 - 9	5 - 8	Thư mục gốc
1	0	1 - 3	9 - 11	Thư mục gốc

1	0	4 - 9	12 - 17	Dữ liệu
0	1	1 - 9	18 - 26	Dữ liệu

**Bảng 4.1:** Tương ứng giữa sector vật lý và sector logic trên đĩa mềm

- Trên bề mặt đĩa tồn tại các sector mà hệ điều hành không thể ghi dữ liệu vào đó hoặc không thể đọc dữ liệu từ đó. Các sector này được gọi là bad sector. Trong quá trình định dạng đĩa hệ điều hành đánh dấu loại bỏ các bad sector này.
- *Cluster (liên cung):* Một nhóm gồm 2, 4 hoặc 6 sector liên tiếp nhau tạo thành một cluster. Kích thước của cluster thường là bội số kích thước của một sector. Các cluster được đánh địa chỉ bắt đầu từ 0. Số sector trên một cluster phụ thuộc vào từng loại đĩa. Một số hệ điều hành cho phép người sử dụng quy định số sector trên một cluster. Các hệ điều hành thường tổ chức lưu trữ dữ liệu, nội dung các tập tin, trên đĩa theo từng cluster. Trên bề mặt đĩa cũng tồn tại các bad cluster, đó là các cluster có chứa bad sector.  
 Một số hệ điều hành có thể khôi phục lại được dữ liệu chứa trên các bad-sector hay bad cluster và ghi nó vào lại một cluster mới. Hệ điều hành có thể chỉ khôi phục và thay thế dữ liệu tại sector bị bad hoặc phải khôi phục và thay thế toàn bộ dữ liệu trên cluster có chứa bad-sector.
- Hệ thống file NTFS của windowsNT/2000 tham chiếu đến các vị trí vật lý trên đĩa bằng số hiệu cluster logic (LCNs: logical cluster numbers). LCN là kết quả của việc đánh số tất cả các cluster trên volume từ vị trí bắt đầu volume đến kết thúc volume. Để chuyển một LCN thành địa chỉ vật lý trên đĩa, NTFS nhân LCN với thừa số cluster (số sector trên một cluster) để có được byte offset vật lý trên volume. NTFS tham chiếu đến dữ liệu trong phạm vi một file bằng số hiệu cluster ảo (VCNs: Virtual cluster numbers), VCN đánh số các cluster dựa vào một file cụ thể và đánh số từ 0 đến m. Các VCN không cần phải liên tục về mặt vật lý, tuy nhiên nó có thể ánh xạ đến bất kỳ một LCN nào trên volume.
- *Cylinder (từ trụ):* Các track có cùng số hiệu trên các mặt đĩa khác nhau của một hệ thống đĩa tạo thành một cylinder. Như vậy mặt đĩa có bao nhiêu track thì đĩa có bấy nhiêu cylinder. Cylinder chỉ có trên các ổ đĩa cứng.
- *Partition (phân khu):* Partition là một tập các sector liên kề trên một đĩa. Mỗi partition có một bảng partition hoặc một cơ sở dữ liệu quản lý đĩa riêng, dùng để lưu trữ sector đầu tiên, kích thước và các đặc tính khác của partition.
- *Volume:* Một volume tương tự một partition logic trên một đĩa, và nó được tạo khi ta định dạng một đĩa hoặc một phần của đĩa theo hệ thống file NTFS. Trong hệ điều hành windowsNT/2000 ta có thể tạo ra một volume trải dài trên nhiều đĩa vật lý khác nhau. Một đĩa có thể có một hoặc nhiều volume. NTFS điều khiển mỗi volume sao cho không phụ thuộc vào các volume khác.

Một volume bao gồm một tập các file cùng với bất kỳ một không gian chưa được cấp phát nào còn lại trên partition đĩa. Trong hệ thống file FAT, một volume cũng chứa các vùng đặc biệt được định dạng cho việc sử dụng của hệ thống file. Trong các volume NTFS thì ngược lại nó lưu trữ tất cả dữ liệu của hệ thống file, như là bitmap, directory và cả system bootstrap, trên các file.

- *Simple volume*: là các đối tượng đại diện cho các sector từ một partition đơn, mà các trình điều khiển hệ thống file, quản lý nó như một đơn vị đơn.
  - *Multipartition volume*: là các đối tượng đại diện cho các sector từ nhiều partition khác nhau, mà các trình điều khiển hệ thống file quản lý nó như một đơn vị đơn. Các multipartition volume có các đặc tính mà các simple volume không có được như: hiệu suất cao, độ tin cậy cao và khả năng mở rộng kích thước.
- *Metadata*: là một dạng dữ liệu đặc biệt, được lưu trữ trên đĩa, nó hỗ trợ cho các thành phần quản lý các dạng thức hệ thống file khác nhau, dữ liệu của nó có thể là vị trí của các tập tin/ thư mục trên các ổ đĩa. Metadata không được sử dụng trong các ứng dụng.
- *File system (hệ thống file)*: Các dạng thức hệ thống file định nghĩa cách mà dữ liệu file được lưu trữ trên thiết bị lưu trữ và sự tác động của hệ thống file đến các file. Một dạng thức hệ thống file cũng có thể đưa ra các giới hạn về kích thước của các file và các thiết bị lưu trữ mà hệ thống file hỗ trợ. Một vài hệ thống file hỗ trợ cho cả các file lớn hoặc nhỏ, hoặc cả các đĩa lớn và nhỏ.

*Một hệ thống file thường bao gồm các thành phần: Sector khởi động (Boot sector), bảng định vị file (FAT: File Allocation Table), bảng thư mục gốc (Root Directory), một tập các file các thư mục và các công cụ quản lý các thành phần này. Các thành phần này có thể có cấu trúc hoặc phương thức tổ chức khác nhau trên các dạng thức hệ thống file khác nhau. Người ta thường dùng tên của FAT trong hệ thống file để gọi tên của hệ thống file đó.*

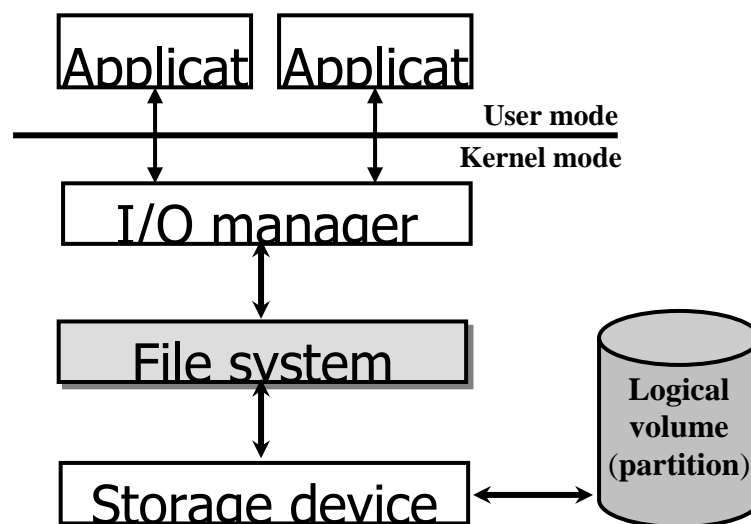
Hệ điều hành MS\_DOS sử dụng hệ thống file FAT12 và FAT16, hệ điều hành Windows9x sử dụng hệ thống file FAT32 và CDFS, hệ điều hành Windows NT và Windows 2000 sử dụng các hệ thống file FAT12, FAT16, FAT32, CDFS (CD\_ROM File System, UDF (Universal Disk Format) và NTFS (New Technology File System).

### Các điều khiển hệ thống tập tin

Các điều khiển hệ thống tập tin (FSD: File system driver) quản lý các dạng thức hệ thống file khác nhau. FSD chính thức xuất hiện từ windowsNT/2000. Trong windows 2000 có 2 loại FSD: Local FSD và Network/ Remote FSD.

**Local FSD:** quản lý các volume được nối trực tiếp với máy tính. **Network/Remote FSD:** cho phép người sử dụng và chương trình của người sử dụng truy cập dữ liệu trên các volume được nối với một máy tính ở xa.

➤ **Local FSD (FSD cục bộ):** Các Local FSD bao gồm các tập tin: Ntfs.sys, Fastfat.sys, Cdfs.sys và Raw FSD (được tích hợp trong Ntoskrnl.exe). Hình sau đây cho thấy cách local FSD tương tác với quản lý I/O và các thiết bị lưu trữ. Các local FSD chịu trách nhiệm đăng ký với bộ phận quản lý I/O, khi FSD đã đăng ký thì bộ phận quản lý I/O có thể gọi nó để thực hiện việc xác nhận volume khi các ứng dụng hoặc các hệ thống khởi tạo truy cập đến volume.



**Hình 4.7.a:** FSD cục bộ

Việc xác nhận volume bao hàm việc kiểm tra boot sector của volume và các thông tin hệ thống khác. Sector đầu tiên của mọi dạng thức hệ thống file được hỗ trợ bởi windows 2000 đều được dành riêng cho boot sector của volume. Boot sector chứa đầy đủ thông tin cần thiết để local FSD vừa nhận biết mà sector trên đó đang chứa một dạng thức mà FSD quản lý và tìm kiếm bất kỳ một metadata khác được lưu trữ trên đĩa.

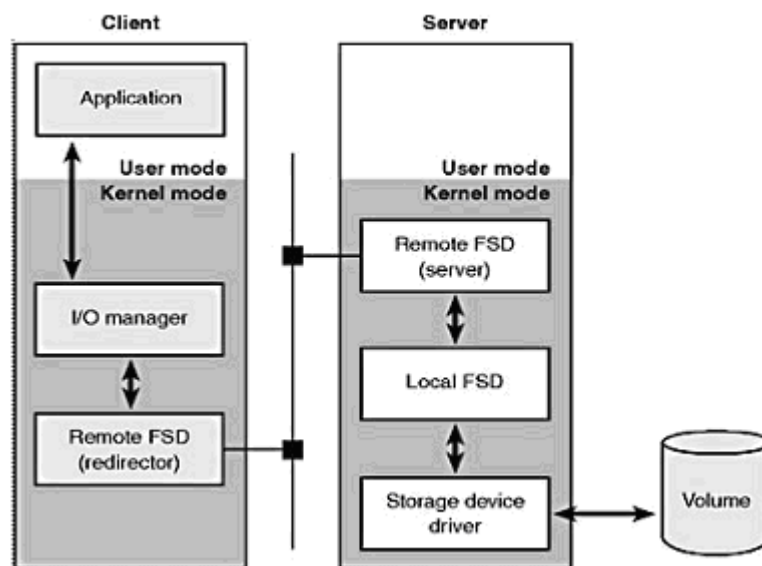
Để cải tiến hiệu suất, các local FSD thường sử dụng hệ thống quản lý cache để cache dữ liệu của hệ thống file bao gồm cả metadata.

➤ **Các Network/Remote FSD (FSD mạng/từ xa):** Các Remote FSD bao gồm 2 thành phần: Một Client và một Server. Các client remote FSD cho phép các ứng dụng truy cập đến các file và các thư mục ở xa.

Client FSD chấp nhận các yêu cầu I/O từ các ứng dụng và chuyển nó thành

các lệnh trong các giao thức về hệ thống file của mạng để thông qua mạng nó được chuyển đến server remote FSD. Server FSD lắng chờ các lệnh được đưa đến từ kết nối mạng và thực hiện chúng bằng cách đưa ra yêu cầu I/O đến bộ phận quản lý local FSD (Local FSD manages) của volume chứa các file và các thư mục mà lệnh có ý định xử lý nó. Hình dưới đây cho thấy một tương tác giữa client và server trong hệ thống remote FSD.

Cũng giống như các local FSD, các client remote FSD thường sử dụng những dịch vụ của bộ phận quản lý cache để che dấu dữ liệu của các tập tin cục bộ và các thư mục ở xa. Các server remote FSD tham gia vào việc duy trì các kết nối đến cache thông qua các client remote FSD.



**Hình 4.7.b:** FSD mạng

**Các hệ thống file được sử dụng trên các hệ điều hành hiện nay**

➤ **FAT12, FAT16, FAT32:** Hệ thống file FAT12 và FAT16 được Microsoft đưa ra sử dụng từ hệ điều hành DOS, hệ thống file FAT32 được Microsoft đưa ra sử dụng từ hệ điều hành windows98. Hệ điều hành windowsNT/2000 vẫn sử dụng các hệ thống file FAT này nhưng linh hoạt hơn.

Mỗi loại FAT có một con số để chỉ ra số lượng bit mà hệ thống file sử dụng để nhận dạng các cluster trên đĩa. FAT12 sử dụng 12 bit để định danh các cluster trên đĩa, do đó với FAT12 hệ thống file chỉ quản lý được 4096 ( $2^{12} = 4096$ ) cluster trên đĩa. Hệ điều hành windows 2000 cho phép các cluster có kích thước từ 512 byte đến 8Kb, vậy với FAT12 windows 2000 có thể quản lý được 32Mb đĩa, điều này có nghĩa windows 2000 chỉ dùng FAT12 để quản lý các đĩa mềm.

Kích thước volume	Kích thước cluster
0-32 MB	512 byte
32 Mb – 64 Mb	1 Kb
65 Mb – 128 Mb	2 Kb
129 Mb – 256 Mb	4 Kb
257 Mb – 512 Mb	8 Kb
513 Mb – 1023 Mb	16 Kb
1024 Mb – 2047 Mb	32 Kb
2048 Mb – 4095 Mb	64 Kb

**Bảng 4.2:** Kích thước cluster phụ thuộc vào kích thước volume

Trên các hệ thống file FAT16, windows 2000 cho phép kích thước cluster đi từ 512 byte đến 64Kb, nên với FAT16 windows 2000 có thể quản lý một không gian đĩa lên đến 4Gb. Khi người sử dụng format đĩa, tùy theo dung lượng đĩa mà windows 2000 quyết định sử dụng hệ thống file nào: FAT12, FAT16 hay FAT32.

Trong windows 2000 kích thước cluster được chọn phụ thuộc vào dung lượng của ổ đĩa. Bảng 4.2 cho thấy kích thước cluster được chọn, phụ thuộc vào dung lượng volume, trên hệ thống file FAT16.

Hệ thống file FAT32 được định nghĩa dựa trên các hệ thống file FAT. Trong thực tế FAT32 sử dụng chỉ sử dụng 28 bit, thay vì 32 bit, để định danh các cluster trên đĩa, vì đã dành riêng 4 bit cao cho mục đích khác. Kích thước của 1 cluster trên hệ thống FAT32 có thể lên đến 32Kb, nên theo lý thuyết thì FAT32 có thể quản lý đến 8Tb dung lượng partition/đĩa. Nhưng trong thực tế windows 2000 chỉ dùng FAT32 trên các partition/đĩa có kích thước nhỏ hơn 32Gb.

Sau đây là một số thuận lợi của FAT32 so với FAT12 và FAT16:

- Số phần tử/ mục vào (entry) trên thư mục gốc không có giới hạn.
- Thư mục gốc không cần lưu trữ tại một vị trí xác định trước.
- Kích thước của một cluster có thể lên đến 32Kb nên nó có thể quản lý được 8Tb, nhưng trong thực tế windows 2000 chỉ dùng FAT32 để quản lý có partition/đĩa có kích thước nhỏ hơn 32Mb.
- Chỉ dùng 28 bit để định danh các cluster, dùng 4 bit cao cho mục đích khác.
- Lưu trữ một bản copy của boot sector.
- Có hai bảng FAT trên một volume nhưng cả hai đều có vai trò như

nhau.

- Kích thước của file có thể lên đến 4Gb.

Hệ thống file FAT32 không được các hệ điều hành sử dụng để định dạng đĩa mềm.

➤ **NTFS: Là hệ thống file dành riêng cho windowsNT/2000. NTFS dùng 64 bit để định danh các cluster, nên nó có thể quản lý được các ổ đĩa có dung lượng lên đến 16 Exabyte (16 tỉ Gb). Trong thực tế windowsNT/2000 chỉ sử dụng 32 bit để định danh cluster, kích thước cluster là 64Kb, nên NTFS chỉ có thể quản lý được các ổ đĩa có dung lượng lên đến 128TB.**

NTFS có một số tính năng cao cấp như bảo mật các file/directory, cấp hạn ngạch cho đĩa, nén file, mã hoá file, ... Một trong những tính năng quan trọng của NTFS là khả năng phục hồi lỗi. Nếu hệ thống bị dừng một cách đột ngột, thì metadata của ổ đĩa FAT sẽ rơi vào tình trạng xung khắc dẫn đến làm sai lệch một lượng lớn dữ liệu tập tin và thư mục. Nhưng trên NTFS thì điều này không thể xảy ra, tức là cấu trúc của file/ Directory không bị thay đổi.

Tên file trong NTFS có độ dài không quá 255 ký tự, đường dẫn đầy đủ đến file dài không quá 32.567 ký tự. Tên file sử dụng mã Unicode. Tên file trong NTFS có sự phân biệt giữa chữ hoa và chữ thường

➤ **CDFS: Là hệ thống file được đưa ra để quản lý các file, thư mục trên các đĩa CD\_ROM. CDFS được ISO đưa ra vào năm 1998 theo chuẩn ISO9660, sau đó Microsoft phát triển theo đặc thù của nó để sử dụng trên windows98 và sau đó là windowsNT/2000. Dạng thức hệ thống file CDFS còn một số hạn chế như: Tên file và thư mục dài không quá 32 ký tự, cây thư mục không sâu quá 8 mức.**

➤ **UDF: Được windows 2000 phát triển dựa theo chuẩn ISO 13346 để thay thế cho CDFS, và dùng để quản lý các đĩa từ-quang, chủ yếu là các đĩa DVD\_ROM. UDF bao gồm cả các đặc tả DVD và có các điểm tiêu biểu sau: Tên tập tin có thể dài đến 255 ký tự, đường dẫn có thể dài đến 1023 ký tự, tên tập tin có thể được viết hoa hay viết thường.**

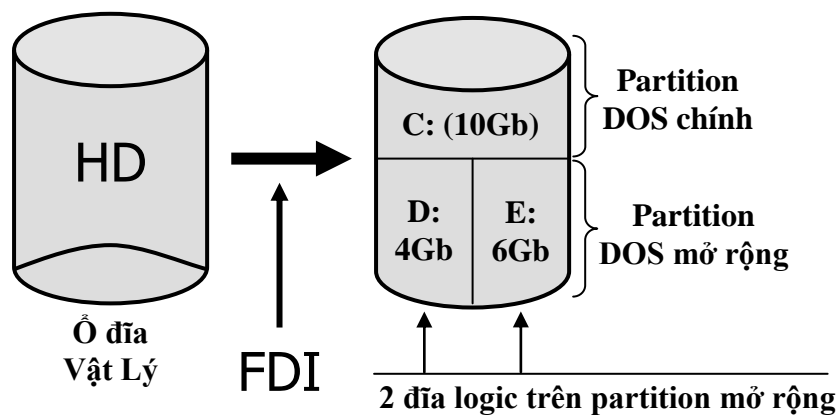
## **Tổ chức đĩa của MS\_DOS**

Chương trình FDISK của hệ điều hành cho phép chia không gian lưu trữ của đĩa cứng (đĩa cơ bản) thành các phần khác nhau, có thể có kích thước không bằng nhau, được gọi là các phân khu (partition) đĩa. Hệ điều hành DOS cho phép tạo ra 3 loại phân khu: Phân khu DOS chính (primary DOS), phân khu DOS mở rộng (Extended DOS), và phân khu phi DOS (non DOS). Muốn cài đặt nhiều hệ điều hành trên một máy tính, hay chính xác hơn là trên một ổ đĩa cơ bản, thì trước hết phải chia đĩa thành các phân khu, sau đó trên các phân khu khác nhau sẽ cài đặt các hệ điều hành khác nhau, thường là MS\_DOS hoặc windows98.

Thông thường ổ đĩa cứng được chia thành 2 phân khu: DOS chính và DOS mở rộng, cũng có thể chỉ tạo thành một phân khu DOS chính. Theo quy định của hệ điều hành, đĩa C: được hình thành trên phân khu DOS chính một cách tự động và chiếm toàn bộ kích thước của phân khu. Người sử dụng phải thực hiện việc tạo ra các đĩa logic (D:, E:, ...) trên phân khu DOS mở rộng trong quá trình FDISK đĩa. Nếu không, phân khu DOS mở rộng sẽ không được sử dụng sau này. Ta có thể tạo ra 1, 2, 3, ... đĩa logic trên phân khu DOS mở rộng và có thể tổng kích thước của các đĩa logic trên phân khu mở rộng nhỏ hơn kích thước của phân khu này (để lại một phần cho mục đích khác sau này). Hệ điều hành chịu trách nhiệm boot hệ thống (MS\_DOS hoặc windows98) thường được cài đặt trên đĩa C: (trên phân khu DOS chính).

Quá trình FDISK đĩa chỉ tạo ra các phân khu và các đĩa logic C:, D:, E:, vv, sau đó người sử dụng phải thực hiện việc định dạng (format) các ổ đĩa này thì mới có thể sử dụng được. Nên nhớ phải định dạng hệ thống (format /s) cho đĩa C: và phải cài đặt hệ điều hành boot chính vào đĩa C:.

Hình sau đây cho thấy một ổ đĩa cứng vật lý được chia thành 2 phân khu và các đĩa logic được tạo ra trên các phân khu:

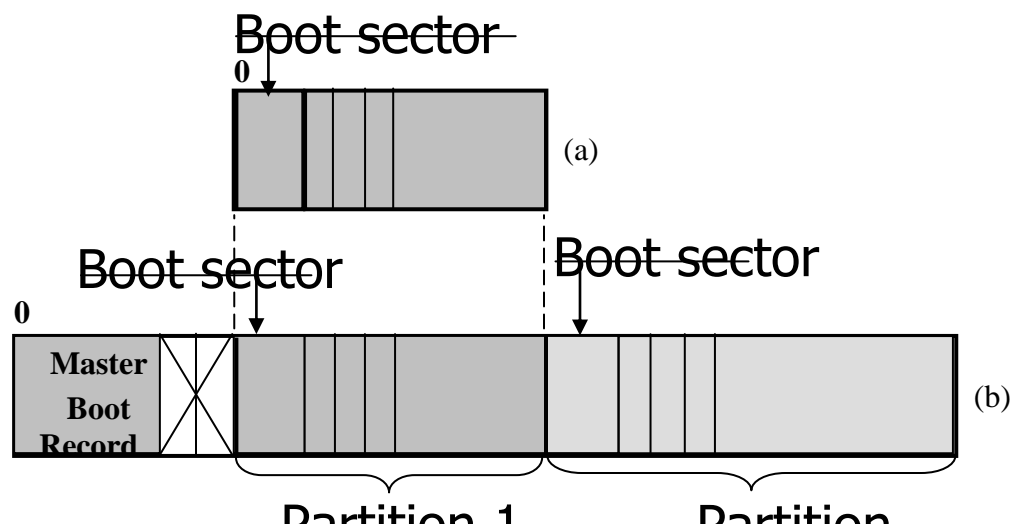


**Hình 4.8:** HDD trước và sau FDISK

Trong số các partition đã tạo phải có 1 (chỉ 1) partition được chọn là partition active (chủ động). Partition Active là partition mà sau này được chọn là partition boot hệ thống. Partition DOS chính thường được chọn là partition active.

