CHƯƠNG 4

LẬP TRÌNH HỆ THỐNG TRÊN WINDOWS

Khi máy tính PC được chế tạo với các tính năng như tốc độ ngày càng cao, độ rộng của kênh dữ liệu ngày càng lớn cùng các các tính năng khác ngày càng hoàn thiện thì hệ điều hành để điều khiển hoạt động của nó phải có khả năng đa nhiệm nhằm khai thác triệt để sức mạnh mới của máy tính. Các hãng phần mềm đã đưa ra nhiều hệ điều hành đa nhiệm khác nhau, trong đó nổi bật nhất là hệ điều hành của hãng MICROSOFT - hệ điều hành đa nhiệm dòng WINDOWS. Trong kỹ thuật lập trình trên WINDOWS cần phải chuyển tư duy sang một hướng khác - lập trình hướng đối tượng, cho phép tiếp cận và khai thác tài nguyên vốn vô cùng phong phú của MICROSOFT WINDOWS.

4.1. HỆ ĐIỀU HÀNH WINDOWS

Để hỗ trợ tốt hơn cho người sử dụng trên máy tính PC, hãng Microsoft liên tục phát triển các hệ điều hành mới cho máy tính PC và hiện nay hệ điều hành WINDOWS đang được dùng phổ biến với nhiều tính năng mạnh ưu việt hơn hẳn hệ điều hành DOS và hệ điều hành WINDOWS 3.x.

Hệ điều hành của Windows bao gồm các thành phần sau đây :

- ***Bộ quản lý máy ảo VMM***

Bộ quản lý máy ảo VMM (Virtual Machine Manager) là thành phần hạt nhân của hệ điều hành Windows. Nhiệm vụ chính của VMM là tạo, thực hiện, giám sát và kết thúc hoạt động một cách hợp thức của các

máy ảo. VMM cung cấp các dịch vụ như quản lý bộ nhớ, xử lý ngắt và ngăn chặn các lỗi hệ thống. VMM làm việc với các thiết bị ảo (virtual devices), các module 32-bit ở chế độ bảo vệ (protected-mode), cho phép các thiết bị ảo ngăn chặn các ngắt và các lỗi để điều khiển truy nhập của một ứng dụng tới các thiết bị phần cứng và cài đặt phần mềm cho các thiết bị phần cứng đó.

VMM cung cấp khả năng xử lý đa nhiệm. Nó thực hiện đồng thời nhiều ứng dụng cùng lúc bằng cách chia sẻ thời gian hoạt động của CPU.

- ***Các thiết bị ảo VxD***

Các thiết bị ảo *(Virtual devices - VxD)* là các chương trình 32 bit hỗ trợ các thiết bị độc lập với VMM bằng cách quản lý các thiết bị phần cứng (hardware devices) của máy tính và hỗ trợ phần mềm xử lý. VxD hỗ trợ tất cả các thiết bị phần cứng cho các máy tính PC bao gồm các bộ điều khiển ngắt lập trình được (Programmable Interrupt Controller - PIC), điều khiển thời gian (Timer), truy nhập trực tiếp bộ nhớ (Direct Memory Access - DMA), điều khiển các thiết bị (device), điều khiển đĩa (disk controller), điều khiển cổng nối tiếp (serial ports), điều khiển cổng song song (parallel ports), điều khiển bàn phím (keyboard), và điều khiển màn hình hiển thị (display adapter). Một VxD được yêu cầu cho bất cứ thiết bị phần cứng nào nếu thiết bị đó được thiết lập ở chế độ làm việc. Nói cách khác, nếu trạng thái của thiết bị phần cứng có thể bị phá vỡ hoặc thay đổi bởi việc chuyển đổi giữa nhiều máy ảo (virtual machines) hoặc giữa nhiều ứng dụng, thì thiết bị đó bắt buộc phải có một VxD phù hợp.

Nói chung, một VxD có thể cung cấp nhiều loại dịch vụ cho VMM và các thiết bị ảo khác. Windows cho phép người sử dụng cài đặt các chương trình điều khiển thiết bị ảo mới để hỗ trợ cho một thiết bị phần cứng mới được thêm vào máy tính hoặc cung cấp một vài dịch vụ mới cho hệ thống.

Một VxD cũng có thể cung cấp các hàm giao diện API (Application Programming Interface) cho các ứng dụng chạy trong chế độ mô phỏng

80X86 hoặc chế độ bảo vệ. Các hàm này có thể cho phép các ứng dụng truy nhập trực tiếp tới các tính năng của VxD.