Các partition khác nhau trên đĩa, có các thông tin sau đây khác nhau: Loại của partition; Partition có phải là Active hay không; Kích thước của partition; Vị trí bắt đầu và kết thúc của partition; Hệ điều hành được cài đặt trên partition; ... Để lưu trữ thông tin khác nhau của các partition, hệ điều hành DOS dùng một khối dữ liệu đặc biệt, được gọi là sector phân khu (partition sector), sector này nằm tại sector vật lý đầu tiên của đĩa cứng (head 0, track 0, sector 1) và nó không thuộc về bất kỳ một partition nào trên đĩa. Sector này thường được gọi là bảng partition. Hình vẽ sau đây minh họa cho điều này:





Hình 4.7: Tổ chức logic của FDD (a) và HDD (b)

Hình trên cũng cho thấy sự tương ứng về mặt về mặt logic giữa một đĩa mềm (a) với một partition/đĩa logic trên đĩa cứng (b). Điều đầu tiên chúng ta cần ghi nhận là master boot record (sector phân khu) chỉ có trên đĩa cứng, nó được tạo ra trong quá trình FDISK đĩa. Thứ hai là: Boot sector của đĩa mềm được định vị tại sector 0 của đĩa, trong khi đó boot sector của các đĩa logic trên các partition được định vị tại sector đầu tiên của partition và số hiệu của sector này được tìm thấy trong các phần tử trong bảng partition của master boot record bởi boot code ở đầu master boot record. Thứ ba: Master boot record không thuộc bất kỳ một partition nào và giữa nó và partition đầu tiên là một vùng trống, có thể DOS dự trữ cho các mục đích khác sau này. Vùng trống này là một kẽ hở của DOS, các đoạn code của Virus có thể được lưu trữ ở vùng này mà hệ điều hành không thể phát hiện được.

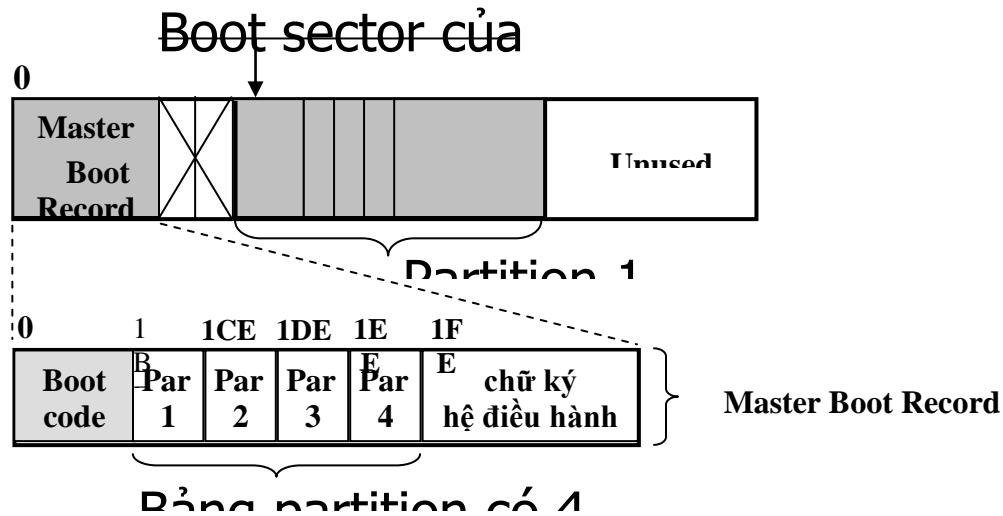
Khi master boot record trên đĩa cứng bị xóa hoặc bị nhiễm virus thì máy tính không thể khởi động được. Để khôi phục lỗi này ta chỉ có thể thực hiện như sau: Khởi động máy từ đĩa mềm, trên đĩa mềm có chứa tập tin FDISK.EXE, rồi sau đó thực hiện lại thao tác FDISK đĩa cứng với tham số MBR (A:\FDISK /MBR). FDISK /MBR làm mới lại master boot record mà không làm hỏng dữ liệu trên các đĩa logic.

Sector phân khu bao gồm 3 thành phần: Boot code, bảng partition và chữ kí hệ điều hành. Hình 4.8 sau đây cho thấy các thành phần trong Sector phân khu:

- **Boot code:** là một đoạn chương trình đặc biệt, được hệ điều hành ghi vào trong quá trình FDISK đĩa. Đoạn chương trình này có nhiệm vụ kiểm tra bảng partition để xác định xem trên đĩa có partition active hay không, nếu có thì đó là partition nào và bắt đầu tại sector nào, rồi sau đó nạp boot sector của đĩa trên

partition active vào RAM và chuyển quyền điều khiển về cho boot sector.

- **Bảng partition** (64 byte: bắt đầu từ byte 1BE h): gồm 4 phần tử, đánh số từ 1 đến 4, mỗi phần tử dài 16 byte dùng để lưu thông tin của một partition. Các thông tin trong một phần tử trong bảng partition cho biết: Phân khu có phải là active hay không; Vị trí bắt đầu phân khu (head, sector, cylinder); Vị trí kết thúc phân khu (head, sector, cylinder); Có bao nhiêu sector nằm trong phân khu; Kích thước của một phân khu tính theo sector; Phân khu được định dạng như thế nào và được cài đặt hệ điều hành nào?.



Hình 4.8: Các thành phần trong master boot record

Vì bảng partition chỉ có 4 phần tử nên DOS chỉ cho phép tạo ra tối đa là 4 partition. Đây là một hạn chế. Để khắc phục điều này hệ điều hành DOS cho phép tạo ra nhiều đĩa logic trên một partition mở rộng, tức là có thể tạo ra được nhiều đĩa logic trên một ổ đĩa cơ sở. Hệ điều hành windowsNT/ 2000 cho phép tạo ra nhiều hơn 4 partition trên một ổ đĩa và số lượng các phần tử trong bảng partition có thể thay đổi.

- **Chữ ký hệ điều hành** (2 byte: bắt đầu từ byte 1FEh): thường chứa giá trị 55AAh. Hệ điều hành DOS kiểm tra giá trị tại vùng này để biết đĩa này có phải được định dạng bởi nó hay không.

Một phần tử trong bảng phân khu chứa các thông tin sau:

Offset	Nội dung	
<b>Kích thước</b>		
00 h	0: partition không phải là active	1 byte
	80 h: partition là active	
01 h	số hiệu head bắt đầu phân khu	1 byte

02 h	sector và cylinde bắt đầu (của boot sector)	2 byte
04 h	mã hệ thống: 0: Non Dos; 1: FAT_12; 4: FAT_16; 5: phân khu Dos mở rộng; 6: phân khu Dos lớn hơn 32 Mb	1 byte
05 h	số hiệu head kết thúc phân khu	1 byte
06 h	sector và cylinde kết thúc	2 byte
08 h	số hiệu sector tương đối bắt đầu	4 byte
0C h	tổng số sector trong phân khu	4 byte

**Bảng 4.3:** Các trường một phần tử bảng partition

**Ví dụ 1:** Để kiểm tra partition nào là active ta thực hiện như sau:

- Đọc sector đầu tiên của đĩa cứng lưu vào biến masterboot
- Kiểm tra offset 00 của 4 phần tử partition trong bảng partition

```
Mov      cx, 4
Mov      SI, 1BE h
```

**Locate\_active:**

```
Mov      AL, masterboot[SI]
Cmp      AL, 80h
Je       Active
Add      SI, 16
Loop     Locate_active
```

**No\_active:**

.....

**Active:**

.....

**Ví dụ 2:** Để đọc nội dung boot sector của đĩa cứng C ghi vào biến BootDat ta phải thực hiện lần lượt các bước sau đây:

- Đọc sector đầu tiên của đĩa cứng lưu vào biến masterboot
- Tìm partition active (phần tử trong bảng partition có offset 00 bằng 80h)
- Đọc byte tại offset 01h và word tại offset 02 của phần tử partition tương ứng ở trên (head, sector và cylinde), để xác định số hiệu sector bắt đầu của partition active, đây chính là boot sector của đĩa cứng.
- Đọc nội dung của sector xác định được ở trên lưu vào BootDat.

**Active:**

```

Mov      ax, 0201h                ; đọc 1 sector
Mov      cx, word PTR mastorboot  [SI+2]    ; sector &
cylinder
Mov      dh, byte PTR mastorboot[SI+1]      ; head
Mov      dl, 80h                  ; đĩa cứng
Mov      es,cs                    ; trỏ ES:BX về
Lea      bx, BootDat              ; đầu vùng BootDat lưu
Int      13h

```

Nếu PC được khởi động bằng đĩa mềm (FDD) khởi động thì sau quá trình POST hệ thống sẽ nạp **boot sector** trên đĩa mềm vào bộ nhớ tại địa chỉ 0:7C00h sau đó quyền điều khiển được trao cho cho **boot sector**, để nó tiếp tục điều khiển quá trình khởi động. Nếu PC được khởi động bằng đĩa cứng khởi động (HDD/C:) thì sau quá trình POST hệ thống sẽ nạp **sector phân khu** của đĩa cứng vào bộ nhớ tại địa chỉ 0:7C00h, sau đó **boot code** trong sector phân khu thực hiện việc xác định **partition active** và nạp **boot sector** trên partition active vào bộ nhớ cũng tại địa chỉ 0:7C00h, sau đó quyền điều khiển được trao cho cho **boot sector**, để nó tiếp tục điều khiển quá trình khởi động tương tự như trong trường hợp đĩa mềm. Chính vì vậy sector phân khu thường được gọi là Master Boot Record, nó cũng được gọi là bảng partition.

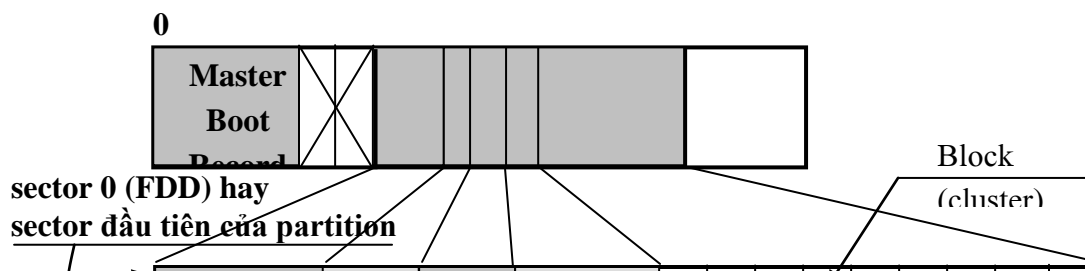
## Quản lý file trên đĩa của MS\_DOS

Trong quá trình định dạng một đĩa mềm, hoặc một đĩa logic trên các phân khu trên đĩa cứng, hệ điều hành chia không gian lưu trữ của đĩa thành 2 vùng: vùng dữ liệu (Data Area) và vùng hệ thống (System Area), đồng thời ghi những thông tin cần thiết vào vùng hệ thống để chuẩn bị cho việc quản lý lưu trữ sau này.

- Vùng dữ liệu: bao gồm các bolck có kích thước bằng nhau và được đánh địa chỉ (12 bit hoặc 16 bit) để phân biệt, đây chính là các cluster trên đĩa mà chúng ta đã nói đến ở trên. Nội dung của các tập tin cũng được chia thành các bolck có kích thước bằng kích thước của một cluster. Các cluster trên đĩa dùng để chứa nội dung của các tập tin trên đĩa. Các thông tin liên quan đến một tập tin trên đĩa được chứa ở vùng hệ thống.

- Vùng hệ thống: bao gồm các đoạn chương trình, các thông tin hệ thống, các thông tin liên quan đến các tập tin/thư mục trên đĩa mà hệ điều hành dùng để quản lý việc lưu trữ tập tin/thư mục trên đĩa sau này. Cụ thể nó bao gồm các thành phần sau đây: Boot sector, FAT1, FAT2 và Root Directory.

Sau đây chúng ta sẽ khảo sát các thành phần trong vùng hệ thống, để thấy được cách mà DOS quản lý các file và các thư mục được lưu trữ trên đĩa.



➤ Boot sector: còn được gọi là boot record (bản ghi khởi động), dài 512 byte (1 sector) được đặt tại sector logic 0 trên đĩa mềm hay sector logic đầu tiên của partition (đĩa logic) trên ổ đĩa cứng. Tất cả các đĩa (FDD và đĩa logic trên đĩa cứng) sau khi được định dạng đều có boot record và đều chứa các thông tin liên quan về đĩa trong đó, nhưng chỉ có đĩa được định dạng là đĩa khởi động mới có chứa một đoạn code *Bootstrap Loader*. Bootstrap Loader thực hiện việc nạp thành phần cốt lõi của DOS như io.sys, msdos.sys, command.com vào bộ nhớ RAM (chính xác hơn là chỉ nạp IO.SYS vào RAM sau đó IO.SYS sẽ tìm nạp các tập tin tiếp theo) trong quá trình khởi động máy tính. Chính vì vậy bootstrap loader còn được gọi là chương trình môi khởi động. Bảng sau đây cho thấy vị trí, độ lớn và nội dung của các trường trong boot sector.

Offset	Nội dung	Size
00 h	Lệnh JUMP, nhảy về Bootstrap Loader	3 byte
03 h	Tên nhà sản xuất và số phiên bản	8 byte
0B h	Số byte trên một sector	2 byte
0D h	Số sector trên một cluster	1 byte
0E h	Số sector dành cho boot sector	2 byte
10 h	Số bảng FAT	1 byte
11 h	Số phần tử (entry) trong Root directory	2 byte
13 h	Tổng số sector trên một tập đĩa (volume)	2 byte
15 h	Mã nhận diện đĩa	1 byte
16 h	Số sector dành cho bản FAT	2 byte
18 h	Số sector trên một track	2 byte
1A h	Số mặt (đầu từ)	2 byte

1C h	Số sector dữ trữ	4 byte
1E h	Số sector nếu kích thước lớn hơn 32Mb	4 byte
22 h	Số hiệu ổ đĩa: 0: ổ mềm; 80h: ổ cứng	1 byte
23 h	Dự trữ	1 byte
24 h	Chữ ký boot sector mở rộng	1 byte
25 h	Số Serial của đĩa, được tạo ra lúc format	4 byte
29 h	Tên tập đĩa (nhãn đĩa)	11 byte
34 h	Loại FAT: “FAT12” hoặc “FAT16”	8 byte
<b>3Ch - 200h</b>	<b>Code của chương trình bootstrap loader</b>	<b>452 byte</b>

**Bảng 4.4:** Các trường trong boot sector

Như vậy, ngay sau khi quyền điều khiển được trả về cho boot sector thì hệ thống sẽ thực hiện lệnh nhảy (Jmp) ở đầu boot sector (offset 00), để nhảy đến thực hiện đoạn code bootstrap loader ở cuối boot sector (từ offset 3Ch đến offset 200h). Và bootstrap loader sẽ thực hiện nhiệm vụ của nó.

Dựa vào boot sector ta biết được nhiều thông tin về đĩa như: loại FAT, nhãn đĩa, số sector trên một cluster, số byte trên một sector, ... Và từ đó ta cũng có thể tính được dung lượng của đĩa tính theo byte: ***Tổng số sector trên một tập đĩa \* số byte trên một sector.***

**Ví dụ:** Để in ra loại FAT đang sử dụng trên đĩa mềm hoặc trên một volume trên đĩa cứng ta thực hiện như sau:

- Đọc nội dung của boot sector lưu vào biến bootdat
- In ra 8 kí tự bắt đầu tại offset 34h của bootdat

```

Mov     cx, 8
Mov     SI, 34h
Mov     ah, 02                ; hàm 02h/21h in kí tự trong DL
Loai_FAT:
Mov     DL, byte PTR bootdat[SI]
Int     21h
Loop    Loai_FAT

```

➤ File Allocation Table (FAT): **Nội dung của một file cần lưu trữ trên đĩa được chia thành các phần có kích thước bằng nhau và bằng kích thước của một cluster, được gọi là các block file. Các block file của các file được lưu trữ tại các cluster xác định trên đĩa, các cluster chứa nội dung của một file có thể**

**không nằm kề nhau. Để theo dõi danh sách các cluster đang chứa nội dung của một file của tất cả các file đang lưu trữ trên đĩa hệ điều hành DOS dùng bảng FAT, hay còn gọi là bảng định vị file. Bảng FAT còn dùng để ghi nhận trạng thái của các cluster trên đĩa: còn trống, đã cấp phát cho các file, bị bad không thể sử dụng hay dành riêng cho hệ điều hành. Trong quá trình khởi động máy tính hệ điều hành nạp bảng FAT vào bộ nhớ để chuẩn bị cho việc đọc/ghi các file sau này.**

Khi cần ghi nội dung của một file vào đĩa hoặc khi cần đọc nội dung của một file trên đĩa hệ điều hành phải dựa vào bảng FAT, nếu bảng FAT bị hỏng thì hệ điều hành không thể ghi/đọc các file trên đĩa. Do đó, hệ điều hành DOS tạo ra hai bảng FAT hoàn toàn giống nhau là FAT1 và FAT2, DOS sử dụng FAT1 và dự phòng FAT2, nếu FAT1 bị hỏng thì DOS sẽ sử dụng FAT2 để khôi phục lại FAT1. Điều không đúng với hệ thống file FAT32, FAT32 vẫn tạo ra 2 FAT như của DOS, nhưng nếu FAT1 bị hỏng thì hệ điều hành sẽ chuyển sang sử dụng FAT2, sau đó mới khôi phục FAT1, và ngược lại.

Hệ điều hành DOS tổ chức cấp phát động các cluster cho các file trên đĩa, sau mỗi thao tác cấp phát/ thu hồi cluster thì hệ điều hành phải cập nhật lại nội dung cho cả FAT1 và FAT2. Có thể hệ điều hành chỉ thực hiện cấp phát động cluster cho các file dữ liệu (có kích thước thay đổi), còn đối với các file chương trình, file thư viện, file liên kết động, ... (có kích thước không thay đổi) thì hệ điều hành sẽ thực hiện cấp tĩnh cluster cho nó.

Bảng FAT bao gồm nhiều phần tử (điểm nhập/ mục vào), các phần tử được đánh địa chỉ bắt đầu từ 0 để phân biệt, địa chỉ cluster cũng có thể gọi là số hiệu của cluster. Giá trị dữ liệu tại một phần tử trong bảng FAT cho biết trạng thái của một cluster tương ứng trên vùng dữ liệu. Ví dụ, phần tử thứ 7 trong bảng FAT chứa giá trị 000h, giá trị này cho biết cluster thứ 7 trên vùng dữ liệu còn trống, có thể dùng để cấp phát cho một file. Phần tử thứ 5 trong bảng FAT chứa giá trị FF7h, giá trị này cho biết cluster thứ 5 trên vùng dữ liệu bị bad, không thể cấp phát được, ...

Hệ điều hành DOS có thể định dạng hệ thống file theo một trong 2 loại FAT là FAT12 và FAT16. Mỗi phần tử trong FAT12 rộng 12 bit(1.5 byte), mỗi phần tử trong FAT16 rộng 16 bit(2 byte). Các đĩa hiện nay thường được DOS định dạng theo hệ thống file với FAT16. Sau đây là danh sách các giá trị dữ liệu được chứa tại các phần tử trong bảng FAT (số trong ngoặc dùng trong FAT16) và ý nghĩa của nó.

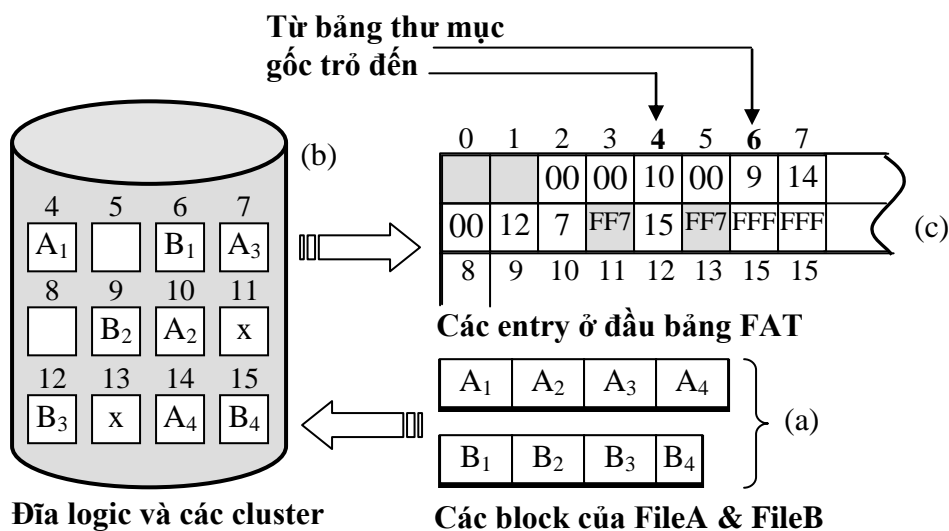
- *000h (0000h)*: cluster tương ứng còn để trống.
- *FF7h (FFF7h)*: cluster tương ứng bị bad. Trong quá trình định dạng đĩa hệ điều hành đánh dấu loại bỏ các cluster bị bad bằng cách ghi giá trị này vào phần tử tương ứng trong bảng FAT.

- *FF0h (FFF0h) - FF6h (FFF6h)*: cluster tương ứng dành riêng cho hệ điều hành.
- *FF8h (FFF8h) - FFFh (FFFFh)*: cluster tương ứng là cluster cuối cùng trong dãy các cluster chứa nội dung của một file.
- *002h (0002h) – FFEh (FFFEh)*: đây là số hiệu của cluster trong bảng FAT, nó cho biết cluster tiếp theo trong dãy các cluster chứa nội dung của một file.

Trong bảng FAT, hai phân tử đầu tiên (00 và 01) không dùng cho việc theo dõi trạng thái cluster và ghi nhận bảng đồ cấp phát file, mà nó được sử dụng để chứa một giá trị nhận biết khuôn dạng đĩa, được gọi là byte định danh (byte ID) của đĩa, đây là byte đầu tiên của bảng FAT. Đối với đĩa cứng thì byte ID = F8h.

Như vậy để đọc được nội dung của một file trên đĩa thì trước hết hệ điều hành phải tìm được dãy các cluster chứa nội dung của một file. Nhưng bảng FAT chỉ cho biết số hiệu các cluster từ cluster thứ hai đến cluster cuối cùng trong dãy nói trên. Cluster đầu tiên trong dãy các cluster chứa nội dung của một file trên đĩa được tìm thấy trong bảng thư mục gốc.

Để thấy được cách mà hệ điều hành DOS dùng bảng FAT để quản lý việc lưu trữ các file trên đĩa cũng như theo dõi trạng thái các cluster trên vùng dữ liệu, ta xem hình minh họa sau đây.



**Hình 4.10:** Các file FileA và FileB (a) được lưu trên các cluster của đĩa logic (b) và sơ đồ định vị của nó trên bảng FAT (c).

Hình (a) ở trên cho thấy: có hai file, FileA và FileB, FileA có kích thước vừa đủ 4 cluster và được chia thành 4 block, FileB có kích thước nhỏ hơn 4



cluster cũng được chia thành 4 block, trong đó block B<sub>4</sub> mặc dù chưa đủ một cluster nhưng vẫn được chứa vào một cluster. Tức là, hệ điều hành cũng phải dùng đủ 8 cluster để lưu trữ nội dung của hai file FileA và FileB vào đĩa (hình b).

Đoạn FAT trong hình (c) ở trên cho biết các thông tin sau đây:

- Các cluster bị bad, không thể sử dụng: cluster 11 và cluster 13.
- Các cluster còn trống, chưa cấp phát: cluster 2, cluster 3, cluster 5, cluster 8.
- FileA được lưu tại các cluster: 4, 10, 7, **14** (chứa block cuối cùng)
- FileB được lưu tại các cluster: 6, 9, 12, **15** (chứa block cuối cùng)

Như vậy bảng thư mục gốc cho biết cluster đầu tiên chứa FileA là cluster 4, phần tử thứ 4 trong bảng FAT chứa giá trị 10, điều này chứng tỏ cluster 10 là cluster tiếp theo chứa nội dung FileA, phần tử thứ 10 trong bảng FAT chứa giá trị 7, điều này chứng tỏ cluster 7 là cluster tiếp theo chứa nội dung FileA, phần tử thứ 7 trong bảng FAT chứa giá trị FFFh, điều này chứng tỏ cluster 7 là cluster chứa block cuối cùng của FileA.

Các cluster chứa nội dung của một file có thể không liên tiếp nhau, nhưng nó thường nằm rải rác trong một phạm vi hẹp nào đó trên đĩa. Điều này giúp hệ điều hành đọc file được nhanh hơn nhờ tiết kiệm được thời gian duyệt và đọc qua các byte từ đầu đến cuối bảng FAT để dò tìm dãy các cluster chứa nội dung của file. Mặt khác, việc phân bố tập trung các cluster của một file rất phù hợp với các thuật toán đọc đĩa của hệ điều hành. Đối với các file dữ liệu, sau một thời gian kích thước của nó có thể tăng lên, hệ điều hành phải cấp phát thêm các cluster cho nó, các cluster mới này có thể nằm tại các vị trí tách xa các cluster trước đó, dẫn đến các cluster chứa nội dung của một file phân bố rải rác khắp bề mặt đĩa, điều này sẽ làm chậm tốc độ đọc file của hệ điều hành. Các file dữ liệu bị mở, thay đổi, ghi và đóng lại nhiều lần cũng có thể dẫn đến hiện tượng trên. Trên đĩa có thể xuất hiện hiện tượng có nhiều file bị phân bố rải rác khắp bề mặt đĩa, hiện tượng này được gọi là hiện tượng đĩa bị phân mảnh (fragmentary). Các đĩa bị phân mảnh sẽ làm cho tốc độ đọc file trên nó chậm đi rất nhiều. Trong trường hợp này người sử dụng phải thực hiện việc sắp xếp lại các cluster trên đĩa, để các cluster chứa nội dung của một file của tất cả các file trên đĩa được phân bố tập trung hơn, thao tác này được gọi là chống phân mảnh cho đĩa. Hệ điều hành DOS cung cấp nhiều công cụ để người sử dụng thực hiện việc chống phân mảnh cho đĩa cả ở mức ứng dụng và mức lập trình.

Để đọc nội dung của một file trên đĩa dựa vào bảng thư mục gốc và bảng FAT, hệ điều hành thực hiện theo các bước sau đây:

Tìm phần tử trong bảng thư mục gốc chứa thông tin của file cần đọc.

Tại phần tử này, xác định số hiệu của cluster đầu tiên trong dãy các cluster chứa nội dung của file (**giả sử cluster 4**), giá trị này được xem như con trỏ trỏ tới bảng FAT để bắt đầu dò tìm các cluster từ thứ 2 đến cuối cùng trong dãy các cluster chứa nội dung của file cần đọc. Sau đó đọc block dữ liệu đầu tiên của file tại **cluster 4** trên vùng data của đĩa.

Xác định byte tương ứng với **phần tử 4** trong bảng FAT. Đọc giá trị dữ liệu tại **phần tử 4** này, giả sử **giá trị đọc được là 10**. Sau đó đọc block dữ liệu tiếp theo của file tại **cluster 10** trên vùng data của đĩa.

Xác định byte tương ứng với **phần tử 4** trong bảng FAT. Đọc giá trị dữ liệu tại **phần tử 4** này, giả sử **giá trị đọc được là 17**. Sau đó đọc block dữ liệu tiếp theo của file tại **cluster 17** trên vùng data của đĩa.

Xác định byte tương ứng với **phần tử 17** trong bảng FAT, sau đó thực hiện hoàn toàn tương tự như bước 4 cho đến khi đọc được giá trị **FFFh** (với FAT12) hoặc **FFFFh** (với FAT16) tại một phần tử nào đó (**giả sử phần tử 43**) trong bảng FAT thì đọc block dữ liệu cuối cùng của file tại **cluster 43** trên vùng data của đĩa, sau đó dừng lại. Tới đây kết thúc quá trình đọc file.

Chúng ta sẽ hiểu rõ hơn các bước 1 và 2 ở phần mô tả về bảng thư mục gốc trong mục này. Các bước trên chỉ đúng cho việc đọc các file mà thông tin của nó được lưu trữ trên ở các phần tử trong bảng thư mục gốc (được lưu trữ ở thư mục gốc) của đĩa.

Thao tác đọc file của DOS như trên là kém hiệu quả, vì ngoài việc đọc nội dung của file tại các cluster trên vùng data của đĩa hệ điều hành còn phải đọc và phân tích bảng FAT để dò tìm ra dãy các cluster chứa nội dung của một file. Hệ thống file NTFS trong windowsNT/2000 khắc phục điều này bằng cách lưu danh sách các cluster chứa nội dung của một file vào một vị trí cố định nào đó, nên khi đọc file hệ điều hành chỉ cần đọc nội dung của các cluster trên đĩa theo danh sách ở trên, mà không phải tốn thời gian cho việc dò tìm dãy các cluster chứa nội dung của file của hệ thống file FAT trong DOS.

Ngoài ra, nếu DOS có một cơ chế nào đó ghi lại được danh sách các cluster còn trống trên đĩa, thì tốc độ ghi file của hệ điều hành sẽ tăng lên vì hệ điều hành không tốn thời gian cho việc đọc bảng FAT để xác định cluster còn trống. Các hệ thống file của các hệ điều hành sau này như windows98, windowsNT/2000 đã thực hiện được điều này.

Độ rộng của một phần tử trong bảng FAT (12 bit hay 16 bit), quyết định dung lượng đĩa tối đa mà hệ điều hành có thể quản lý được. Nếu hệ điều hành sử dụng FAT12 thì mỗi phần tử trong FAT12 có thể chứa một giá trị

lên đến  $2^{12}$ , đa số trong số này là số hiệu các cluster trên vùng data của đĩa, điều này có nghĩa là trên vùng data của đĩa có tối đa là  $2^{12}$  cluster. Từ đây ta có thể tính được dung lượng đĩa tối đa (byte) mà hệ thống file FAT12 có thể quản lý được là:  $2^{12}$  cluster \* 4 sector/1 cluster \* 512 byte/1 sector (a). Tương tự, dung lượng đĩa tối đa (byte) mà hệ thống file FAT16 có thể quản lý được là:  $2^{16}$  cluster \* 4 sector/1 cluster \* 512 byte/1 sector (b). Rõ ràng với hệ thống file FAT12 thì DOS sẽ quản lý được một không gian đĩa lớn hơn so với FAT12 (theo a và b).

Windows98 sử dụng hệ thống file FAT32 và nó cho phép có tới 6 sector trên một cluster, nên nó có thể quản lý được một không gian đĩa lớn hơn nhiều lần ( $2^{32}$  cluster \* 6 sector/1 cluster \* 512 byte/1 sector) so với DOS. Nếu một file có kích thước là 50 sector, thì trong DOS file được chia thành 13 block (có 4 sector trong 1 block  $\equiv$  cluster) còn trong windows98 file được chia thành 9 block (có 6 sector trong 1 block  $\equiv$  cluster). Tức là, trong DOS file này được chứa ở 13 cluster trên đĩa (dãy các cluster chứa file gồm 13 phần tử) còn trong windows98 file này được chứa trong 9 cluster (dãy các cluster chứa file gồm 9 phần tử). Điều này cho thấy file này sẽ được đọc nhanh hơn trong windows98, vì chỉ cần đọc 9 lần thay vì phải đọc 13 lần như trong DOS. Chưa kể, để đọc file các hệ điều hành phải phân tích bảng FAT để dò ra dãy các cluster chứa nội dung của file. Như vậy, hệ thống file của windows98 quản lý được một không gian đĩa lớn hơn và có tốc độ đọc file nhanh hơn, so với hệ thống file của DOS.

Để ghi một file vào đĩa, thì trong nhiều thao tác phải thực hiện hệ điều hành phải thực hiện việc đọc nội dung của các phần tử trong bảng FAT để tìm phần tử chứa giá trị 0, để ghi một block file vào cluster tương ứng trên vùng data. Trong khi đọc giá trị của các phần tử trong bảng FAT hệ điều hành có thể đọc được giá trị FF7h hoặc FFF7h, dấu hiệu của bad cluster, trong trường hợp này hệ điều hành sẽ không ghi file vào cluster tương ứng với phần tử này, và hệ điều hành sẽ tìm đọc một phần tử khác. Như vậy các bad cluster trên đĩa sẽ làm chậm tốc độ ghi file của hệ điều hành. Đây là một trong các hạn chế của các hệ thống file FAT. Các hệ thống file khác, NTFS chẳng hạn, khắc phục điều này bằng cách tạo ra một danh sách riêng để theo dõi các cluster bị bad và khi tìm cluster trống để ghi file hệ điều hành sẽ không đọc các phần tử trong bảng FAT tương ứng với các cluster này.

Việc đọc nội dung của một phần tử trong FAT16 chỉ đơn giản là đọc nội dung của 2 byte (1 word), trong khi đó việc đọc nội dung của một phần tử trong FAT12 sẽ phức tạp hơn vì 1.5 byte không phải là kiểu dữ liệu chuẩn của ngôn ngữ máy và DOS. Do đó, DOS phải gộp 2 phần tử liên tiếp để có được 3 byte sau đó đọc hết 3 byte và phân tích để có được nội dung của 2

phần tử liên tiếp trong FAT12.

➤ **Root Directory: Để quản lý thông tin của các file và các thư mục (thư mục con của thư mục gốc) đang được lưu trữ trên thư mục gốc của đĩa mềm hoặc đĩa logic trên đĩa cứng, hệ điều hành DOS sử dụng bảng thư mục gốc (root directory).**

Bảng thư mục gốc gồm nhiều phần tử (entry/mục vào), số lượng phần tử trong bảng thư mục gốc được DOS quy định trước trong quá trình Format đĩa và được ghi tại word tại offset 11h trong boot sector, giá trị này không thể thay đổi. Do đó, tổng số file và thư mục con mà người sử dụng có thể chứa trên thư mục gốc của đĩa là có giới hạn. Đây là một hạn chế của DOS. Trong hệ thống file FAT32 và NTFS số phần tử trong bảng thư mục gốc không bị giới hạn, có thể thay đổi được và có thể được định vị tại một vị trí bất kỳ trên đĩa hoặc chứa trong một tập tin nào đó.

Mỗi phần tử trong bảng thư mục gốc dùng để chứa thông tin về một file hay thư mục nào đó đang được lưu trên thư mục gốc của đĩa. Khi có một file hoặc một thư mục nào đó được tạo ra trên thư mục gốc của đĩa thì hệ điều hành dùng một phần tử trong bảng thư mục gốc để chứa các thông tin liên quan của nó, khi một file hoặc thư mục bị xoá/ di chuyển khỏi thư mục gốc thì hệ điều hành sẽ thu hồi lại phần tử này để chuẩn bị cấp cho các file thư mục khác sau này.

Một phần tử trong thư mục gốc dài 32 byte, chứa các thông tin sau:

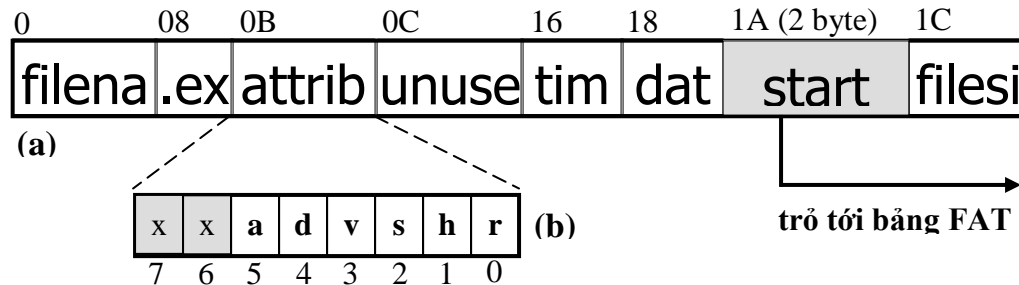
Offset	Nội dung	Độ lớn
00h	Tên chính của file (filename)	8 byte
08h	Phần mở rộng của tên File (.Ext)	3 byte
0Bh	Thuộc tính file (attribute)	1 byte
0Ch	Dự trữ, chưa được sử dụng (Unused)	<b>10 byte</b>
16h	Giờ thay đổi tập tin cuối cùng (time)	2 byte
18h	Ngày thay đổi tập tin cuối cùng (date)	2 byte
1Ah	<b>Cluster đầu tiên của File (start cluster)</b>	<b>2 byte</b>
1Ch	Kích thước của File (filesize)	4 byte

**Bảng 4.5:** Các trường trong một entry của bảng thư mục gốc

Hình vẽ 4.11 sau đây cho thấy rõ hơn về cấu trúc của một phần tử trong bảng thư mục gốc của DOS. Và byte thuộc tính của phần tử này:

- Sở dĩ DOS quy định tên file/tên thư mục dài không quá 8 ký tự và phần mở rộng tên file dài không quá 3 ký tự (tên theo chuẩn 8.3) vì DOS chỉ dùng 8 byte

cho trường tên file (filename) và 3 byte cho trường tên mở rộng (.ext). Nếu người sử dụng tạo file/ thư mục với tên không đủ 8 kí tự thì DOS tự động thêm vào các kí tự trắng để cho đủ 8 kí tự, ngược lại DOS tự động bỏ bớt các kí tự cuối chỉ giữ lại 8 kí tự đầu. Điều này cũng đúng cho phần mở rộng của tên file.



**Hình 4.11:** Một phần tử trong bảng thư mục gốc (a) và byte thuộc tính (b)

Byte đầu tiên (offset 00) của một phần tử trong thư mục gốc còn được gọi là byte trạng thái, byte này có thể chứa một trong các giá trị đặc biệt sau đây:

- 0: cho biết phần tử này chưa được sử dụng.
- E5h: cho biết phần tử này là của một file đã được tạo nhưng đã bị xóa.
- 05h: cho biết kí tự đầu tiên của tên file này thực tế là E5h. Nếu người sử dụng cố tình tạo ra một file có tên bắt đầu là kí tự có mã ascii là E5h thì hệ điều hành sẽ tự động thay bằng kí tự có mã là 05h, để phân biệt file này với các file đã bị xóa.
- 2Eh (kí tự dấu chấm “.”): cho biết phần tử này chứa thông tin của một thư mục con, nếu byte thứ 2 cũng chứa giá trị 2Eh (hai dấu chấm liên tiếp “..”) thì trường start cluster sẽ chứa số hiệu cluster đầu tiên của thư mục cha, nếu là thư mục gốc thì là 0000h.

Nếu bằng cách nào đó người lập trình đưa được một giá trị 1 byte không thuộc một trong các giá trị trên và không phải là các chữ cái tên file thông thường, vào byte đầu tiên của phần tử trong bảng thư mục gốc đang cấp phát cho một file/thư mục nào đó, thì DOS và các tiện ích trên DOS không thể nhận biết các file/thư mục này, và không thể thực hiện các thao tác Dir, Del, vv trên nó. Đây là một cách để bảo vệ các file/thư mục của người sử dụng trên đĩa.

DOS dùng 1 byte, sau 2 phần tên file, để ghi các thuộc tính của file, tuy nhiên DOS chỉ dùng 6 bit từ 0 đến 5 của byte này để ghi 5 thuộc tính lần lượt là: chỉ đọc, ẩn, hệ thống, nhãn đĩa, thư mục con và lưu trữ (trong hình 4.xx.b ở trên 2 bit chưa sử dụng được đánh dấu x):

- Thuộc tính chỉ đọc (r: read only): Nếu bit 0 của byte thuộc tính

bằng 1, thì tập tin tương ứng có thuộc tính chỉ đọc. Một tập tin có thuộc tính chỉ đọc sẽ không bị thay đổi nội dung và không bị xóa.

- Thuộc tính ẩn (h: hidden): Nếu bit 1 của byte thuộc tính bằng 1, thì tập tin tương ứng có thuộc tính ẩn. Một tập tin có thuộc tính ẩn thì các lệnh DOS thông thường như Edit, Del, Dir, Tree, ... sẽ không tác động được đến nó.

- Thuộc tính hệ thống (s: system): Nếu bit 2 của byte thuộc tính bằng 1, thì tập tin tương ứng có thuộc tính hệ thống. Một tập tin có thuộc tính hệ thống tương tự như tập tin có thuộc tính ẩn hoặc vừa ẩn vừa hệ thống. Thuộc tính hệ thống chỉ có ý nghĩa kế thừa, nó không có ý nghĩa trong hệ điều hành DOS.

- Thuộc tính nhãn đĩa (v: volume): Nếu bit 3 của byte thuộc tính bằng 1, thì phần tử này chứa nhãn nhận dạng đĩa, được lưu tại trường filename và trường Ext. Phần tử này chỉ được DOS nhận biết nếu nó nằm trên thư mục gốc. Trong trường hợp này chỉ có trường Date và Time là được sử dụng. Trường start cluster và trường filesize chứa giá trị 0.

- Thuộc tính thư mục con (d: subdirectory): Nếu bit 4 của byte thuộc tính bằng 1, thì phần tử này chứa các thông tin về thư mục con của thư mục gốc trên đĩa. Đối với DOS thư mục con là một tập tin chứa dữ liệu thông thường, nó có một thuộc tính đặc biệt, đó là thuộc tính d.

- Thuộc tính lưu trữ (a:archive): thuộc tính này dùng để trợ giúp cho việc tạo backup của các tập tin trên đĩa cứng. Bit này = 0 đối với tất cả các tập tin chưa bị sửa đổi kể từ lần backup gần đây nhất. Như vậy trong lần tạo backup sau DOS chỉ cần cập nhật cho các tập tin có bit này bằng 1. Tóm lại, lúc đầu bit a = 0, sau đó nếu có bất kỳ sự thay đổi nào trong nội dung của file thì bit này sẽ được chuyển lên thành bằng 1.

- 10 byte bắt đầu từ offset 0Ch chưa được DOS sử dụng, Microsoft dự trữ vùng này cho các mục đích khác sau này. Hệ điều hành windows98 sử dụng tất cả 10 byte này. Việc dự trữ 10 byte này tạo thành một kẽ hở của DOS, các đoạn code virus có thể định vị ở vùng này mà hệ điều hành không thể phát hiện được.

- Trường Date và trường Time kết hợp với nhau để lưu chính xác ngày giờ tạo ra file/ thư mục hay ngày giờ file/ thư mục được thay đổi gần đây nhất.

- Trường quan trọng nhất trong của một phần tử trong bảng thư mục gốc là trường start cluster, dài 2 byte, bắt đầu tại offset 1Ah. Nó chứa số hiệu của cluster đầu tiên trong dãy các cluster chứa nội dung của file tương ứng với phần tử này. Do đó trong các thao tác đọc file của hệ điều hành, trường này được xem như một con trỏ trỏ tới bảng FAT để dò tìm dãy các cluster chứa nội dung của một file.

- Trường filesize cho biết kích thước của file tính theo byte. Nếu một

phần tử trong bảng thư mục gốc được dùng cho một thư mục con nào đó thì trường filesize này chứa giá trị 0. Kích thước của thư mục con chỉ được biết nhờ lần theo chuỗi số hiệu cluster tương ứng trong bảng FAT.

Khi người sử dụng xóa một file trên đĩa hệ điều hành không xóa nội dung của file tại các cluster trên vùng data, không xóa dãy các cluster chứa file trong bảng FAT, thậm chí không xóa cluster đầu tiên trong dãy các cluster chứa file tại phần tử tương ứng với file trong bảng thư mục gốc mà hệ điều hành chỉ thay kí tự đầu tiên của tên file tại phần tử trong bảng thư mục gốc bằng giá trị E5h. Do đó, sau khi đã xóa một file thì hệ điều hành có thể khôi phục lại được file này, bằng cách thay kí tự mã E5h ở byte đầu tiên của tên file bằng một kí tự khác. Trường hợp không khôi phục được là do sau một thời gian, hệ điều hành đã sử dụng phần tử trong thư mục gốc, các phần tử trong bảng FAT và các cluster trên vùng data của file đã bị xóa, cấp phát cho các file mới sau này. Khi duyệt bảng thư mục gốc gặp các phần tử có byte đầu bằng E5h hệ điều hành biết đây là phần tử của file đã bị xóa nên không in ra màn hình. Điều vừa trình bày trên đây hoàn toàn đúng với trường hợp của các thư mục con trên đĩa.

Để ghi một file vào thư mục gốc của đĩa (thông tin của file chứa ở một phần tử trong bảng thư mục gốc) hệ điều hành DOS thực hiện các bước sau đây:

1. Tìm một phần tử trong bảng thư mục gốc chưa sử dụng, đó là phần tử mà byte đầu tiên của nó chứa giá trị 00. **Giả sử tìm được phần tử thứ 105.**
2. Ghi tên file, phần mở rộng, thuộc tính của file, ngày giờ tạo file vào các trường tương ứng tại **phần tử 105** trong bảng thư mục gốc.
3. Tìm một entry trong bảng FAT chứa giá trị 000h, giả sử tìm được **entry 207**, điều này có nghĩa **cluster 207** trên vùng data còn trống.
4. Ghi số hiệu của entry này, **entry 207**, vào **trường start cluster** tại offset 1Ah của **phần tử 107** trong bảng thư mục gốc.
5. Ghi block đầu tiên của file vào **cluster 107** trên vùng data. Nếu nội dung của tập tin chứa vừa đủ trong 1 cluster, thì DOS sẽ thực hiện bước cuối cùng (bước 9), ngược lại DOS tiếp tục thực hiện bước 6.
6. Tiếp tục tìm một entry trong bảng FAT chứa giá trị 000h, giả sử tìm được **entry 215**, điều này có nghĩa **cluster 215** trên vùng data còn trống.
7. Ghi **giá trị 215** vào **entry 207** trong bảng FAT và ghi block thứ hai của file vào **cluster 215** trên vùng data.
8. Lặp lại các bước 6 và 7 cho đến khi ghi hết các block của file vào các cluster trên vùng data. Giả sử block cuối cùng của file được ghi vào **cluster 302** trên vùng data, tức là entry cuối cùng được tìm thấy (chứa giá

trị 00h) là **entry 302**.