Windows có một giao diện điều khiển các thiết bị vào/ra (Input and Output Controll - IOCTL) cho phép các ứng dụng dựa trên môi trường Microsoft Win32 có thể truyền thông trực tiếp với các VxD. Các ứng dụng sử dụng giao diện này thực hiện các hàm hệ thống của MS-DOS để thu lượm các thông tin về một thiết bị, hoặc thực hiện các thao tác vào/ra không có sẵn trong các hàm chuẩn của Wind32.

- *Các chương trình điều khiển thiết bị (Device drivers)*

Một chương trình điều khiển thiết bị trong Windows được đặt trong thư viện liên kết động được Windows sử dụng để tương tác với thiết bị phần cứng, chẳng hạn như bàn phím hoặc màn hình... Thay vì truy nhập trực tiếp tới một thiết bị phần cứng, Windows nạp trình điều khiển cho thiết bị đó và gọi các hàm trong trình điều khiển thiết bị để thực hiện các tác động trên thiết bị đó. Mỗi trình diều khiển thiết bị cung cấp một tập hợp các hàm chức năng. Windows gọi các hàm chức năng này để hoàn thành một thao tác đối với thiết bị bị điều khiển, chẳng hạn như truyền 1 byte ra cổng COM hoặc dịch mã quét của một phím được ấn. Các hàm chức năng của một trình điều khiển thiết bị cũng bao gồm các mã lệnh đặc trưng cần thiết của thiết bị để thực hiện các hành động trên thiết bị đó.

Windows đòi hỏi phải có các trình điều khiển thiết bị tương ứng cho màn hình hiển thị, bàn phím và các cổng truyền thông thông thường (như cổng nối tiếp và cổng song song). Các trình điều khiển thiết bị khác cũng có thể được yêu cầu nếu người dùng bổ sung thêm các thiết bị mới vào hệ thống.

- *Các thư viện liên kết động DLL*

Trong Microsoft Windows, thư viện liên kết động DLL (Windows dynamic-link libraries) là các module chứa các hàm và dữ liệu. Một thư viện liên kết động được nạp tại thời điểm thực hiện (runtime) bằng cách gọi các module của nó (.EXE or.DLL). Khi một thư viện liên kết động đã

được nạp, nó được sắp xếp vào vùng địa chỉ trống của tiến trình gọi thực hiện các hàm của thư viện liên kết động đó (Tiến trình cha).

Các thư viện liên kết động có thể định nghĩa hai kiểu hàm: Các hàm tổng thể và các hàm cục bộ. Các hàm tổng thể có thể được gọi bởi các module khác nhau. Các hàm cục bộ chỉ có thể bị gọi bên trong thư viện liên kết động nơi mà chúng được định nghĩa. Mặc dù một thư viện liên kết động có thể kết xuất dữ liệu, nhưng các dữ liệu của nó thường chỉ được sử dụng bởi chính các hàm của các thư viện động đó. Các DLL giúp giảm bộ nhớ khi nhiều ứng dụng sử dụng cùng một hàm chức năng ở cùng một thời điểm.

Liên kết động cung cấp một kỹ thuật để liên kết các ứng dụng với các thư viện của các hàm ở thời điểm hoạt động (Run Time) của các ứng dụng đó. Các thư viện được lưu trữ trong các file thực hiện được và không được sao vào file của ứng dụng như một liên kết tĩnh. Các thư viện này là liên kết động bởi vì chúng được liên kết với một ứng dụng khi ứng dụng đó được nạp và thực hiện. Khi một ứng dụng sử dụng một DLL, hệ điều hành nạp DLL vào bộ nhớ, thiết lập tham chiếu tới các hàm trong DLL sao cho ứng dụng có thể gọi được chúng, và DLL đó được loại bỏ khỏi bộ nhớ khi nó không còn cần thiết cho ứng dụng nữa.

DLL được thiết kế để cung cấp tài nguyên cho các ứng dụng. Nhiều ứng dụng có thể sử dụng mã trong một DLL, nghĩa là chỉ một bản sao của mã lệnh thường trú trong hệ thống. Ngoài ra cũng có thể nâng cấp hoặc sửa đổi DLL mà không cần sửa đổi ứng dụng nếu như các giao diện của các hàm trong DLL không thay đổi.

Những nhà phát triển phần mềm có thể mở rộng môi trường của Windows bằng cách tạo các DLL mới và thêm nó vào hệ thống để hỗ trợ các ứng dụng của Windows.

- ***Các thông báo hệ thống***

Windows quản lý và điều khiển các ứng dụng thông qua một tập hợp các thông báo hệ thống (system messages). Các thông báo hệ thống cung cấp cách thức để báo cho tất cả các ứng dụng cũng như các thành

phần khác của hệ thống những thay đổi có thể ảnh hưởng đến hoạt động và sự truy nhập tới các tài nguyên hệ thống của chúng.

Các ứng dụng và các thành phần khác của hệ thống cũng có thể định nghĩa các thông báo hệ thống của riêng chúng và sử dụng các thông báo đó để phục vụ cho việc thông báo về các kiểu sự kiện khác nhau trong hệ thống.

Mỗi một thông báo hệ thống bao gồm một định danh và hai tham số 32-bit, wParam và lParam. Định danh của thông báo hệ thống là một giá trị phân biệt dùng để chỉ định mục đích của thông báo đó. Các tham số cung cấp thêm các thông tin cần thiết, trong đó tham số wParam thường là một giá trị thông báo.

Một thông báo hệ thống có thể được gửi tới một hoặc nhiều thành phần trong hệ thống. Các thành phần này có thể là các ứng dụng, các trình điều khiển thiết bị, các trình điều khiển mạng của Windows hoặc các trình điều khiển thiết bị ở mức hệ thống.

Hầu hết các ứng dụng không gửi đi các thông báo hệ thống, mà chúng nhận và xử lý các thông báo hệ thống được gửi tới từ các thành phần khác của hệ thống. Hệ điều hành thường gửi các thông báo hệ thống phản ánh sự thay đổi của các trình điều khiển thiết bị ở mức hệ thống. Trình điều khiển thiết bị hoặc các thành phần liên quan thường sản sinh ra các thông báo hệ thống, sau đó gửi chúng tới các ứng dụng và các thành phần khác để báo cho chúng những thay đổi. Chẳng hạn, hệ thống con chịu trách nhiệm quản lý đĩa sản sinh và gửi các thông báo hệ thống khi trình điều khiển thiết bị cho đĩa mềm nhận biết sự thay đổi khi người dùng đưa đĩa vào ổ đĩa.

Các ứng dụng nhận được thông báo hệ thống thông qua thủ tục cửa sổ của cửa sổ làm việc chính của các ứng dụng đó. Thông báo hệ thống không được gửi tới các cửa sổ con của ứng dụng (đối với các ứng dụng có nhiều cửa sổ). Khi đó tùy thuộc vào nội dung của thông báo, mỗi ứng dụng có thể có những hành động tương ứng. Một vài thông báo hệ thống,

được gọi là các thông báo hàng đợi, đòi hỏi các ứng dụng phải trả lại hoặc giá trị TRUE hoặc BROADCAST_QUERY_DENY để chỉ định liệu hệ thống có nên tiếp tục gửi các thông báo tới những nơi nhận khác không.

Người lập trình có thể tạo các thông báo hệ thống của riêng mình để xác định các hành động giữa các ứng dụng và các thành phần khác của hệ thống. Điều này đặc biệt hữu ích nếu người lập trình phải tạo các trình điều khiển thiết bị hoặc các trình điều khiển thiết bị ở mức hệ thống. Những thống báo hệ thống mới đó có thể gửi hoặc nhận các thống tin giữa trình điều khiển thiết bị và các ứng dụng có sử dụng thiết bị của trình điều khiển đó.

Người lập trình cũng có thể gửi các thông báo do họ tạo ra tới các thành phần trong hệ thống. Các thông báo đó sẽ được gửi tới các nơi nhận theo thứ tự: Các trình điều khiển thiết bị mức hệ thống, các trình điều khiển mạng của Windows, các trình điều khiển thiết bị khác và các ứng dụng. Điều này có nghĩa là các trình điều khiển thiết bị mức hệ thống, nếu được chọn là nơi nhận, luôn luôn nhận được các thông báo đầu tiên. Trong cùng một kiểu những nơi nhận, không có trình điều khiển nào được đảm bảo sẽ nhận được các thông báo trước các trình điều khiển khác. Điều này có nghĩa là một thông báo muốn được gửi cho một trình điều khiển nhất định nào đó bắt buộc phải có một định danh phân biệt tổng thể để sao cho các trình điều khiển thiết bị khác không có ý định xử lý nó.

Windows có một cơ chế hàng đợi các thông báo cho phép chọn lựa những nơi nhận đã được trao quyền nhận thông báo để thực hiện một hành động nào đó. Người lập trình có thể tạo ra hàng đợi thông báo cho riêng mình. Với các hàng đợi này, người lập trình có thể chỉ ra thứ tự nhận các thông báo của các thành phần cần nhận thông báo (các nơi nhận) trong hệ thống. Mỗi nơi nhận thông báo của hàng đợi này phải trả lại giá trị để báo cho hàng đợi gửi thông báo cho nơi nhận tiếp theo hoặc kết thúc việc gửi thông báo. Người lập trình cũng có thể tạo các trình điều khiển để gửi và xử lý các thông báo.