9. Bước cuối cùng: ghi **giá trị FFFh** vào **entry 107** hoặc vào **entry 302**.

10. Tính kích thước của tập tin và ghi vào **trường filesize** của **phần tử 105** trong bảng thư mục gốc.

➤ Thư mục con (subdirectory): Như đã biết, bảng thư mục gốc của DOS định vị tại một vị trí cố định trên đĩa logic, sau 2 bảng FAT, số lượng phần tử trong bảng thư mục gốc là cố định, không thể mở rộng được, và được DOS quy định trong quá trình định dạng đĩa. Đó là những hạn chế về cấu trúc và tổ chức bảng thư mục gốc của DOS. Cụ thể là người sử dụng không thể chứa quá nhiều file, thư mục trên thư mục gốc và bảng thư mục gốc dễ bị virus tấn công. Hệ điều hành DOS đưa ra khái niệm thư mục con để khắc phục một phần những hạn chế trên.

Đối với người sử dụng, thư mục con là những thư mục nằm trên thư mục gốc của đĩa, trên đĩa chỉ có một thư mục gốc nhưng có thể có nhiều thư mục con, trong thư mục con có thể chứa nhiều (không giới hạn) file và thư mục con khác, gắn liền với thư mục con là thư mục cha của nó, thư mục cha có thể là thư mục gốc hoặc một thư mục con khác. Nhưng đối với DOS, thư mục con là một file đặc biệt, file này có thuộc tính thư mục con, byte thuộc tính có giá trị 00010000 (16), và có trường filesize = 0.

Về mặt hệ thống, thư mục con có các điểm khác sau đây so với thư mục gốc:

- Hệ điều hành lưu trữ nó giống như lưu trữ các file khác trên đĩa. Tức là, muốn đọc được thư mục con hệ điều hành phải lần theo dấu vết của nó trong bảng FAT.

- Bảng thư mục của nó có số lượng phần tử không giới hạn, có thể tăng lên hay giảm xuống tùy thuộc vào số lượng file và thư mục chứa trong nó. Nhờ vậy mà người sử dụng có thể chứa nhiều file thư mục trong một thư mục con trên đĩa. Số lượng phần tử trong bảng thư mục của các thư mục con chỉ bị giới hạn bởi dung lượng đĩa và kích thước các file thư mục chứa trong nó.

- Bảng thư mục của nó có thể định vị tại một vị trí bất kỳ trên vùng data của đĩa. Có thể xem đây là một hạn chế của thư mục con, vì nếu tạo quá nhiều thư mục con trên đĩa thì bảng thư mục của các thư mục con sẽ chiếm hết nhiều không gian đĩa trên vùng data. Do đó hệ điều hành không khuyến khích tạo quá nhiều thư mục con trên các đĩa mềm. Virus khó có thể tấn công bảng thư mục của thư mục con vì nó không cố định.

- Muốn biết được kích thước của thư mục con hệ điều hành phải tính toán từ kích thước của tất cả các file trong thư mục con, hệ điều hành dựa vào bảng thư mục của thư mục con và bảng FAT để thực hiện việc tính toán này.

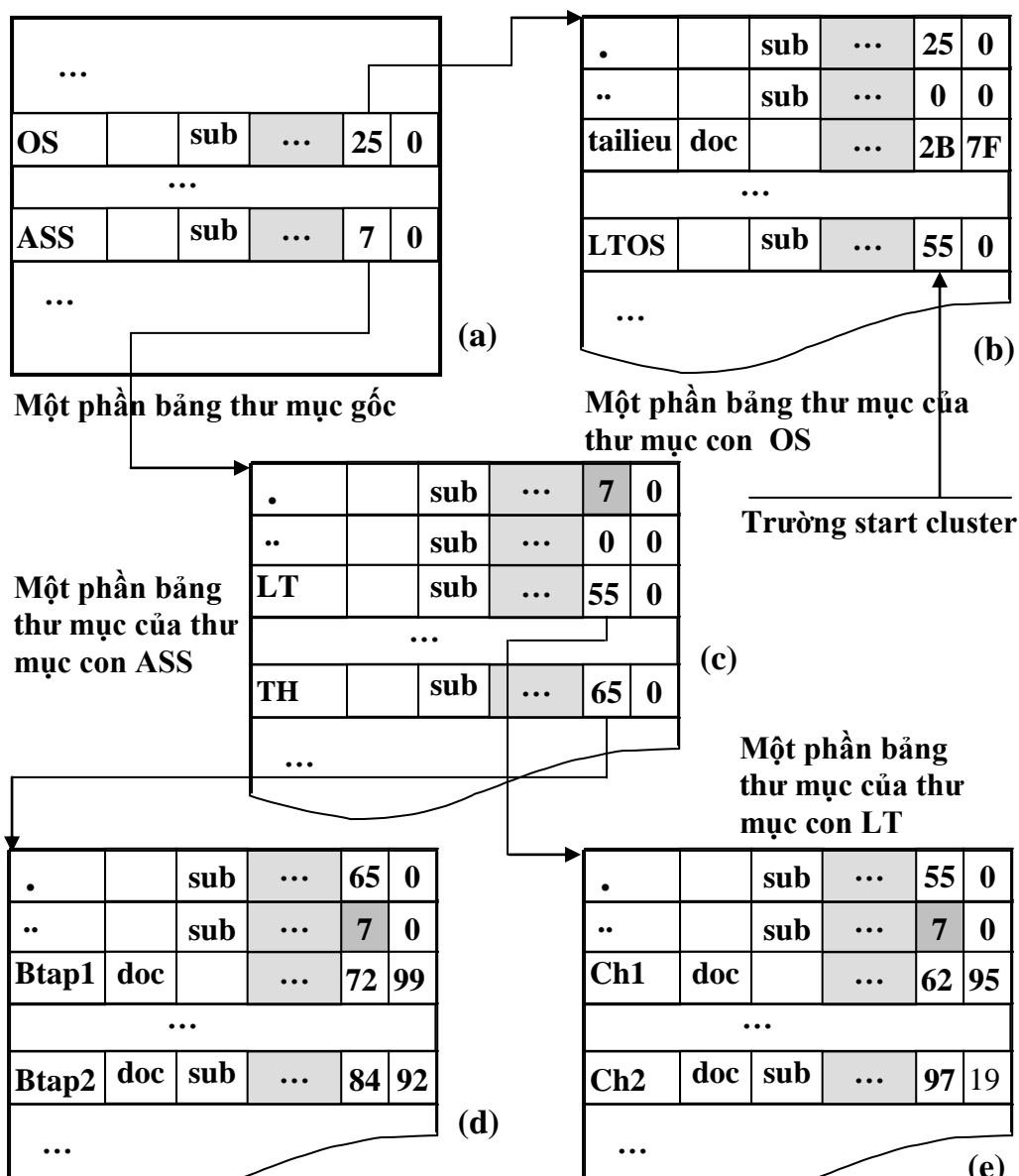
Cấu trúc của một phần tử trong bảng thư mục của thư mục con tương tự cấu



trúc của một phần tử trong bảng thư mục gốc. Đa số các phần tử trong bảng thư mục gốc của đĩa hay phần tử trong bảng thư mục của thư mục con, chứa thông tin của một thư mục (con), đều có trường Filename là tên của thư mục (con), trường Attribute = 16, trường Start cluster = cluster đầu tiên của thư mục (con), trường Filesize = 0, ... Chỉ có hai phần tử đầu tiên trong bảng thư mục của các thư mục con là có chứa các giá trị đặc biệt.

Ngay sau khi có một thư mục con được tạo ra, hệ điều hành tạo ngay bảng thư mục cho nó và khởi tạo 2 phần tử đầu tiên trong bảng thư mục này:

- Phần tử thứ nhất, chứa mã ascii của kí tự dấu chấm (.) ở byte đầu tiên của trường filename, phần tử này chỉ đến chính thư mục hiện hành. Trường Start cluster cho biết cluster bắt đầu của thư mục này.
- Phần tử thứ hai, chứa 2 mã ascii của 2 kí tự dấu chấm (..) ở 2 byte đầu tiên của trường filename, phần tử này chỉ đến thư mục cha của nó. Trường Start cluster cho biết cluster bắt đầu của thư mục cha của nó. Nhưng nếu cha của nó là thư mục gốc thì trường này chứa giá trị 0.



Để thấy rõ hơn mối quan hệ giữa bảng thư mục gốc và bản thư mục của các thư mục con, cũng như các phần tử trong 2 bảng thư mục này, ta hãy xem sơ đồ minh họa 4.12 trên đây. Trong sơ đồ này ta giả sử: trên thư mục gốc có hai thư mục con là ASS và OS, trong thư mục ASS có hai thư mục con LT và TH, trong OS có chứa tập tin tailieu.txt và thư mục con LTOS, trong LT có chứa 2 tập tin ch1.txt và ch2.txt, trong TH có chứa 2 tập tin btap1.txt và btap2.txt.

Hình 4.12.a ở trên cho thấy trên thư mục gốc có 2 thư mục con OS và ASS, tổ chức của bảng thư mục con của thư mục OS được lưu trữ bắt đầu tại block 25, tổ chức của bảng thư mục con của thư mục ASS được lưu trữ bắt đầu tại block 7. Hình 4.12.b cho thấy thư mục cha của thư mục OS là thư mục gốc, phần tử có tên là hai dấu chấm “..” chứa giá trị 0 ở trường start cluster, tập tin tailieu.doc được lưu trữ bắt đầu tại cluster 2Bh và có kích thước là 7Fh, tổ chức bảng thư mục con của thư mục LTOS được lưu trữ bắt đầu tại cluster 55. Hình 4.12.c & 4.12.d cho thấy thư mục TH là thư mục con của thư mục ASS, bảng thư mục con của thư mục TH lưu trữ bắt đầu tại cluster 65 (phần tử có tên là một dấu chấm “.” chứa giá trị 65 ở trường start cluster), và tổ chức bảng thư mục của thư mục cha của nó, thư mục ASS, được lưu trữ bắt đầu tại cluster 7 (phần tử có tên là hai dấu chấm “..” chứa giá trị 7 ở trường start cluster), ...

Nếu một tập tin bị giảm kích thước thì DOS sẽ giải phóng ngay các cluster đĩa mà tập tin không còn sử dụng nữa. Nhưng các cluster chứa thư mục con chỉ được giải phóng khi tất cả các tập tin thư mục trong thư mục con này đã bị xóa hết ra khỏi đĩa.

➤ Quá trình khởi động máy tính với hệ điều hành DOS: **Quá trình khởi động máy tính là quá trình từ khi máy tính được cung cấp nguồn điện cho đến khi màn hình xuất hiện dấu nhắc hệ điều hành. Quá trình này được chia thành 2 giai đoạn, giai đoạn thứ nhất tạm gọi là giai đoạn khởi động máy, giai đoạn này chủ yếu do chương trình POST trong BIOS thực hiện. Giai đoạn thứ hai tạm gọi là giai đoạn tìm nạp phần lõi của hệ điều hành vào RAM, giai đoạn này chủ yếu do BIOS và Boot sector/master boot record thực hiện.**

### **Giai đoạn khởi động máy tính:**

(Bắt đầu từ khi máy tính được cung cấp nguồn điện)

1. Các thanh ghi segment được gán giá trị 0FFFFh, các thanh ghi còn lại bằng 0.
2. Cặp thanh ghi **CS:IP** chứa giá trị 0FFFFh:0000h, địa chỉ này chứa một lệnh nhảy xa (JMP Far) chuyển quyền điều khiển đến một đoạn chương trình ở đầu ROM\_BIOS. Đoạn chương trình này sẽ thực hiện quá trình POST máy.
3. POST sẽ lần lượt kiểm tra các thanh ghi, bộ nhớ khởi tạo các chip điều khiển DMA, bộ điều khiển ngắt, đĩa, ... . Nếu có thiết bị nào bị lỗi thì thông báo lỗi lên màn hình và dừng quá trình khởi động, nếu không thì đoạn code ở cuối chương trình POST sẽ đọc nội dung của boot sector của đĩa mềm/ hoặc master boot record của đĩa cứng vào RAM tại địa chỉ 0000:7C00h và bắt đầu quá trình khởi động hệ điều hành (tìm và nạp hệ điều hành vào bộ nhớ RAM). Nếu không tìm thấy boot sector hoặc master boot record thì sẽ xuất hiện thông báo lỗi hoặc dừng quá trình khởi động.

### **Giai đoạn tìm nạp hệ điều hành vào bộ nhớ:**

(Kết thúc khi màn hình xuất hiện dấu nhắc hệ điều hành)

4. Nếu đĩa khởi động là đĩa mềm: chuyển sang bước 7.
5. Nếu đĩa khởi động là đĩa cứng: dò tìm tại offset 00 của các phần tử trong bảng partition giá trị 80h. Nếu tìm thấy thì partition tương ứng là partition active (partition chứa sector khởi động), chuyển sang bước 7. Nếu không tìm được giá trị 80h tại offset 00 trên tất cả các phần tử trong bảng partition thì xuất hiện thông báo lỗi hoặc dừng quá trình khởi động.
6. Boot code trong master boot record sẽ nạp boot sector (sector đầu tiên) của partition active vào RAM tại địa chỉ 0000:7C00h đồng thời với việc chuyển chính nó đi nơi khác.
7. Quyền điều khiển quá trình khởi động tiếp theo được chuyển cho boot sector, boot sector thực hiện ngay lệnh nhảy (JMP) tại offset đầu tiên của nó, để nhảy đến thực hiện chương trình bootstrap loader ở cuối bootsector.
8. Bootstrap loader thực hiện việc tìm và nạp IO.SYS vào bộ nhớ, sau đó trao quyền điều khiển quá trình khởi động tiếp theo cho IO.SYS. Nếu không tìm thấy sẽ xuất hiện thông báo lỗi và/hoặc dừng quá trình khởi động. Thông thường phần tử đầu tiên trong bảng thư mục gốc chứa thông tin của tập tin IO.SYS.
9. IO.SYS thực hiện lần lượt việc tìm và nạp MSDOS.SYS, các trình điều khiển thiết bị khai báo trong Config.sys (nếu tìm thấy config.sys trên thư mục gốc của đĩa khởi động), phần thường trực của Command.com, các lệnh

khai báo trong Autoexec.bat (nếu tìm thấy autoexec.bat trên thư mục gốc của đĩa khởi động). Bước này cũng có thể xuất hiện lỗi và dừng quá trình khởi động nếu MSDOS.SYS và/hoặc Command.com bị xóa hoặc bị lỗi.

Nếu hệ điều hành không thành công trong việc nạp các trình điều khiển thiết bị được khai báo trong config.sys thì quá trình khởi động cũng bị dừng lại, lỗi này là do một hoặc tất cả các trình điều khiển thiết bị nói trên bị hỏng (thường do bị nhiễm virus). Điều này cũng đúng với các lệnh trong autoexec.bat.

10. Kết thúc quá trình khởi động bằng việc xuất hiện dấu nhắc hệ điều hành lên màn hình.

Đối với người sử dụng thông thường thì việc chuyển từ giai đoạn khởi động máy sang giai đoạn nạp hệ điều hành, được đánh dấu bằng thông báo: Starting MSDOS .... trên màn hình. Xét về mặt bản chất thì điều này không chính xác.

Người sử dụng có thể bỏ qua việc thực hiện các khai báo trong Config.sys và Autoexec.bat trong quá trình khởi động ở bước 9, bằng cách gõ phím F5 khi màn hình vừa xuất hiện thông báo: Starting MSDOS .... Người sử dụng cũng có thể chỉ định thực hiện hay không thực hiện một khai báo nào đó trong Config.sys và Autoexec.bat trong quá trình khởi động ở bước 9, bằng cách gõ phím F8 khi màn hình vừa xuất hiện thông báo: Starting MSDOS .... Điều này cũng có thể áp dụng trong quá trình khởi động máy tính với hệ điều hành windows9x.

Qua việc phân tích quá trình khởi động máy tính ở trên ta có thể biết được nguyên nhân và cách khắc phục một số lỗi xảy ra trong quá trình khởi động máy và nạp hệ điều hành vào bộ nhớ RAM.

## **Tổ chức bảng thư mục gốc của Windows98**

*Trong phần này chúng tôi chỉ trình bày về tổ chức của bảng thư mục gốc của windows98 để các bạn thấy được cách tổ chức lưu trữ tên file dài của windows98 và sự tương thích của tên file dài trong DOS và các windows trước đó, mà không trình bày về cấu trúc của hệ thống FAT32. Vì, FAT32 tương thích với FAT16, chỉ có một điểm khác nhỏ cần chú ý ở đây là: mỗi phần tử trong FAT chiếm 2 byte, do đó theo lý thuyết mỗi phần tử có thể chứa trong nó một giá trị lên đến  $65536 (2^{16})$  hay bảng FAT có 65536 phần tử, nhờ vậy mà windows98 quản lý được một không gian đĩa cứng lớn hơn nhiều so với hệ điều hành DOS. Còn nhiều khác biệt nữa của FAT32 so với FAT12&FAT16 chúng tôi đã chỉ ra ở mục IV.3 ở trên.*

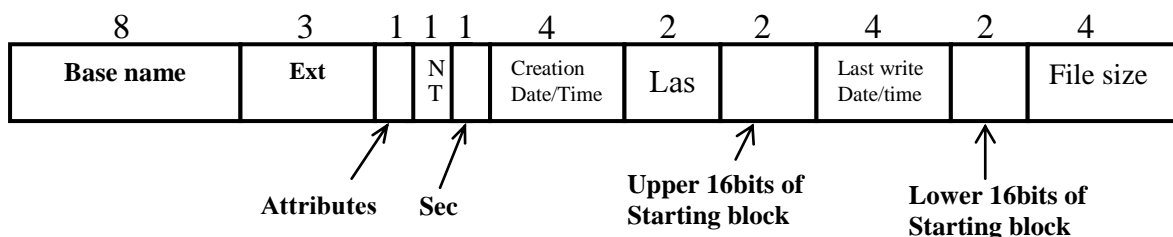
Phiên bản windows95 thứ nhất và các phiên bản trước đó đều sử dụng hệ thống file của hệ điều hành DOS (tên file theo chuẩn 8.3 với FAT12 hoặc FAT16), bắt đầu từ phiên bản Windows95 thứ 2 đến Windows98, hệ điều hành windows sử dụng hệ thống tên file dài với FAT32, nhờ đó mà nó có thể quản lý được những partition có kích thước lớn hơn 2GB và những đĩa cứng lớn hơn 8GB. Trong phần

này chúng ta sẽ xem xét hệ thống file Windows98.

Với tên file dài và FAT32 hệ điều hành Windows đã mang lại cho người sử dụng nhiều thuận lợi hơn so với tên file 8.3 và FAT12/FAT16 của hệ điều hành DOS. Nhưng quan trọng hơn, windows98 vẫn cho phép truy xuất đến các file được tạo ra từ các hệ điều hành như DOS, windows3.1, và phiên bản windows95 thứ 1. Và ngược lại trong môi trường hệ điều hành DOS hệ thống vẫn có thể truy xuất đến các file (kể cả các file có tên file dài) được tạo ra từ Windows 98. Phải nói rằng đây là một thành công lớn của hệ điều hành windows98. Windows98 đã phát triển hệ thống tên file dài và FAT32 dựa trên nền tảng của hệ điều hành DOS và tương thích với hệ điều hành DOS nhờ vậy mà một file do nó tạo ra có thể được truy xuất trên cả 2 hệ thống: Windows 98 và DOS. Sau đây chúng ta sẽ xem xét kỹ hơn về điều này.

Windows 98 tổ chức cấu trúc của bảng thư mục, hay chính xác hơn là cấu trúc của một phần tử trong bảng thư mục gốc, hoàn toàn tương thích với cấu trúc của một phần tử trong bảng thư mục gốc của hệ điều hành DOS. Như vậy bảng thư mục của Windows 98 cũng là một danh sách gồm nhiều phần tử, mỗi phần tử dài 32 byte, nhưng không giống với DOS (mỗi phần tử chứa thông tin về 1 file hay thư mục đang lưu trữ trên thư trong mục) ở đây Windows 98 có thể sử dụng nhiều phần tử trong bảng thư mục để chứa thông tin về một file đang lưu trữ trong thư mục hay trên đĩa. Đây là sự khác biệt lớn giữa cấu trúc bảng thư mục của Windows98 và MS\_DOS và cũng chính nhờ điều này trong MS\_DOS ta vẫn có thể truy xuất đến các file được tạo ra từ Windows 98.

Hình vẽ sau đây là cấu trúc của một phần tử trong bảng thư mục của windows98.



**Hình 4.13** Phần tử thư mục mở rộng của MS-DOS sử dụng trong windows 98

Điều ghi nhận đầu tiên ở cấu trúc này là 10 byte bắt đầu tại offset 0Ch mà hệ điều hành DOS chưa sử dụng nay đã được Windows98 sử dụng, Windows98 dùng 10 byte này để tạo ra 5 trường mới: NT, Sec, Creation Date/Time (ngày giờ tạo file/directory), Last Access (thời điểm file/directory được truy xuất gần đây nhất) và Upper 16 bit of Starting block (16 bit cao của block đầu tiên).

- Trường NT: tương thích với WindowsNT, trong trường hợp hiển thị tên file đúng như khi tạo file ban đầu (chỉ dùng với windowsNT).

- Trường *Sec*: kết hợp cùng với trường *Creation Date/Time* để lưu chính xác thời điểm tạo ra file bao gồm ngày giờ và giây (chính xác đến 10msec).
- Trường *Last access*: lưu trữ thời gian truy cập đến file cuối cùng trước đó.
- Trường *Upper 16 bit of Starting block* kết hợp cùng với trường *Lower 16 bit of Starting block* (16 bit thấp của block đầu tiên): để lưu số hiệu của block đầu tiên trong dãy các block chứa nội dung của File tương ứng.

**Chú ý:** Vì windows98 sử dụng 32 bit(4 byte) để định danh các cluster trên đĩa nên trong entry phải có đủ 4 byte để chứa đủ giá trị (4 byte) của các cluster trên đĩa. Trong trường hợp này windows98 sử dụng kết hợp 2 trường 2 byte là Upper 16 bit of Starting block (để chứa 2 byte cao của số hiệu cluster bắt đầu chứa các block file) và Lower 16 bit of Starting block (để chứa 2 byte thấp của số hiệu cluster bắt đầu chứa các block file).

Để đảm bảo sự tương thích về tên file, đặc biệt là những tên file dài, trong cả 2 hệ thống DOS và Windows98, Windows98 sử dụng hai tên file cho một file: Một tên file dài để tương thích với Windows98 và WindowsNT và một tên file theo chuẩn 8.3 để tương thích với hệ điều hành DOS, các file có thể được truy cập theo một trong hai tên file này.

Như vậy khi một file được tạo ra mà tên file của nó không tuân theo chuẩn đặt tên của DOS thì Windows98 sẽ tạo thêm một tên file nữa theo chuẩn về tên file của hệ điều hành DOS. Windows98 thực hiện điều này như sau:

- Lấy 6 ký tự đầu tiên (không kể ký tự trắng và những dấu chấm phụ) của tên dài, chuyển chúng sang dạng viết hoa (nếu cần).
- Nối thêm 2 ký tự “~1” vào sau 6 ký tự trên để có được tên file theo chuẩn của hệ điều hành DOS, nếu tên file này đã tồn tại thì windows98 sẽ dùng 2 ký tự “~2”, và cứ như thế windows98 có thể thêm vào các ký tự “~3”, “~4”, vv sao cho đảm bảo tên file không trùng nhau trong cùng một thư mục.

Ví dụ: Có 2 tên file dài: **Lap trình huong doi tuong** (file 1)

**Lap trình huong thanh phan** (file 2)

Thì windows98 tạo ra 2 tên file theo chuẩn MS\_DOS như sau:

**LAPTRI~1** (file 1)

**LAPTRI~2** (file 2)

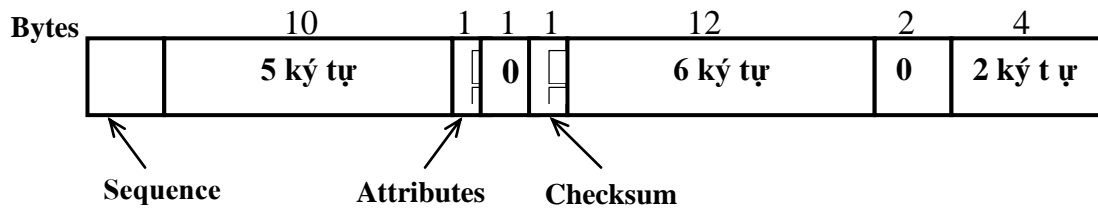
Để lưu trữ thông tin về một file có tên file dài hệ điều hành windows98 sử dụng các entry liên tiếp nhau trong bảng thư mục, các entry này được bố trí như sau:

- Một entry đầu tiên dùng để lưu trữ tên file theo chuẩn của DOS và các thông tin khác, cấu trúc của entry này hoàn toàn tương tự như hình trên.

- Một hoặc nhiều entry kế tiếp sau đó để lưu tên file dài, số lượng các entry này phụ thuộc vào số kí tự của tên file (mỗi entry chứa được 13 kí tự Unicode= 26byte).

Ví dụ: một file có tên file gồm 28 kí tự thì windows98 sẽ dùng 4 entry: một entry dùng để lưu tên file theo chuẩn MS\_DOS, 3 entry còn lại dùng để lưu tên file dài, trong đó 2 entry đầu chứa 26 kí tự (13x2) đầu, entry cuối cùng chứa 2 kí tự còn lại của tên file.

Hình 4.14 cho thấy cấu trúc của một entry trong bảng thư mục gốc của windows98 dùng cho tên file dài:



**Hnh 4.14** Một entry cho một tên file dài của windows 98

- Trường Sequence: chiếm 1 byte dùng để ghi thứ tự của các entry chứa tên file dài, nếu là entry cuối cùng thì thứ tự này sẽ được cộng thêm 64 đơn vị để đánh dấu đây là entry cuối cùng của một file, theo ví dụ trên thì trường Sequence của các entry lần lượt là: 1, 2 và **67** (3+64).
- Trường Attributes: chiếm 1 byte, chứa giá trị 0Fh, đây là dấu hiệu để phân biệt giữa entry chứa tên file theo chuẩn DOS với các entry chứa tên file dài. Hệ điều hành DOS và các tiện ích về file của DOS sẽ bỏ qua (không nhận ra) các entry này.
- Trường Checksum: chiếm 1 byte, dùng để ghi nhận các tình huống sau: một chương trình Windows98 tạo ra một file với 1 tên dài, máy tính được Reboot để chạy với hệ điều hành DOS hoặc Windows 3, một chương trình cũ xóa tên file DOS từ thư mục nhưng không xóa tên file dài đứng trước nó (vì không hiểu về nó), vv.

Với 1 byte cho checksum Windows 98 có thể ghi nhận 256 tình huống khác nhau, mỗi tình huống tương ứng với một mã nào đó. Hiện nay chúng tôi chưa có tài liệu mô tả về đầy đủ về checksum nên chúng tôi không thể trình bày nhiều hơn nữa về checksum ở đây.

- 10 byte sau trường Sequence, 12 byte sau trường Checksum và 4 byte cuối cùng dùng để lưu trữ 13 kí tự trong phần tên file dài (vì windows98 sử dụng mã Unicode nên mỗi ký tự chiếm 2 byte).

**Chú ý:** Theo hệ điều hành DOS và windows3.1 thì byte đầu tiên của một entry trong bảng thư mục gốc phải là một trong các chữ cái hoặc một số kí tự đặc biệt

nào đó, chứ không thể là các giá trị số. Nên DOS, windows3.1 và các chương trình trên nó không nhận ra các entry của tên file dài theo kiểu windows98 là điều tất nhiên.

Hình vẽ sau đây minh họa cho việc lưu trữ tên file dài trong bảng thư mục của file có tên: **“Nguyen Le Tram Uyen Va Tram Thanh”**.

6 7	m T h a												A	0	C K	n h												0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
2	m U y e												A	0	C K	n V a T												0	r a																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
1	N g u y e												A	0	C K	n L e T												0	r a																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
N	G U Y E N ~ 1												A	N T	S	Creation Time				Last Acc		Upp		Last write				Low		Size																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								

**Hình 4.15:** Các phần tử trong bảng thư mục gốc của windows98 của một file

Theo hình trên thì, windows98 dùng 4 entry liên tiếp nhau, từ thấp đến cao, trong bảng thư mục gốc để lưu trữ tên file và các thông tin liên quan của file **“Nguyen Le Tram Uyen Va Tram Thanh”**: 1 entry theo kiểu DOS để lưu trữ tên file thu gọn NGUYEN~1, và 3 entry đánh số thứ tự 1, 2, và 67 theo kiểu windows98 để lưu toàn bộ phần tên file dài.

**Chú ý:** Vì bảng FAT quá lớn nên windows98 không nạp hết bảng FAT vào bộ nhớ mà chỉ nạp những phần tử cần thiết, các phần tử được sử dụng hiện tại, nhưng các phần tử này luôn thay đổi, do đó Windows 98 sử dụng một “cửa sổ” trên bảng FAT để phục vụ cho mục đích này, theo đó chỉ các phần tử nằm trong “cửa sổ” mới được nạp vào bộ nhớ để sử dụng tại thời điểm hiện tại.

## Tổ chức đĩa của Windows 2000

### Các loại Partition

Windows 2000 giới thiệu hai khái niệm: *Basic Disk* và *Dynamic Disk*. Windows 2000 gọi các đĩa được phân khu dựa vào sơ đồ phân khu của MS\_DOS là các basic disk. Basic là khái niệm mà windows 2000 kế thừa từ MS\_DOS, dynamic disk là một khái niệm mới của nó. Windows 2000 đã cài đặt thành công các dynamic disk và nó có nhiều đặc tính nổi trội hơn so với basic disk. Sự khác nhau cơ bản giữa dynamic disk và basic disk đó là dynamic disk hỗ trợ các multipartition volume.

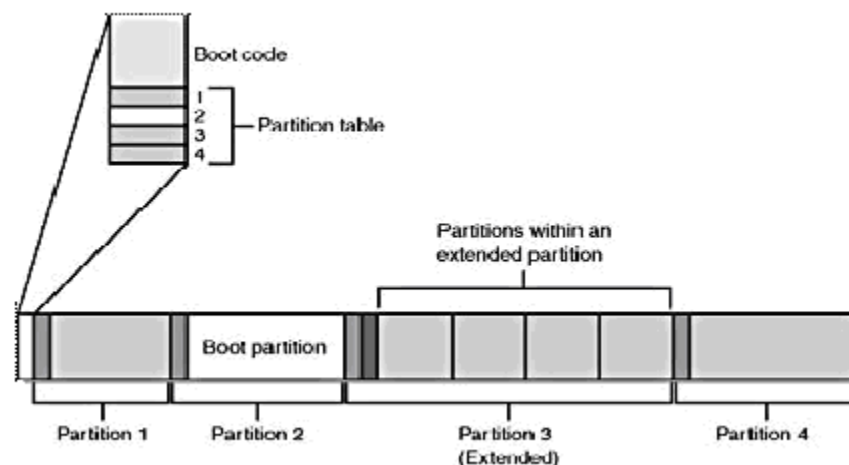
Windows 2000 quản lý tất cả các đĩa như là basic disk trừ khi ta tạo một dynamic disk bằng tay hoặc chuyển một basic disk đang tồn tại thành dynamic disk. Để khuyến khích người quản trị hệ thống sử dụng dynamic disk, Microsoft đưa ra một vài giới hạn sử dụng trên các basic disk, như: chỉ có thể tạo các



multipartition volume mới trên các dynamic disk, chỉ có thể chuyển thành dynamic disk trên các volume NTFS. Một bất lợi của dynamic disk là định dạng mà nó sử dụng là độc quyền và không tương thích với các hệ điều hành khác, bao gồm cả các phiên bản khác nhau của Windows. Vì thế ta không thể truy cập dynamic disk trong môi trường đa boot (dual-boot environment).

➤ **Basic Partitioning:** Khi cài đặt Windows 2000, đầu tiên nó sẽ yêu cầu ta tạo một partition trên ổ đĩa vật lý chính (primary) của hệ thống. Windows 2000 định nghĩa một *system volume* trên partition này để lưu trữ các file mà nó có thể gọi trong quá trình boot máy. Windows 2000 cũng yêu cầu ta tạo một partition mà nó phục vụ như là một home cho *boot volume*, đây là nơi mà chương trình setup sẽ cài đặt các file hệ thống của Windows 2000 và tạo ra một thư mục hệ thống \Winnt. *System volume* và *boot volume* có thể nằm trên cùng một volume. System volume là nơi Windows 2000 lưu trữ các file boot, bao gồm các file Ntldr và Ntdetect, còn boot volume là nơi Windows 2000 lưu trữ các file chính của hệ điều hành như là Ntoskrnl.exe và các file lõi khác.

Theo chuẩn của BIOS trên các hệ thống x86, tất cả các hệ điều hành đều chọn sector đầu tiên của đĩa primary để chứa master boot record (MBR). Chúng ta đã biết khi một processor được boot thì BIOS của máy tính sẽ đọc MBR và xem xét đoạn mã có khả năng thực hiện của MBR. BIOS sẽ gọi đoạn mã của MBR để khởi tạo tiến trình boot hệ điều hành sau khi đã POST máy tính. Cũng như các hệ điều hành của Microsoft khác, MBR của Windows 2000 cũng chứa một bảng *partition*. Bảng này cũng có 4 mục vào, nó dùng để cấp phát cho 4 partition chính trên một đĩa. Bảng partition cũng ghi lại loại của partition là FAT32 hay NTFS.

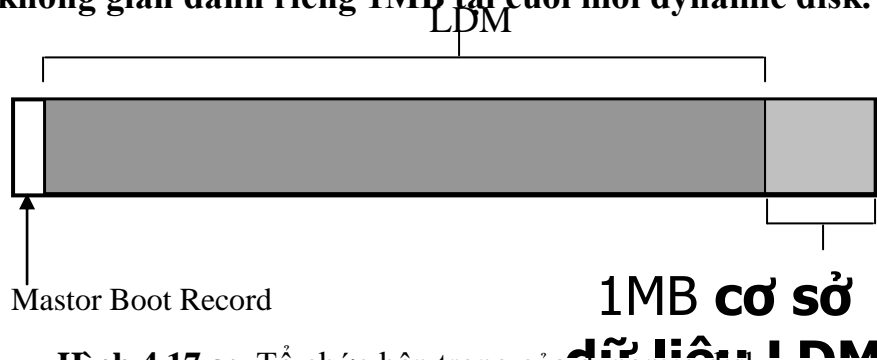


**Hình 4.16:** các partition trên volume NTFS

Có một loại partition đặc biệt đó là *extended partition*, nó chứa một MBR khác với một bảng partition riêng của nó. Bằng cách sử dụng *extended partition* các

hệ điều hành của Microsoft đã phá bỏ giới hạn 4 partition trên một đĩa. Tiến trình boot là một thể hiện đầu tiên trong Windows 2000 tạo nên sự khác biệt rõ rệt nhất giữa các partition *primary* và *extended*. Hệ thống phải tạo một partition primary trên đĩa primary là active. Trong quá trình boot đoạn mã của Windows 2000 trong MBR sẽ nạp đoạn mã được lưu trữ tại sector đầu tiên trong partition active (system volume) vào bộ nhớ và trao quyền điều khiển cho đoạn mã này. Windows 2000 chỉ định sector đầu tiên của bất kỳ partition nào đều là boot sector. Như đã biết ở trước, trong Windows 2000 mọi partition được định dạng với một hệ thống file thì nó có một boot sector, là nơi lưu trữ thông tin về cấu trúc của hệ thống file trên partition đó.

➤ **Dynamic Partitioning:** Các đĩa động được định dạng trong Windows 2000 và là cần thiết để tạo mới các volume multipartition. Hệ thống con Logical Disk Manager (LDM) trong Windows 2000 bao gồm các thành phần trong user mode và device driver giám sát các đĩa động. Cơ sở dữ liệu LDM thường trú trong không gian dành riêng 1MB tại cuối mỗi dynamic disk.



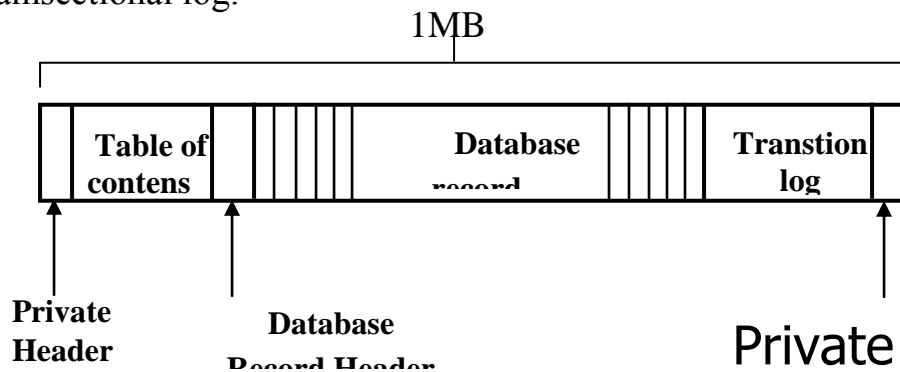
Hình 4.17.a: Tổ chức bên trong của dynamic disk

Không gian đĩa 1MB này là cần thiết trên các basic disk vì Windows 2000 cần có không gian trống này khi người sử dụng cần chuyển một basic disk thành dynamic disk. Mặc dù dữ liệu partition của dynamic disk thường trú trong cơ sở dữ liệu LDM, LDM vẫn cài đặt một bảng partition loại MS\_DOS để thừa kế các tiện ích quản lý đĩa, bao gồm cả các tiện ích chạy dưới Windows 2000 và dưới các hệ điều hành khác trong môi trường boot đôi (dual-boot). Vì các partition LDM không được mô tả trong bảng partition loại MS\_DOS nên nó được gọi là các *soft partition*, các partition loại MS\_DOS được gọi là các *hard partition*.

Một nguyên nhân khác mà LDM phải tạo các bảng partition loại MS\_DOS đó là đoạn mã boot của Windows 2000 có thể tìm trong các system volume hoặc boot volume, ngay cả khi các volume nằm trên các dynamic disk. Nếu một đĩa chứa các volume system hoặc boot, hard partition trong bảng partition MS\_DOS mô tả vị trí của các partition này. Mặt khác, một hard partition bắt đầu tại cylinder đầu tiên của đĩa và mở rộng đến nơi bắt đầu của cơ sở dữ liệu LDM. LDM đánh dấu

partition này như là “LDM”, một partition loại MS\_DOS kiểu mới của Windows 2000. Vùng được bao quanh bởi place-holding này của các partition loại MS\_DOS là nơi LDM tạo các soft partition mà ở đó cơ sở dữ liệu LDM được tổ chức.

Hình 4.17.b sau đây cho thấy các vùng của một dynamic disk. Cơ sở dữ liệu LDM gồm có 4 vùng: Một sector header mà LDM gọi là Private Header. Một bảng nội dung của vùng (Table of Contents). Một cơ sở dữ liệu các bản ghi của vùng. Và một vùng Transactional log.



**Hình 4.17.b:** Cơ sở dữ liệu LDM

- *Private Header sector* và *Private Header mirror*: Các thành phần khác nhau trong Windows 2000 sử dụng một giá trị 128 bit được gọi là GUID: globally unique identifier, để định danh duy nhất các đối tượng. LDM gán cho mỗi dynamic disk một GUID, và *Private Header sector* ghi GUID của dynamic disk lên vùng thường trú của nó, do đó Private Header được thiết kế như là thông tin riêng của từng đĩa. Private Header cũng lưu trữ tên của nhóm đĩa, đó là tên của máy tính cộng thêm Dg0, và điểm bắt đầu của bản cơ sở dữ liệu của nội dung. Để an toàn LDM lưu trữ một bản sao của Private Header ở sector cuối cùng của đĩa đó là *Private Header mirror*.
- *Table of contents* chiếm 16 sector và chứa thông tin về layout của cơ sở dữ liệu.
- *Database record header* lưu trữ thông tin về database record area, bao gồm số các record mà nó chứa, tên và GUID của nhóm đĩa và số thứ tự định danh mà LDM sử dụng cho entry kế tiếp mà nó tạo trong cơ sở dữ liệu.
- *Các sector* sau database record header chứa các record có kích thước cố định 128 byte mà nó lưu trữ các mục vào (entry) mô tả các volume và các partition của nhóm đĩa.

Một mục vào cơ sở dữ liệu có thể là một trong 4 loại: *Partition*, *Disk*, *Component* và *Volume*. LDM sử dụng các mục vào cơ sở dữ liệu để định danh 3 cấp mô tả volume. LDM nối các entry với các định danh đối tượng bên trong. Tại cấp thấp nhất, các *entry partition* mô tả các soft partition, các

định danh được lưu trữ trong partition entry liên kết entry đến một mục vào component và disk. Một *entry disk* trình bày một dynamic disk, nó là một phần của nhóm đĩa và bao gồm GUID của đĩa. Một *entry component* như là một kết nối giữa một hoặc nhiều entry partition và entry volume mà mỗi partition kết nối với nó. Một *entry volume* lưu trữ một GUID của volume, tổng kích thước và trạng thái của volume, và các ký tự đĩa gợi ý. **Các entry ...**

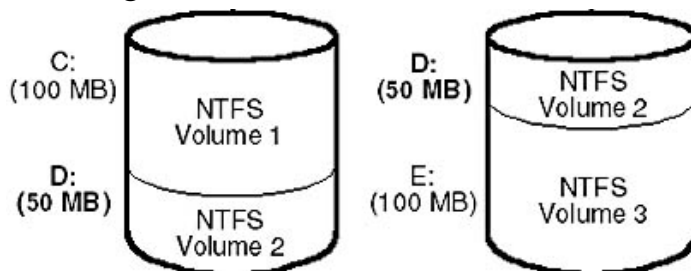
Entry partition mô tả một vùng trên đĩa mà được hệ thống gán thành một volume, entry component nối entry partition với entry volume, và entry volume chứa GUID mà Windows 2000 sử dụng để định danh volume. Các volume multipartition yêu cầu nhiều hơn 3 entry. Ví dụ một striped volume bao gồm ít nhất 2 entry partition: 1 entry component và 1 entry volume. Chỉ có một loại volume có nhiều hơn một entry component đó là Mirror, mirror có 2 entry component, mỗi entry trình bày một nửa (one-half) của mirror. LDM sử dụng 2 entry component cho một mirror vì thế khi bạn break một mirror, LDM có thể split nó tại cấp component, tạo 2 volume mà mỗi volume một entry component. Bởi vì một simple volume yêu cầu 3 entry và 1 MB cơ sở dữ liệu chứa khoảng 8000 entry, do đó số volume mà ta có thể tạo trên hệ điều hành Windows 2000 khoảng 2500.

- Vùng cuối cùng của cơ sở dữ liệu LDM *transactional log area*, gồm một vài sector dùng để lưu trữ thông tin backup cơ sở dữ liệu, với mục đích bảo vệ cơ sở dữ liệu trong trường hợp lỗi crash hoặc power.

### Các loại volume Multipartition

Các volume multipartition phức tạp hơn so với các volume simple, bởi vì nó được tạo ra từ các phân vùng không liên tiếp nhau hoặc trên các đĩa khác nhau. Trong windows 2000 có các loại volume multipartition sau đây:

**Spanned volume:** Một spanned volume là một volume logic đơn bao gồm tối đa 32 partition tự do trên một hoặc nhiều đĩa. Windows 2000 cho phép kết hợp nhiều partition thành một spanned volume, và nó được định dạng theo bất kỳ hệ thống file nào được windows 2000 hỗ trợ. Hình sau đây cho thấy một spanned volume 100Mb được định danh bởi ký tự ổ đĩa D, mà nó được tạo ra từ 1/3 dung lượng của đĩa thứ nhất và 1/3 dung lượng của đĩa thứ 2. Trong hệ điều hành windows NT 4 các spanned volume được gọi là “volume sets”.

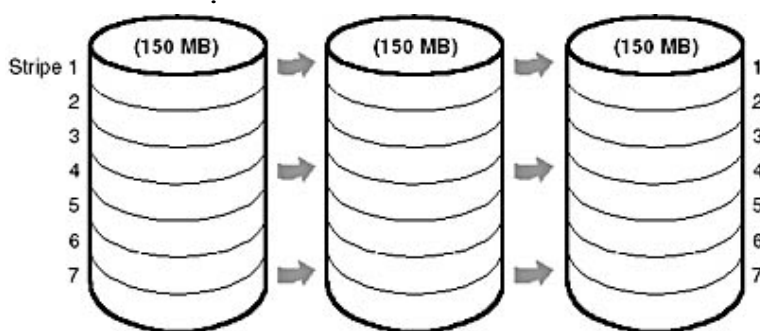


### Hình 4.18.a: Spanned volume

Nếu spanned volume được định dạng theo hệ thống file NTFS, nó có thể được mở rộng bằng cách các thêm vào vùng tự do hoặc các đĩa tự do mà không làm ảnh hưởng đến dữ liệu đang được lưu trữ trên volume. Khả năng mở rộng này là một lợi ích lớn nhất trong việc mô tả tất cả dữ liệu trên một volume NTFS như là một file. NTFS có thể tăng kích thước của một volume logic một cách động, bởi vì trạng thái cấp phát của volume được ghi lại trong một file khác, được gọi là file bitmap. File bitmap có thể được mở rộng để bao gồm bất kỳ một không gian đĩa nào được thêm vào volume.

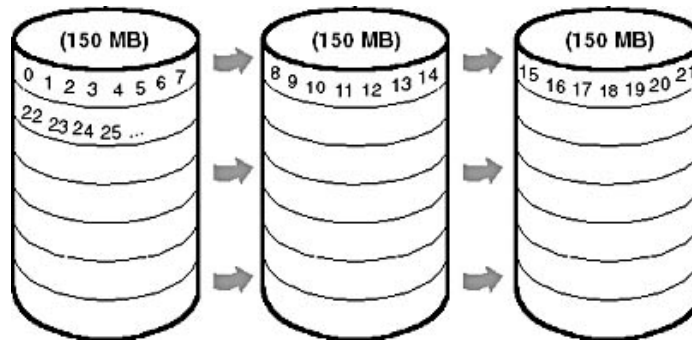
Trình quản lý volume ẩn cấu hình vật lý của các đĩa từ các hệ thống file được cài đặt trên windows 2000. Ví dụ, NTFS xem volume D: trong hình trên như là một volume 100Mb thông thường. NTFS tra cứu bitmap của nó để xác định không gian nào trong volume là còn trống để thực hiện việc cấp phát. Rồi nó sẽ gọi bộ phận quản lý volume để đọc hoặc ghi dữ liệu bắt đầu tại offset byte liên quan trên volume. Bộ phận quản lý volume xem các sector vật lý trong các spanned volume như là được đánh số thứ tự từ vùng trống đầu tiên trên đĩa đầu tiên đến vùng trống cuối cùng trên đĩa cuối cùng. Nó xác định sector vật lý nào trên đĩa nào tương ứng với byte offset được đưa ra.

**Striped volume:** Một striped volume là một dãy có thể lên đến 32 partition, một partition trên một đĩa, các partition không cần phải trải dài trên một đĩa mà chỉ cần nó có cùng kích thước với nhau. Striped volume cũng được hiểu như là các volume RAID cấp 0 (RAID-0). Hình sau đây cho thấy một striped volume bao gồm 3 partition, mỗi partition trên một đĩa.



Hình 4.18.b: Striped volume

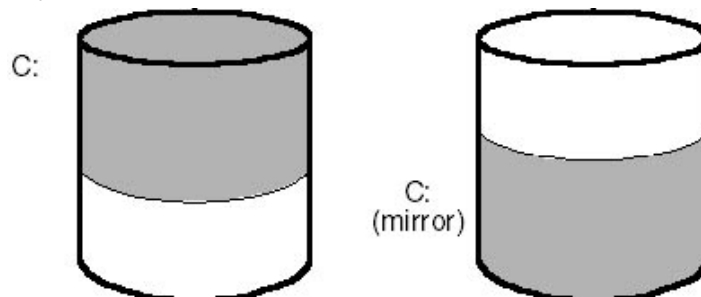
Đối với hệ thống file thì striped volume chỉ là một volume đơn 450Mb thông thường, nhưng đối với bộ phận quản lý volume thì striped volume sẽ giúp tối ưu được thời gian lưu trữ và đọc lại dữ liệu trên nó, bằng cách phân tán dữ liệu của volume trên các đĩa vật lý khác nhau của cùng volume. Bộ phận quản lý volume truy cập sector vật lý trên các đĩa như là nó được đánh số thứ tự trong các stripe xuyên qua các đĩa, như được đưa ra ở hình dưới đây:



**Hình 4.18.c:** Đánh số logic các sector vật lý trong một striped volume

Trong hệ thống striped volume, tất cả dữ liệu trên các đĩa của nó có thể được truy xuất đồng thời, do đó thời gian cho các thao tác vào/ra đĩa có thể được giảm xuống. Điều này đặc biệt có lợi trên các hệ thống cần nhiều thao tác nạp dữ liệu từ đĩa.

**Mirrored volume:** Trong một mirrored volume nội dung của một partition trên một đĩa được nhân đôi (duplicate) trên một partition có kích thước bằng nhau trên một đĩa khác. Mirrored volume đôi khi được tham chiếu như là các volume RAID-1. Hình sau đây cho thấy một mirrored volume.



**Hình 4.18.d:** Mirrored volume

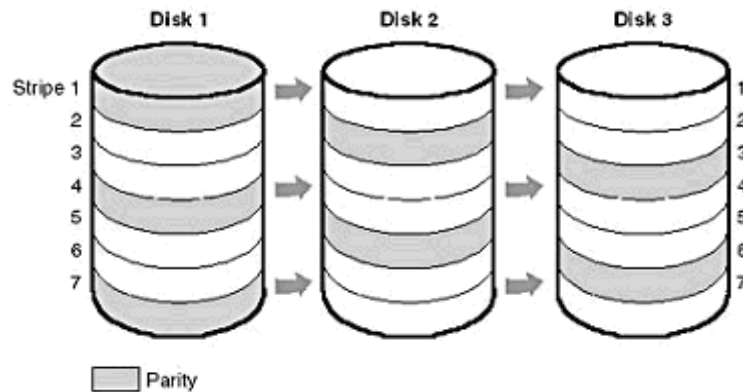
Khi một chương trình ghi đến đĩa C:, bộ phận quản lý volume sẽ ghi cùng dữ liệu đó đến cùng vị trí trên partition mirror. Nếu dữ liệu trên bất kỳ một partition C: nào đó trở thành không thể đọc được vì lý do phần cứng hay phần mềm, thì bộ phận quản lý volume sẽ tự động truy cập đến dữ liệu từ partition mirror.

Một mirrored volume có thể được định dạng bởi bất kỳ hệ thống file nào được hỗ trợ bởi windows 2000.

Mirrored volume có thể giúp cải thiện thông lượng nạp dữ liệu trong các hệ thống yêu cầu nhiều thời gian cho việc nạp dữ liệu, các hệ thống này thường được gọi là các hệ thống tải nặng (heavily load). Khi hoạt động vào/ra tăng cao, bộ phận quản lý volume sẽ cân bằng tải bằng cách cho thực hiện thao tác đọc trên cả partition chính và partition mirror, nhờ vậy mà thời gian đọc có thể giảm được một nửa. Khi một file bị thay đổi thì sự thay đổi này sẽ được cập nhật trên cả partition chính và partition mirror. Mirrored volume là loại multipartition volume chỉ được

hỗ trợ cho các các volume boot và system.

**RAID-5 volume:** RAID-5 volume là một trường hợp chịu lỗi của striped volume. RAID-5 volume cài đặt RAID cấp 5. Nó cũng được hiểu như là striped volume với parity (kiểm tra lỗi chẵn lẻ), bởi vì nó dựa trên cách tiếp cận stripe được tạo nên bởi striped volume. Khả năng chịu lỗi trong trường hợp này có được bằng cách dự trữ một phần tương đương trên đĩa để lưu trữ parity cho mỗi stripe. Hình sau đây trình bày một cách nhìn trực quan về RAID-5 volume.



**Hình 4.19.e:** RAID-5 volume

Trong hình trên, parity cho stripe 1 được lưu trữ trên đĩa 1. Nó chứa tổng của phép logic XOR của các byte của stripe đầu tiên trên đĩa 2 và 3. Parity cho stripe 2 được lưu trữ trên đĩa 2, và parity cho stripe 3 được lưu trữ trên đĩa 3. Sự xoay vòng parity xuyên qua các đĩa theo cách này là một kỹ thuật tối ưu vào ra. Mỗi khi dữ liệu được ghi đến một đĩa, các byte parity tương ứng với các byte bị thay đổi phải được tính toán lại và ghi vào lại. Nếu parity luôn ghi đến cùng một đĩa. Đĩa đó trở thành bận liên tục và có thể trở thành một hiện tượng thắt cổ chai trong I/O.

Khôi phục một lỗi đĩa trong RAID-5 volume dựa vào một nguyên lý cơ bản như sau: trong một biểu thức có  $n$  biến, nếu ta biết được giá trị của  $n-1$  biến, ta có thể xác định giá trị của biến còn lại bằng phép trừ. Ví dụ, trong biểu thức  $x + y = z$ , ở đây là parity stripe, bộ phận quản lý volume tính  $z - y$  để xác định nội dung của  $x$ , và tính  $z - x$  để xác định nội dung của  $y$ . Bộ phận quản lý volume sử dụng sự logic tương tự ở trên để khôi phục dữ liệu bị mất. Nếu một đĩa trong RAID-5 volume lỗi hoặc dữ liệu trên một đĩa trở thành không thể đọc được, thì bộ phận quản lý volume sẽ tìm lại dữ liệu mất bằng cách sử dụng thao tác XOR.

Nếu đĩa 1 trong hình trên lỗi thì nội dung stripe 2 và 5 của nó được tính toán lại bằng cách XOR các stripe tương ứng của đĩa 3 với các stripe parity trên đĩa 2. Nội dung của các stripe 3 và 6 trên đĩa 1 được xác định tương tự bằng cách XOR các stripe tương ứng của đĩa 2 với các stripe parity trên đĩa 3.

## **Quản lý lưu trữ file trên đĩa của WindowsNT/2000**

Một số chức năng được hỗ trợ bởi NTFS của windows 2000

Hệ thống NTFS được thiết kế bao gồm những tính năng được yêu cầu một hệ thống file chuyên nghiệp. Để giảm tối đa việc mất dữ liệu do hệ thống bị ngưng đột ngột hoặc bị phá hỏng, hệ thống file phải đảm bảo metadata của hệ thống phải luôn ở trạng thái nguyên vẹn. Để bảo vệ dữ liệu tránh những truy xuất bất hợp lệ, hệ thống file phải có các mô hình bảo mật thích hợp. Cuối cùng, một hệ thống file phải tính đến việc dư thừa dữ liệu, và cho phép ngăn chặn dư thừa dữ liệu dựa trên phần mềm, đây là một giải pháp mang lại hiệu quả về mặt kinh tế hơn so với các giải pháp phần cứng.

Với những mục tiêu trên, NTFS đã cung cấp cho người sử dụng và chương trình của người sử dụng những tính năng cao cấp: Tên dựa trên mã Unicode; Chỉ mục chung; Ánh xạ động các Bad cluster; Nén và giải nén File; Cấp hạn ngạch đĩa; Chống phân mảnh; Mã hoá các File; vv. Sau đây chúng ta sẽ xem xét một vài tính năng trong số đó:

**Chỉ mục chung:** Kiến trúc NTFS được cấu trúc để cho phép chỉ mục các thuộc tính của file trên volume đĩa. Cấu trúc này cho phép hệ thống file tìm kiếm các file trên đĩa một cách hiệu quả dựa trên các điều kiện kết hợp. Hệ thống file FAT chỉ chỉ mục theo tên file và không có sự sắp xếp trong đó nên hiệu quả của việc tìm kiếm file trong các thư mục lớn sẽ rất thấp.

**Ánh xạ tự động các Bad Cluster:** Thông thường nếu một chương trình cố gắng đọc dữ liệu từ một sector hỏng trên đĩa, thì việc đọc đó sẽ thất bại và dữ liệu trên cluster được định vị sẽ trở thành không thể truy cập được. Nhưng nếu volume được cài đặt hệ thống NTFS có khả năng sửa lỗi thì trình điều khiển lỗi của windows 2000 sẽ tự động tìm một bản sao dữ liệu tốt của dữ liệu tại sector hỏng đó và báo cho NTFS biết là sector đó đã bị hỏng. NTFS sẽ cấp một cluster mới thay cho cluster chứa sector hỏng và sao chép dữ liệu vào sector mới đó. Và sector này sẽ được chỉ báo để không được sử dụng nữa.

**Nén file:** NTFS hỗ trợ việc nén dữ liệu các tập tin. Bởi vì NTFS thực hiện việc nén và giải nén một cách trong suốt, nên các ứng dụng không cần thay đổi để có được những thuận lợi từ tính năng này. Các thư mục cũng có thể được nén, tức là tất cả các file được tạo ra trong thư mục cũng đều được nén.

**Cấp hạn ngạch (Quotas) đĩa cho người sử dụng:** Người quản trị hệ thống thường muốn theo dõi hoặc giới hạn không gian lưu trữ trên volume mà họ đã chia sẻ cho người sử dụng, vì thế NTFS đã cung cấp các công cụ quản lý hạn ngạch để họ thực hiện điều này. Với công cụ quản lý hạn ngạch, NTFS cho phép chỉ định không gian đĩa mà mỗi người sử dụng được phép sử dụng và thông báo cho người sử dụng biết không gian đĩa còn lại mà họ được phép sử dụng để lưu trữ, một cách tức thời.

#### IV.9.2. Cấu trúc của MFT

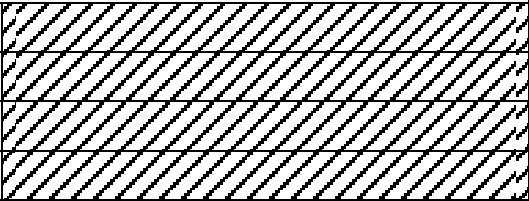


Trong hệ thống file NTFS, tất cả dữ liệu lưu trữ trên volume đều được chứa trong các tập tin, bao gồm cấu trúc dữ liệu được dùng để định vị và đọc lại các file, dữ liệu của bootstrap, và bitmap mà nó dùng để ghi lại trạng thái cấp phát của tất cả các block (cluster) trên volume (gọi là metadata NTFS). Việc lưu trữ mọi dữ liệu trong các tập tin cho phép hệ thống file dễ dàng tìm kiếm và bảo trì dữ liệu, và mỗi tập tin riêng có thể được bảo vệ bởi một mô tả an toàn dữ liệu riêng.

**MFT (Master File Table)** là trung tâm trong cấu trúc của các volume NTFS. Windows 2000 sử dụng MFT để quản lý việc lưu trữ các file và thư mục trên một volume. MFT bao gồm một dãy các record (còn gọi là record file hay entry), có kích thước cố định là 1Kb. Mỗi record trong MFT dùng để mô tả về một file hoặc thư mục trên volume, kể cả record của chính nó. Nó chứa những thuộc tính của file, như tên file, timestamps (các đặc trưng về thời gian tạo và thay đổi tập tin), và danh sách địa chỉ đĩa (cluster) nơi lưu trữ các block của file. Nếu kích thước file quá lớn hệ điều hành cần phải sử dụng 2, hoặc nhiều hơn một record MFT để lưu danh sách địa chỉ tất cả các block đĩa chứa file trên volume. Trong trường hợp này, record MFT đầu tiên của file được gọi là record cơ sở, nó trỏ đến các record MFT mở rộng khác.

Hệ thống file của Windows 2000 sử dụng một bitmap để theo dõi các toàn bộ MFT còn trống. Số lượng record của MFT có thể tăng lên khi cần và có thể tăng đến  $2^{48}$  record vì bản thân MFT là một file, nó như có thể được xuất hiện ở mọi nơi trong volume.

Hình sau cho thấy một đoạn đầu tiên của MFT của một volume.

	.....		
16	Các file và thư mục đầu tiên của người sử dụng		
15	Unused		
14	Unused		
13	Unused		
12	Unused		
11	\$Extend:	Extentions: quotas, ...	
10	\$Upcase:	Case conversion table	
9	\$Secure:	Security descriptor for all files	
8	\$BadClus:	List of bad of block	
7	\$Boot:	Bootstrap Loader	
6	\$Bitmap:	Bitmap of blocks used	
5	\	Root directory	
4	\$AttrDef	Attribute definitions	

}

Metadata File

3	\$Volume	Volume file
2	\$LogFile	Log file to recovery
1	\$MftMirr	Mirror copy of MFT
0	\$MFT	Master File Table

Hình 4.6: Các record đầu tiên trong MFT

Mỗi record MFT bao gồm một tập các cặp (attribute header, value). Mỗi thuộc tính thường biểu diễn bởi phần bắt đầu của giá trị thuộc tính, hoặc là tất cả, vì giá trị này có chiều dài thay đổi. Nếu giá trị thuộc tính ngắn thì nó sẽ được lưu ở record MFT, ngược lại nếu nó quá dài thì sẽ được lưu ở một nơi khác trên đĩa và tại record có một con trỏ trỏ đến địa chỉ này.

Mười sáu record đầu tiên của MFT được dành riêng cho các file metadata NTFS. Mỗi một record mô tả 1 file có thuộc tính và các block dữ liệu, giống như các file khác. Tên các file metadata này được bắt đầu bởi dấu '\$':

- Record đầu tiên (tên là *\$MFT*) mô tả chính bản thân MFT. Đặc biệt, nó chỉ ra địa chỉ block, nơi lưu trữ bảng file (MFT), vì thế hệ thống có thể dễ dàng tìm thấy file MFT. Rõ ràng, Window 2000 cần phải có cơ chế để tìm thấy block đầu tiên của file MFT để tìm đến thông tin còn lại của hệ thống file. Cơ chế trên sẽ tìm đến block khởi động, nơi mà địa chỉ của nó được cài đặt lúc cài đặt hệ thống.
- Record 1 (tên là *\$MftMirr*) chỉ đến một bản sao của phần đầu tiên của file MFT. Thông tin này là rất quan trọng vì vậy phải có 1 bản sao khác để khắc phục việc những block đầu tiên của file MFT thường bị bad.
- Record 2 (tên là *\$LogFile*) chỉ đến một log-file. Khi thêm vào volume 1 thư mục mới hay khi xóa khỏi volume 1 thư mục thì hệ thống sẽ ghi lại sự thay đổi này vào log file trước khi thực hiện. Sự thay đổi về thuộc tính của các file cũng được ghi lại ở log file. Sự thay đổi về dữ liệu của người sử dụng không được ghi ở đây.
- Record 3 (tên là *\$volume*) chỉ đến file chứa những thông tin về volume như kích thước volume, nhãn của volume và version của nó.
- Như đã đề cập ở trên, mỗi record MFT chứa dãy các cặp: thuộc tính header, giá trị. File Attribute chứa một bảng định nghĩa thuộc tính, nó định nghĩa các loại thuộc tính được hỗ trợ bởi windows 2000 trên volume. Thông tin của file này được chỉ bởi record 4 (tên là *\$Attr Def*) của MFT.
- Kế tiếp là thư mục gốc, nó là 1 file và có kích thước tăng tùy ý. Nó được mô tả trong record 5 (tên là *\*) của MFT. Entry này được dự trữ cho thư mục gốc. Nó là một file chứa một chỉ mục của các file và thư mục được lưu trữ trong phần gốc của cấu trúc thư mục NTFS. Khi NTFS được yêu cầu để mở

một file, thì nó bắt đầu tìm kiếm trong file của thư mục gốc. Sau quá trình mở file, NTFS lưu trữ sự tham chiếu MFT của file đó để nó có thể truy cập trực tiếp record MFT của file khi nó đọc hoặc ghi file sau đó.

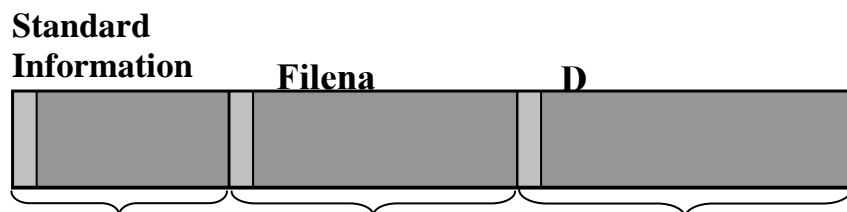
- Không gian trống trên volume được theo dõi bởi 1 bitmap. Bitmap này là một file. Địa chỉ đĩa của file bitmap này được ghi ở record 6 (tên là *\$bitmap*) của MFT.
- Record 7 chỉ đến file Bootstrap (có tên là *\$Boot*): File này chứa mã bootstrap của windows 2000. Để hệ thống boot được, mã bootstrap phải được nạp vào một vị trí đặc biệt trên đĩa. Trong quá trình format đĩa hệ điều hành định nghĩa vùng này như là một byte bằng cách tạo ra một file record cho nó. File boot cũng như các file metadata NTFS, cũng có thể được bảo vệ riêng bằng các công cụ của mô tả bảo mật mà nó được áp dụng cho tất cả các đối tượng windows 2000.
- Record 8 (có tên là *\$BadClus*) chỉ đến một file, file này được dùng để lưu trữ danh sách liên kết các block (cluster) bị bad trên volume.
- Record 9 (tên file là *\$Secure*): chỉ đến file chứa thông tin bảo mật, file này lưu trữ cơ sở dữ liệu về các mô tả bảo mật trên toàn bộ volume. Các file và thư mục NTFS có thể có một thiết lập mô tả bảo mật riêng, nhưng NTFS lưu trữ các thiết lập này trong một file chung, nó cho phép các file và các thư mục có cùng thiết lập bảo mật tham chiếu đến cùng một mô tả bảo mật. Trong nhiều môi trường, toàn bộ cây thư mục có cùng một thiết lập bảo mật, nên cơ chế này mang lại nhiều thuận lợi trong việc tiết kiệm không gian lưu trữ các mô tả bảo mật.
- Record 10 (tên là *\$Upcase*): chỉ đến file case mapping, file này chứa bảng chuyển đổi giữa kí tự thường và kí tự hoa.
- Cuối cùng, record 11 (tên là *\$Extend*) chỉ đến thư mục, thư mục này chứa các file hỗn hợp như: hạn ngạch đĩa (Quotas), các định danh đối tượng (Object identifier), ...
- 4 record MFT cuối cùng dự trữ cho việc sử dụng sau này.

Khi lần đầu tiên truy cập volume, NTFS phải Mount nó, tức là đọc metadata từ đĩa và xây dựng cấu trúc dữ liệu bên trong để nó có thể xử lý các truy cập từ các ứng dụng. Để mount volume, NTFS tìm trong boot sector địa chỉ vật lý của MFT. Theo trên, record file riêng của MFT là entry đầu tiên trong MFT, record file thứ hai trỏ đến một file được định vị ở vùng giữa đĩa, đó là MFT mirror (tên là *\$MftMir*). Khi NTFS tìm thấy record file của MFT, nó lấy được những thông tin ánh xạ từ VNC-to-LCN trong thuộc tính data và lưu vào bộ nhớ để chuẩn bị cho các quá trình ghi/đọc file sau này.

## **Các Record File**

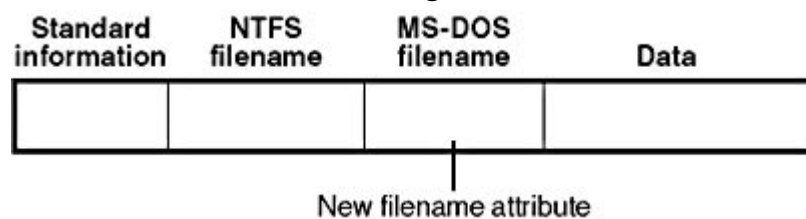
Hệ thống file NTFS lưu trữ các file như là một tập các cặp thuộc tính/giá trị (attribute/value), một trong số đó là dữ liệu nó chứa (được gọi là thuộc tính dữ liệu không tên). Các thuộc tính của một file bao gồm tên file, thông tin time stamp và có thể là các thuộc tính có tên được thêm vào. Hình 4.19.a sau đây cho thấy một record MFT cho một file nhỏ: có 3 thuộc tính: Standard Information (thông tin chuẩn), Filename (tên file) và Data (dữ liệu).

Mỗi thuộc tính file được lưu trữ như là một dòng tách biệt của các byte trong phạm vi file. Nói cách khác NTFS không đọc ghi các file mà nó chỉ đọc ghi các dòng thuộc tính. NTFS cung cấp các thao tác thuộc tính như: create, delete, read (byte range), write (byte range). Các dịch vụ ghi đọc thường thao tác trên các thuộc tính dữ liệu không đặt tên. Tuy nhiên, một lời gọi có thể chỉ ra một thuộc tính dữ liệu khác bằng cách sử dụng cú pháp dòng dữ liệu được đặt tên.



**Hình 4.19.a:** Một record MFT cho một file nhỏ, có 3 thuộc tính

Cả NTFS và FAT đều cho phép mỗi tên file trong đường dẫn có thể dài đến 255 ký tự, trong đó có cả các ký tự unicode và các dấu phân cách khác. Để tương thích với các ứng dụng 16 bit của DOS, khi có file với tên file dài được tạo ra thì windows 2000 sẽ tự động sinh ra một tên file theo kiểu của DOS (tên file 8.3). Tên theo kiểu DOS được lưu trữ trong cùng một record trong MFT với tên file NTFS (tên file dài), vì thế nó tham chiếu đến cùng một file.

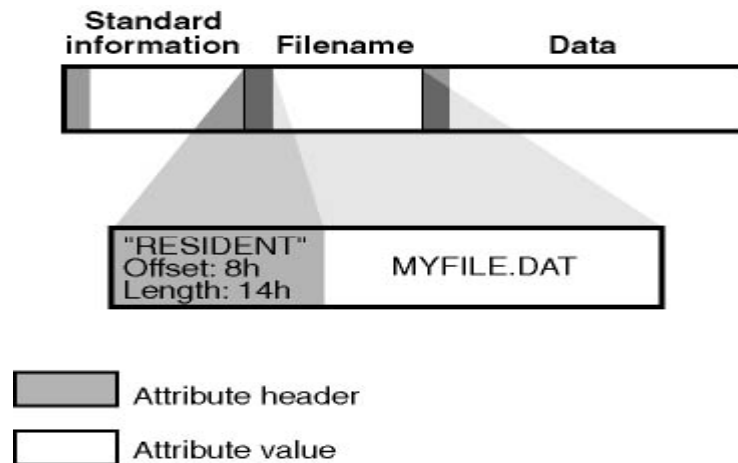


**Hình 4.19.b:** Một record MFT có chứa tên MS\_DOS

Windows 2000 tạo ra tên file theo kiểu DOS từ tên file dài tương tự như cách mà hệ điều hành windows98 đã làm. Tên MS\_DOS được dùng để ghi, đọc, copy một file.

### Thuộc tính thường trú và thuộc tính không thường trú

Với các file có kích thước nhỏ thì tất cả thuộc tính và giá trị của nó được chứa trong một record trong MFT. Khi giá trị của thuộc tính được lưu trực tiếp trong MFT thì thuộc tính đó được gọi là thuộc tính thường trú. Thuộc tính thông tin chuẩn và thuộc tính chỉ mục gốc thường được định nghĩa là thuộc tính thường trú.

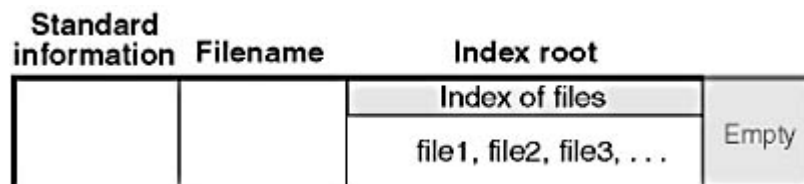


**Hình 4.20.a:** Header và value của thuộc tính thường trú

Mỗi thuộc tính đều bắt đầu với một header, header này chứa thông tin về thuộc tính, đó là thông tin mà NTFS dùng để quản lý thuộc tính. Header cho biết các thông tin liên quan đến giá trị của nó như là có phải là thường trú (RESIDENT) hay không, offset từ header đến giá trị của thuộc tính, độ dài (length) giá trị của thuộc tính, vv. Hình 4.20.b sau đây cho thấy thuộc tính filename gồm có header là “RESIDENT” + Offset:8h + Length:14h và value là MYFILE.DAT.

Khi giá trị của thuộc tính được lưu trữ trong record MFT thì thời gian đọc nội dung của một file của NTFS sẽ được giảm xuống, vì nó không phải tìm danh sách các cluster chứa nội dung của file dựa vào việc phân tích bảng FAT như trong các hệ thống file FAT, mà chỉ cần đọc ngay giá trị tại các cluster trên đĩa chứa nội dung của file, danh sách các cluster này được ghi ở phần giá trị của thuộc tính.

Thuộc tính cho các thư mục nhỏ cũng giống như thuộc tính của các file nhỏ, nó có thể thường trú trong MFT. Hình sau đây là một record MFT cho thư mục nhỏ:



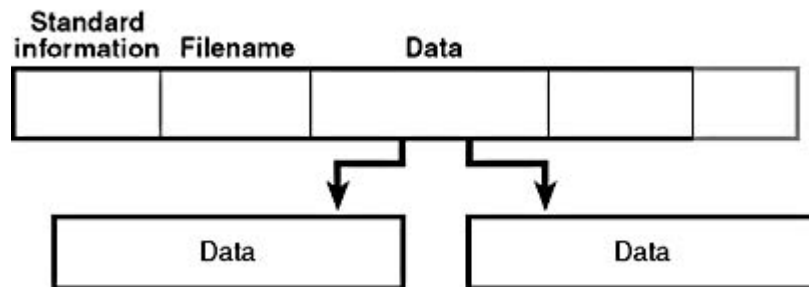
**Hình 4.20.b:** Một record MFT cho thư mục nhỏ

Trong đó thuộc tính Index root chứa một chỉ mục của các tham chiếu đến các file và các thư mục con trong thư mục.

Trong thực tế nội dung của một thư mục, gồm các file và các thư mục con trong nó, không thể nén thành một record MFT có kích thước cố định 1MB. Nếu một thuộc tính đặc biệt, thuộc tính dữ liệu của file chẳng hạn, là quá lớn để chứa hết trong một record MFT thì NTFS sẽ cấp phát các cluster riêng cho dữ liệu của thuộc tính từ MFT. Vùng này được gọi là Run (hay là phạm vi). Nếu giá trị của

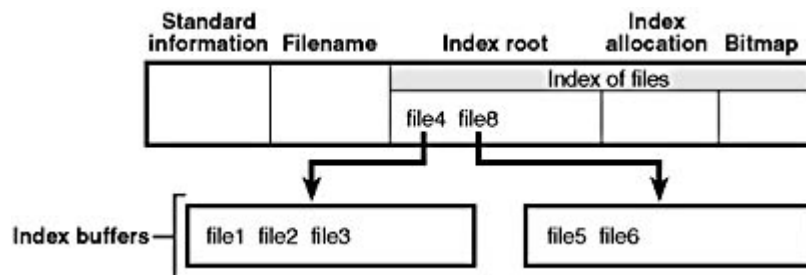
thuộc tính sau đó phát triển, nội dung của file tăng lên chẳng hạn, thì NTFS sẽ định vị một Run khác cho dữ liệu được thêm vào. Các thuộc tính mà giá trị của nó được lưu trữ trong các Run chứ không phải trong các record MFT được gọi là các thuộc tính không thường trú. Hệ thống file sẽ quyết định có hay không một thuộc tính cụ thể là thường trú hay không thường trú.

Khi một thuộc tính là không thường trú thì header của nó sẽ chứa các thông tin mà NTFS cần để tìm đến giá trị của thuộc tính trên đĩa. Hình sau đây cho thấy thuộc tính dữ liệu không thường trú được lưu trữ trong 2 Run.



**Hình 4.20.c:** Record cho một file lớn với 2 Run

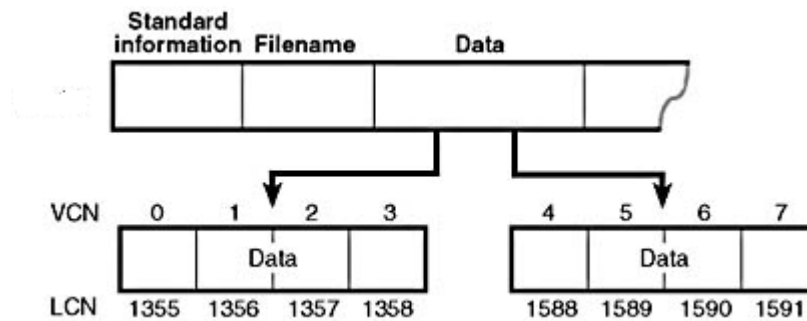
Trong số các thuộc tính chuẩn (standard attribute), chỉ có các thuộc tính có thể tăng lên là có thể trở thành không thường trú. Các thuộc tính thông tin chuẩn và tên file luôn luôn là thường trú.



**Hình 4.20.d:** Record MFT cho thư mục lớn với chỉ mục filename không thường trú

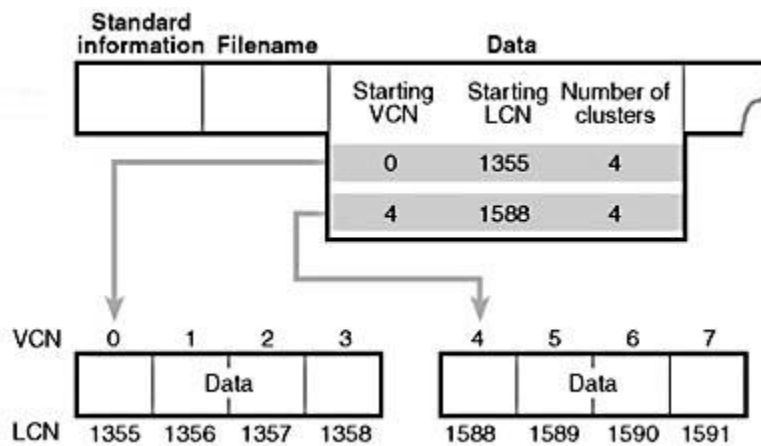
Một thư mục lớn có thể có các thuộc tính không thường trú. Chúng ta hãy xem hình say đây, ta thấy record MFT không đủ chỗ để lưu trữ hết Index của các file chứa trong một thư mục lớn. Một phần của Index được lưu trữ trong thuộc tính Index Root, phần còn lại được lưu trong các Run không thường trú được gọi là vùng đệm chỉ mục (Index Buffer).

NTFS theo dõi các Run bằng các cặp ánh xạ VCN-to-LCN. Các LCN đánh số thứ tự của các cluster trên toàn volume từ 0 đến n. Các VCN đánh số các cluster riêng cho từng file từ 0 đến m. Xem hình 4.20.e sau đây:



**Hình 4.20.e:** Các VCN cho thuộc tính dữ liệu không thường trú

Nếu file này có nhiều hơn 2 Run thì Run thứ 3 sẽ bắt đầu với VCN 8. Xem hình sau đây, header của thuộc tính dữ liệu chứa các ánh xạ VCN-to-LCN cho 2 Run ở đây, nó cho phép NTFS dễ dàng tìm đến các định vị cấp phát trên đĩa.



**Hình 4.20.f:** Các ánh xạ VCN-to-LCN cho thuộc tính dữ liệu không thường trú.

Hình 4.20.f chỉ ra các Run dữ liệu, các thuộc tính khác có thể lưu trữ trong các Run nếu record trong MFT không đủ chỗ để chứa chúng và nếu một file nào đó có quá nhiều thuộc tính dẫn đến không chứa đủ trong một record thì một record thứ 2 trong MFT được sử dụng để chứa các thuộc tính thêm vào. Trong trường hợp này một thuộc tính được gọi là danh sách thuộc tính (attribute list) sẽ được thêm vào. Thuộc tính attribute list chứa tên và mã loại cho mỗi thuộc tính của file và file tham chiếu của record MFT, là nơi thuộc tính được nạp. Thuộc tính attribute list được cung cấp cho các trường hợp: kích thước file tăng lên hoặc file bị phân mảnh, dẫn đến một record MFT đơn không thể chứa hết các ánh xạ VCN-to-LCN cần thiết để tìm tất cả các Run của nó.

**Tóm lại trong một record MFT có thể có các trường sau:**

- Trường đầu tiên của một record MFT là *record header*, theo sau đó là các cặp header và value của các thuộc tính. Record header chứa 1 mã số sử dụng để kiểm tra tính hợp lệ, số kế tiếp được cập nhật mỗi khi record được sử dụng lại cho

file mới, các tham chiếu đến file, số byte hiện tại trong record được sử dụng, nhận dạng (chỉ số, dãy số liên tiếp) của record cơ sở (chỉ sử dụng cho record mở rộng), và 1 số trường khác. Sau record header là header của thuộc tính thứ nhất và giá trị của thuộc tính thứ nhất, header của thuộc tính thứ hai và giá trị của thuộc tính thứ hai,...

- NTFS định nghĩa 13 thuộc tính có thể xuất hiện ở các record MFT. Chúng được liệt kê ở bảng sau đây. Mỗi record MFT bao gồm 1 dãy các header thuộc tính, chúng được đồng nhất với phần đầu của thuộc tính và cho biết độ dài và vị trí của trường giá trị cùng với trạng thái cờ và một số thông tin khác. Thông thường, giá trị thuộc tính nằm ngay sau các header của chúng, nhưng nếu giá trị này quá dài để đặt trong 1 record MFT, thì chúng được đặt vào 1 block đĩa tách rời. Thuộc tính như vậy được gọi là thuộc tính không thường trú. Một vài thuộc tính như là tên, có thể được lặp lại, nhưng tất cả các thuộc tính phải được đặt trong 1 hàng cố định trong record MFT. Các header cho thuộc tính thường trú có độ dài 24 byte; đối với những thuộc tính không lưu trú là dài hơn vì chúng lưu thêm những thông tin để tìm thuộc tính trên đĩa.

Thuộc tính	Mô tả
Standard information	Các bítcờ, timestamp,...
File name	Tên file trong Unicode: có thể lặp lại đối với tên DOS
Security descriptor	Đã lỗi thời. Thông tin bảo mật trong \$extend\$Secure
Attribute list	Vị trí của các MFT record thêm vào nếu cần
Object ID	64-bit, file được nhận biết là duy nhất trên volume
Repase point	Dùng cho các liên kết tăng dần và tượng trưng
Volume name	Tên của volume này (chỉ sử dụng trong \$Volume)
Volume information	Phiên bản của Volume (chỉ sử dụng trong \$Volume)
Index root	Được sử dụng cho các thư mục
Index allocation	Được sử dụng cho các thư mục rất lớn
Bitmap	Được sử dụng cho các thư mục rất lớn
Logged utility stream	Điều khiển kết nối đến \$LogFile
Data	Dữ liệu luồng; có thể lặp lại nhiều lần

**Bảng 4.7:** Các thuộc tính sử dụng trong record MFT

- Trường thông tin chuẩn (*Standard Information*) của file bao gồm: thông tin bảo mật, timestamp, các liên kết cố định, bít chỉ đọc và bít lưu trữ, vv. Nó



là trường có kích thước cố định và luôn hiện hữu.

- Trường tên file (File Name) là một chuỗi mã Unicode có độ dài thay đổi được. Để tạo các file với các tên không phải là tên MS-DOS gần với các ứng dụng 16-bit cũ, các file cũng có thể có tên 8+3 của MS-DOS. Nếu tên file thực sự tuân theo quy tắc đặt tên 8+3 của MS-DOS, thì tên file MS\_DOS phụ sẽ không được sử dụng.

- Trong NT 4.0, thông tin bảo mật (*Security*) có thể đưa vào một thuộc tính nhưng trong Windows 2000 tất cả được đưa vào một file riêng, vì thế nhiều file có thể chia sẻ một phần bảo mật.

- Danh sách thuộc tính (*Attribute List*) là cần thiết nếu thuộc tính không đặt trong record MFT. Thuộc tính này là để tìm ra các record mở rộng. Mỗi mục vào của dãy thuộc tính chứa một chỉ số 48-bit trong MFT gọi đó là record mở rộng và một dãy số 16-bit cho phép kiểm tra sự phù hợp của record mở rộng và record cơ sở.

- Thuộc tính định danh đối tượng ID (*Object Identifier*) của đối tượng làm cho tên file là duy nhất.

- Trường parse point gọi là các thủ tục phân tách tên file để thực hiện một thao tác đặc biệt nào đó. Kỹ thuật này được sử dụng trong cài đặt và liên kết biểu tượng.

- Hai thuộc tính volume (*Volume Name* và *Volume Information*) chỉ sử dụng để xác định volume.

- Ba thuộc tính tiếp theo (*Index Root*, *Index Allocation* và *Bitmap*) được sử dụng cho việc cài đặt các thư mục.

- Thuộc tính Logged *utility stream* được sử dụng bởi hệ thống file mã hoá.

- Cuối cùng, là thuộc tính dữ liệu. Một dãy tên file nằm trong thuộc tính heard. Tiếp theo header là một danh sách các địa chỉ đĩa mà chúng được gọi là block chứa file, hay chỉ cho các file vài trăm byte của riêng nó. Trong thực tế, đặt dữ liệu file vào record MFT được gọi là file trực tiếp. Phần lớn thì dữ liệu không đặt ở MFT record, vì vậy thuộc tính này thường không thường trú.

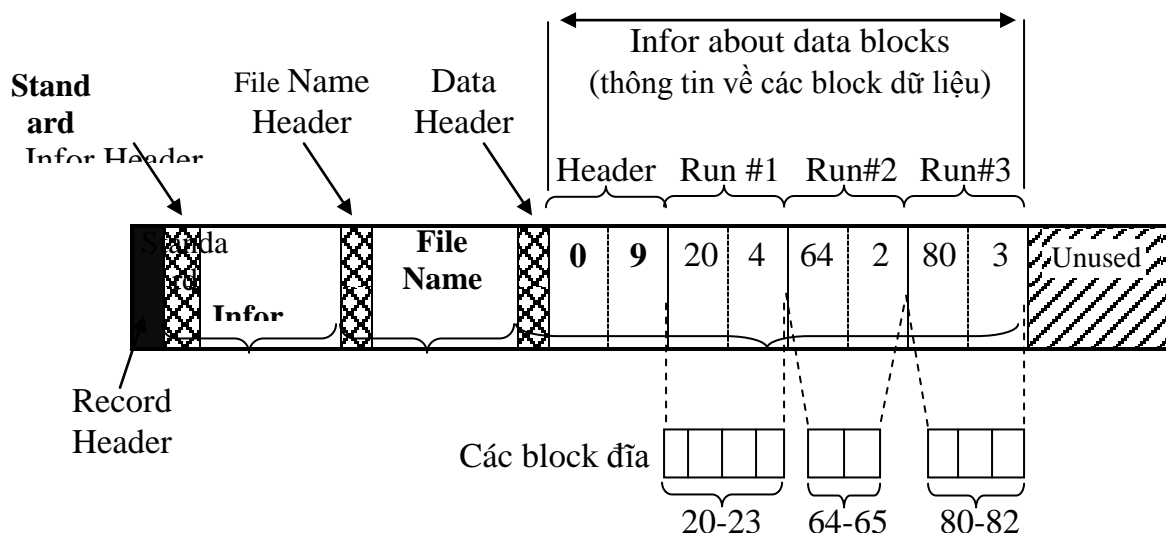
#### IV.9.3. Quản lý danh sách các block chứa File trên đĩa

Trong Windows 2000 với NTFS thì nội dung của một File cần lưu trữ trên volume cũng được chia thành các block (tương ứng với một block hay cluster trên đĩa), các block file còn được gọi là các block logic. Các block file có thể được lưu tại một hoặc nhiều đoạn block không liên tiếp nhau trên đĩa, một đoạn block bao gồm  $n$  block liên tiếp nhau ( $n = 2, 3, 4, \dots$ ), một đoạn block trong trường hợp này được gọi là một Run. Ví dụ file A được lưu trữ trên 7 block liên tiếp: từ block 10 đến block

16 và file B được lưu trữ trên 35 block: từ block 30 đến block 44 và từ block 41 đến block 60. Như vậy file A được lưu trữ trên một đoạn block (Run #1) còn file B được lưu trữ trên hai đoạn block (Run #1 và Run #2). Thông thường, nếu block logic đầu tiên của một file ghi ở block 20 trên đĩa thì block logic 2 được ghi vào block 21 trên đĩa và block logic thứ 3 được ghi vào block 22 trên đĩa, vv. Với cách lưu trữ này hệ điều hành có thể ghi nhiều block logic vào các block đĩa cùng một lúc nếu có thể. Đây là một ưu điểm của Windows 2000.

Danh sách các block đĩa chứa nội dung của một file được mô tả bởi một record trong MFT (đối với file nhỏ) hoặc bởi một dãy tuần tự các record trong MFT, có thể không liên tiếp nhau (đối với file lớn). Mỗi record MFT dùng để mô tả một dãy tuần tự các block logic kề nhau. Mỗi record MFT trong trường hợp này bắt đầu với một header lưu địa chỉ offset của block đầu tiên trong file. Tiếp đến là địa chỉ offset của block đầu tiên mà không nằm trong record. Ví dụ nếu có một file được lưu tại 2 đoạn block là: 0 – 49 và 60 – 79 thì record đầu tiên có một header của (0, 50) và sẽ cung cấp địa chỉ đĩa cho 50 block này và record 2 sẽ có một header của (60, 80) và sẽ cung cấp địa chỉ đĩa cho 20 block đó. Tiếp sau mỗi record header là một hay nhiều cặp, mỗi cặp lưu một địa chỉ của block đĩa bắt đầu đoạn block và số block có trong đoạn (độ dài của một Run).

Hình sau đây là một record MFT chứa thông tin của một file.



**Hình 4.21:** Một record MFT cho 1 file có 3 run, gồm 9 block

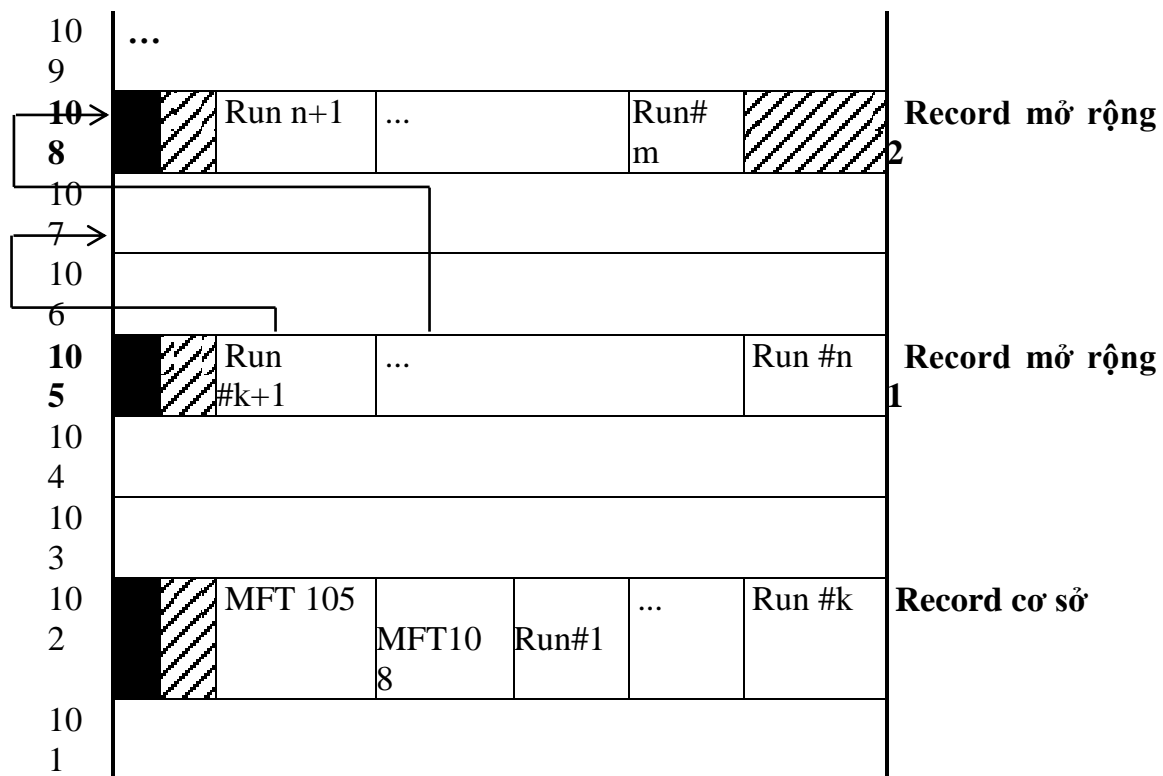
Hình 4.21 là một record MFT cho một file ngắn (ngắn ở đây có nghĩa là tất cả các thông tin về các block của file chứa vừa trong một record MFT). File này gồm 9 block, nó chứa 3 Run: Run #1 gồm 4 block từ 20 đến 23, Run #2 gồm 2 block từ 64 đến 65 và Run #3 gồm 3 block từ 80 đến 82. Mỗi Run được ghi vào record MFT theo từng cặp (địa chỉ block đĩa, tổng số block).

Cần chú ý rằng không có giới hạn trên cho kích thước của các file được trình bày theo cách này. Trong trường hợp thiếu địa chỉ nén, mỗi cặp cần có 2 số 64-bit trong tổng số 16 byte. Tuy nhiên một cặp có thể tượng trưng cho một triệu hoặc hơn nữa các block đĩa liên tiếp. Trên thực tế, một file 20 Mb bao gồm 20 Run của 1 triệu block 1 Kb, mỗi Run được đặt trong một record MFT.

Khi lưu trữ một file có kích thước lớn (số lượng block lớn) hoặc thông tin của file có sự phân mảnh lớn thì các lưu trữ các block của nó trong MFT được trình bày như trong hình sau:

Trong hình sau ta thấy record cơ sở của file nằm ở record MFT 102. Nó có quá nhiều Run trên một record MFT, vì thế hệ thống cần phải có nhiều record mở rộng, ở đây là thêm 2 record mở rộng, và đặt chúng vào record cơ sở. Phần còn lại của record cơ sở được sử dụng cho k Run đầu tiên.

Khi tất cả các byte trong record 102 đã được dùng hết, thì các run được tích lũy sẽ tiếp tục với record MFT 105. Khi nhiều run được sắp xếp vào hết trong record này hay khi record này đã đầy thì phần run còn lại sẽ vào record MFT 108. Theo cách này, nhiều record MFT có thể được dùng để quản lý các file lớn khác.



**Hình 4.22:** Các record MFT của một file lớn

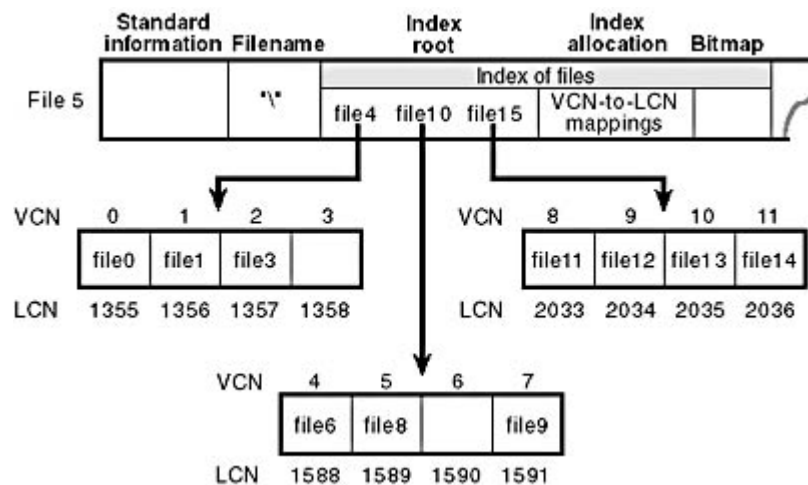
Một vấn đề nảy sinh là nếu cần quá nhiều record MFT thì sẽ không đủ chỗ

trong MFT cơ sở để liệt kê tất cả các chỉ mục của chúng. Có một giải pháp cho vấn đề này là thực hiện không lưu danh sách phần đuôi mở rộng của các record MFT (chẳng hạn, được lưu trữ trên đĩa thay vì trên bản ghi MFT cơ sở). Lúc đó, kích thước của file có thể phát triển lớn đến một mức cần thiết.

## Một số kỹ thuật được hỗ trợ bởi hệ thống file NTFS

### Lập bảng chỉ mục

Trong hệ thống file NTFS, một danh mục file là một chỉ mục đơn của các tên file, đó là một tập các tên file (cùng với các tham chiếu file của chúng) được tổ chức theo một cách đặc biệt để tăng tốc độ truy xuất file. Để tạo một danh mục, NTFS lập chỉ mục cho thuộc tính filename của các file trong thư mục. Một record cho thư mục gốc của volume được đưa ra ở hình 4.23 sau đây.



**Hình 4.23:** Chỉ mục tên file cho thư mục gốc của volume

Một entry MFT cho một thư mục chứa trong thuộc tính index root của nó một danh sách được sắp xếp của các file trong thư mục. Đối với các thư mục lớn, tên file thực tế được lưu trữ trong các vùng đệm chỉ mục (index buffer) có kích thước cố định là 4Kb, index buffer này chứa và tổ chức các tên file. Index buffer cài đặt cấu trúc dữ liệu cây b+, nhờ đó mà cực tiểu được số lần truy cập trực tiếp đĩa cần thiết để tìm đến một file, đặc biệt là đối với các thư mục lớn. Thuộc tính Index root chứa cấp đầu tiên của cây b+ và trỏ đến Index buffer chứa cấp tiếp theo.

Hình 4.23 trình bày các tên file trong thuộc tính index root và index buffer (file5), nhưng mỗi entry trong index cũng chứa tham chiếu file trong MFT, nơi chứa các thông tin mô tả, kích thước, timestamp của file. NTFS nhân đôi thông tin về timestamp và kích thước file từ record MFT của file. Kỹ thuật này được sử dụng bởi FAT và NTFS, yêu cầu được cập nhật thông tin để ghi vào cả hai nơi. Do đó, nó tăng tốc độ đáng kể cho các thao tác duyệt thư mục vì nó cho phép hệ thống file hiển thị timestamp và kích thước file của mỗi file mà không cần mở mỗi file trong

thư mục.

Thuộc tính index allocation ánh xạ các VCN của cả Run index buffer mà nó chỉ báo nơi index buffer thường trú trên đĩa. Thuộc tính bitmap theo dõi các VCN trong index buffer là đang được sử dụng hay đang rỗi. Hình trên cho thấy một entry file trên VCN, nhưng các entry filename thực tế được đóng gói trong mỗi cluster. Một index buffer 4Kb có thể chứa từ 20 đến 30 entry filename.

Cấu trúc dữ liệu cây b+ là một kiểu cây cân bằng, nó là ý tưởng cho việc tổ chức sắp xếp dữ liệu được lưu trữ trên đĩa bởi vì nó cực tiểu số lần truy cập đĩa cần thiết để tìm đến một entry. Trong MFT, một thuộc tính index root của thư mục chứa nhiều filename mà nó đóng vai trò như là các chỉ mục vào cấp thư hai của cây b+. Mỗi filename trong thuộc tính index root có một con trỏ tùy chọn được kết hợp với nó để chỉ đến index buffer. Index buffer mà nó chỉ đến chứa các filename với giá trị (về mặt tự điển) ít hơn sở hữu của nó. Trong hình trên, file4 là entry cấp đầu tiên trong cây b+, nó chỉ đến một index buffer chứa các filename mà ít hơn chính nó, đó là các filename file0, file1, và file3.

Lưu trữ tên file trong cây b+ mang lại nhiều thuận lợi. Việc tìm kiếm thư mục sẽ nhanh hơn vì filename được lưu trữ theo thứ tự được sắp xếp. Và khi một phần mềm cấp cao đếm các file trong thư mục, NTFS sẽ trả lại tên file vừa được sắp xếp.

NTFS cũng cung cấp sự hỗ trợ cho chỉ mục dữ liệu bên cạnh filename. Một file có thể có một đối tượng ID được gán cho nó, ID của file được lưu trữ trong thuộc tính \$OBJECT\_ID của file. NTFS cung cấp một API cho phép các ứng dụng mở file bằng các đối tượng ID của file thay vì dùng tên file của nó. Do đó, NTFS phải tạo ra một tiến trình để chuyển đổi một đối tượng ID thành số file của file một cách hiệu quả. Để thực hiện được điều này NTFS lưu trữ một ánh xạ của tất cả các đối tượng ID của volume thành số tham chiếu file của chúng trong một file metadata \ \$Extend \ \$ObjID. NTFS sắp xếp các đối tượng ID trong file nói trên như chỉ mục filename mà ta đã đề cập ở trên. Chỉ mục đối tượng ID được lưu trữ như là cây b+.

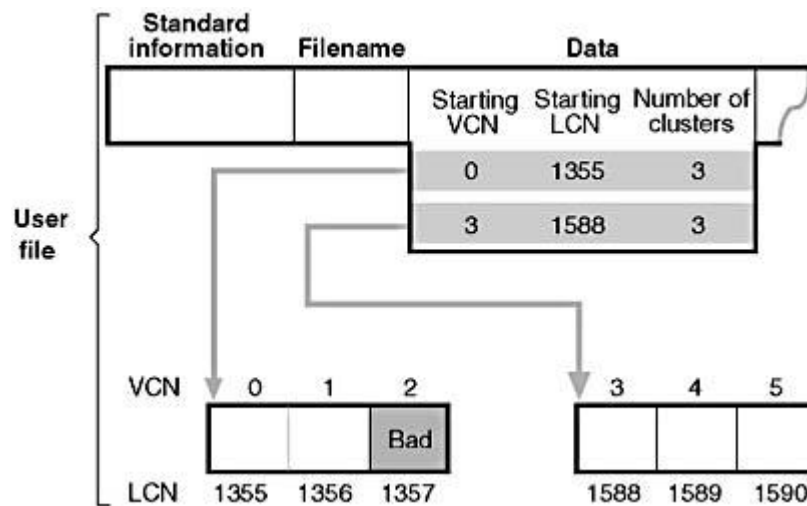
### Ánh xạ bad-cluster

Các thành phần quản lý đĩa logic trong windows 2000 như FDISK (đối với basic disk) và LDM (đối với các dynamics disk), có thể khôi phục dữ liệu tại các bad-sector trên các đĩa có khả năng chịu lỗi (fault tolerant disk), với điều kiện đĩa phải sử dụng chuẩn SCSI và còn các sector trống trên đĩa. Các volume chịu lỗi là các volume thuộc loại Mirrored và RAID-5. Hệ thống file FAT và thành phần quản lý đĩa logic của nó không thể đọc dữ liệu từ các bad-sector cũng như không thể phát sinh thông báo khi ứng dụng đọc dữ liệu tại bad-sector.

NTFS thay thế một cách tự động các cluster chứa bad-sector và theo dõi bad-

cluster vì thế nó không được sử dụng lại sau này. Khi bộ phận quản lý volume trả về cảnh báo bad-sector hoặc khi bộ điều khiển đĩa cứng trả về lỗi bad-sector, NTFS sẽ tìm một cluster mới để thay thế cluster chứa bad-sector. NTFS copy dữ liệu mà bộ phận quản lý volume khôi phục được vào cluster mới để thiết lập lại sự dư thừa dữ liệu.

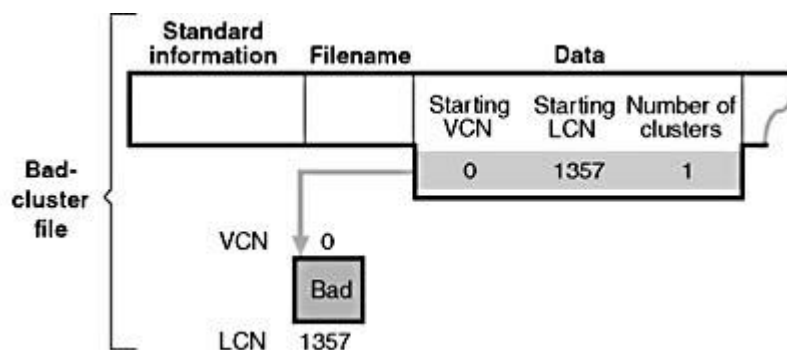
Hình 4.24.a cho thấy một record MFT cho một file của người sử dụng với một bad-cluster trong một trong các Run dữ liệu của nó. Khi nó nhận được lỗi bad-sector, NTFS gán lại cluster chứa bad-sector vào tập bad-cluster của nó. Điều này ngăn cản hệ thống cấp bad-cluster cho các file khác. Sau đó NTFS tìm một cluster mới cho file và thay đổi ánh xạ VCN-to-LCN để chỉ đến cluster mới.



**Hình 4.24.a:** Record MFT cho một File có bad-cluster

Ánh xạ lại bad-cluster trong hình sau. Cluster 1357 có chứa bad-sector, được thay thế bằng cluster mới 1049.

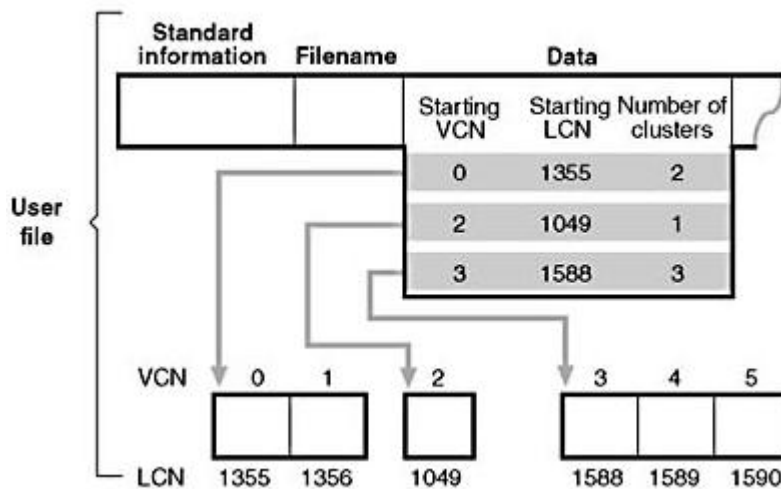
Nếu bad-sector ở trên volume redundant, thì bộ phận quản lý volume sẽ khôi phục dữ liệu và thay thế sector nếu có thể. Nếu không thể thay thế sector thì nó sẽ trả về một cảnh báo cho NTFS và NTFS sẽ thay thế cluster chứa bad-sector đó.



**Hình 4.24.b1:** Ánh xạ lại bad-cluster

Nếu một volume không được cấu hình là volume redundant thì dữ liệu trong bad-sector không thể khôi phục được. Khi một volume được định dạng như là một volume FAT và bộ phận quản lý volume không thể khôi phục dữ liệu thì việc đọc dữ liệu từ bad-sector sẽ không thành công và cũng không nhận được kết quả trả lời. Nếu các thành phần quan trọng của hệ điều hành đang được chứa tại các bad cluster thì có thể toàn bộ các file thư mục trên volume sẽ bị mất.

Giống như các hệ thống file khác, NTFS không thể khôi phục dữ liệu từ bad-sector mà không có sự hỗ trợ từ bộ phận quản lý volume. Tuy nhiên, NTFS chứa nhiều sự hư hại mà các bad-sector có thể gây ra. Nếu NTFS phát hiện ra bad-sector trong quá trình đọc nó sẽ ánh xạ lại cluster chứa bad-sector trong nó, như trình bày trong hình 4.24.b2 sau đây:



**Hình 4.24.b2:** Ánh xạ lại bad-cluster

Nếu volume không được cấu hình là volume redundant, NTFS trả lại thông báo lỗi “đọc dữ liệu” cho chương trình người sử dụng yêu cầu đọc dữ liệu. Nếu NTFS phát hiện bad-cluster trong thao tác ghi, thì NTFS sẽ ánh xạ lại cluster trước khi ghi, nên không bị mất dữ liệu cũng như không phát sinh lỗi.

## Tổ chức lưu trữ file trên đĩa CD\_ROM

Về nguyên tắc hệ thống file trên CD\_ROM đơn giản hơn so với những hệ thống file khác, vì các đĩa CD\_ROM chỉ được ghi một lần (write-once media), do đó các file ghi trên nó không thể xóa bỏ hay thay đổi sau khi đĩa đã được chế tạo, chính vì vậy thành phần quản lý File/đĩa của hệ điều hành sẽ không lo đến việc quản lý các Block còn tự do trên đĩa cũng như việc cấp phát và thu hồi các Block cho các file, trong trường hợp các file được lưu trữ trên đĩa CD\_ROM.

Sau đây chúng ta xem xét hệ thống file chính trên CD\_ROM và 2 hệ thống mở rộng của chúng:

**Hệ thống file ISO 9660:** Đây là chuẩn phổ biến nhất đối với các hệ thống file CD\_ROM và đã được chấp nhận như một chuẩn quốc tế vào năm 1988 với cái tên

ISO 9660. Một trong những mục đích của chuẩn này là làm cho tất cả các CD\_ROM đều có thể đọc được trên các máy tính khác nhau, nó không phụ thuộc vào thứ tự byte cũng như hệ điều hành đang sử dụng, kể cả hệ điều hành yếu nhất như MS\_DOS.

Trên CD\_ROM không có track, cylinder như trên các đĩa từ, nó chỉ có một đường xoắn ốc đi từ tâm đĩa ra bên ngoài, đường xoắn ốc này được chia thành các khối (block) logic có kích thước bằng nhau và bằng 2352 byte, đôi khi cũng được gọi là các sector logic. Một vài byte trong khối dành cho phần mở đầu, sửa chữa lỗi, và những việc khác. Phần chính của mỗi khối logic còn lại khoảng 2048 byte.

ISO 9660 hỗ trợ cho một tập đĩa CD\_ROM với một tập gồm  $2^{16}-1$  đĩa, một CD\_ROM riêng lẻ có thể được chia thành nhiều partition. Trong phần này chúng ta chỉ tìm hiểu chuẩn ISO 9660 với một CD\_ROM và không được chia thành các Partition.

- Mỗi CD\_ROM đều có phần đầu của đĩa, dài 16 block, chức năng của phần này không được định nghĩa trong chuẩn ISO 9600. Các nhà sản xuất CD\_ROM có thể sử dụng phần đầu này để ghi vào đó chương trình BootStrap cho phép máy tính có thể khởi động được từ đĩa CD\_ROM, hoặc dùng cho những mục đích khác.
- Phần tiếp theo là 1 block chứa bộ mô tả Volume chính, bộ mô tả này chứa một số thông tin chung về CD\_ROM, bao gồm: định danh hệ thống (32byte), định danh volume (32byte), định danh nhà sản xuất (128byte) và định danh dữ liệu (128byte). Khi chế tạo có thể lấp đầy những trường trên theo ý muốn. Trong phần này còn chứa phần giới thiệu, bản quyền tác giả, thông tin thư mục, kích thước của một khối logic (2048, 4096, 8192, ...), số các block trên CD\_ROM, và thời gian tạo và kết thúc của CD\_ROM. Cuối cùng, trong bộ mô tả Volume chính còn chứa một tập các mục vào (directory entry) cho thư mục gốc, tại đây chứa địa chỉ của block bắt đầu của thư mục gốc trên CD\_ROM. Trên CD\_ROM có 2 bộ mô tả volume chính, có nội dung hoàn toàn giống nhau, sử dụng một bộ và một bộ để dự phòng.
- Sau các phần trên là phần bắt đầu của CD\_ROM dùng để chứa các file đang được ghi trên đĩa.
- Thư mục gốc và tất cả các thư mục khác, chỉ gồm một số mục vào, phần cuối của chúng chứa một bit đánh dấu (mark). Mỗi mục vào chứa từ 10 đến 12 trường, trong đó có một số thuộc ASCII và số khác là những trường số thuộc số nhị phân.

**Mở rộng Rock Ridge:** Các chuyên viên thiết kế của UNIX nhận thấy ISO 9660 còn một vài hạn chế, do đó họ đã mở rộng ISO 9660 với mục đích là cho nó có thể thay thế cho hệ thống file của UNIX trên các đĩa CD\_ROM và các file được tạo từ



UNIX có thể được sao chép sang CD\_ROM và ngược lại, chuẩn mở rộng này được gọi là Rock Ridge.

Rock Ridge giữ lại tất cả các trường của ISO 9660, và sử dụng trường System để đưa thêm vào các trường mới, các hệ thống file khác không nhận biết các trường này và xem CD\_ROM như một đĩa CD\_ROM thông thường. Rock Ridge bổ sung thêm các trường, theo thứ tự là: PX: Posix Attributes, PN: Major and minor device number, SL: Symbolic link, NM: Alternative name, CL: Child location, PL: Parent location, RE: Relocation, TF: Times stamps, trong đó trường quan trọng nhất là NM, trường này cho phép sử dụng 2 tên file cho một file, một tên file trong mục vào của thư mục và một tên file kết hợp, tên này không phụ vào tập kí tự hoặc giới hạn chiều dài của chuẩn ISO 9660.

**Mở rộng Joliet:** Cũng như các chuyên viên thiết kế của UNIX, các chuyên viên thiết kế của Microsoft muốn mở rộng ISO 9660 sao cho các file được tạo từ Windows có thể được sao chép sang CD\_ROM và ngược lại và họ đã thành công với mở rộng Joliet. Mở rộng Joliet cho phép: Tên file dài đến 64 kí tự; Sử dụng tập kí tự Unicode nên tên file có thể dài đến 128 kí tự; Có nhiều hơn 8 cấp thư mục lồng nhau; Sử dụng tên thư mục với phần mở rộng.

## **TÀI LIỆU THAM KHẢO**

---

1. Nguyễn Thanh Tùng. Bài giảng Hệ điều hành. Đại học Bách khoa Hà Nội, 1996.
2. Trần Hạnh Nhi. Giáo trình Hệ điều hành Nâng Cao. Đại học Khoa học Tự nhiên, TP Hồ Chí Minh, 1998.
3. Dương Quang Thiện. Hệ điều hành MSDOS 6.22. Văn phòng SAMIS, TP Hồ Chí Minh, 1995.
4. Lê Mạnh Thanh - Nguyễn Kim Tuấn. Hướng dẫn lập trình với Assembly. Nhà xuất bản Khoa học Kỹ thuật Hà Nội, 2001.
5. Michael Tischer. Cẩm nang Lập trình Hệ thống (tập I và tập II). Nguyễn Mạnh Hùng và Phạm Tiến Dũng dịch. Nhà xuất bản Thống Kê, 1995.
6. William Stalling. Operating Systems. Prentice Hall, 1995.
7. Andrew S. Tanenbum. Modern Operating Systems. Prentice Hall, 1995.
8. David A. Solomon – Mark E. Russinovich. Inside Microsoft Windows 2000. Microsoft Press, 2000.