4.2. QUẢN LÝ BỘ NHỚ CỦA HỆ ĐIỀU HÀNH WINDOWS

Hệ điều hành Windows cho phép truy xuất một không gian nhớ dung lượng nhiều GB. Trong chế độ bảo vệ, hệ điều hành điều khiển tác vụ nạp chương trình phải xác lập một bảng mô tả thông tin địa chỉ bộ nhớ. Cách xử lý chế độ bảo vệ tính toán một địa chỉ vật lý ở trên các CPU x86 cũng giống như ở trên CPU cơ sở 286. Bộ xử lý dùng nội dung của một thanh ghi phân đoạn như một chỉ dẫn tới một bảng mô tả, và việc nhập liệu vào bảng mô tả chứa đựng hầu hết những thông tin còn lại. Hầu như x86 cho phép hệ điều hành thi hành một sơ đồ bộ nhớ ảo phân trang hoàn hảo. Khi hệ điều hành có thể phân trang bộ nhớ, thông tin địa chỉ được rút ra từ bảng mô tả phải đi qua một cấp diễn giải cao hơn trước khi được sử dụng như một địa chỉ bộ nhớ thực.

4.2.1. Quản lý bộ nhớ

Quản lý bộ nhớ (memory management) trong Windows được phân thành hai mức độ khác nhau: một mức độ lập trình viên có thể nhìn thấy và một mức độ chỉ hệ điều hành nhìn thấy mà thôi.

Không gian địa chỉ hệ thống System address space (not addressable by application)	4GB
Không gian địa chỉ chia xẻ Shared address space	3GB
Không gian địa chỉ riêng Private address space	2GB

Hình 4.1. **Ánh xạ bộ nhớ ảo dành cho trình ứng dụng**

Hình 4.1 minh họa cách bố trí của bộ nhớ ảo trong trình ứng dụng Win32. Mỗi trình ứng dụng Win32 lại có một ánh xạ bộ nhớ tương ứng và mỗi không gian địa chỉ đều là duy nhất. Tuy nhiên, bộ nhớ vẫn không được bảo vệ trọn vẹn: bộ nhớ riêng được cấp phát cho một trình ứng dụng Win32 có thể bị trình ứng dụng khác gán địa chỉ tranh chấp. Không gian địa chỉ riêng của trình ứng dụng Win32 lại cũng là miền mà

trong đó hệ thống cấp phát bộ nhớ để thỏa mãn nhu cầu đòi hỏi của trình ứng dụng khi đang được thực thi trên máy.

Không gian địa chỉ hệ thống được dùng để ánh xạ những thư viện liên kết động (DLL) trong hệ thống vào địa chỉ không gian của trình ứng dụng. Những lệnh gọi tới những thư viện liên kết động của hệ thống trở thành những lệnh gọi tới vùng này. Những trình ứng dụng cũng có thể yêu cầu cấp phát bộ nhớ động bằng phương tiện của những địa chỉ ảo được ánh xạ vào vùng phân chia. Những địa chỉ ảo được ánh xạ vào không gian địa chỉ phân chia phục vụ nhu cầu kiểm soát sự phân chia bộ nhớ cho những trình ứng dụng khác nữa.

- ***Bộ nhớ ảo***

Bộ nhớ ảo là một phương pháp cho phép nhiều chương trình cùng chạy một lúc có thể dùng chung một bộ nhớ vật lý của một máy tính. Chế độ ảo ở đây muốn nói tới thao tác của bộ vi xử lý x86 ở trong chế độ ảo. Sự quản lý bộ nhớ ảo được hoàn toàn đặt dưới sự kiểm soát của hệ điều hành. Bất cứ một chương trình riêng rẽ nào đều cũng có thể truy xuất được bộ nhớ cần thiết vào bất cứ lúc nào. Thí dụ, chúng ta chạy hệ Widows trên một hệ thống có 4MB bộ nhớ và một đĩa cứng còn trống nhiều, ngay Windows với những thành phần như shell, bộ quản lý in ấn, v.v... đã chiếm một bộ nhớ hơn 1 MB. Còn ở trên đĩa là một chương trình xử lý văn bản mà người dùng định dùng tới. Một khi được nạp vào bộ nhớ, chương trình này phải chiếm 2 MB, và người dùng phải nạp vào bộ nhớ một tài liệu lớn bao gồm nhiều *font* chữ khác nhau. Như thế, tài liệu này chiếm tới 400 KB của MB còn lại trong bộ nhớ. Giờ đây, người dùng định nhập thêm một bảng số vào trong tài liệu đó. Những số liệu này nằm trong phần mềm bảng tính điện tử, và người dùng lại phải chạy một trình ứng dụng bảng tính điện tử để cắt, dán và sao chép vào tài liệu. Như thế Windows bắt buộc phải nạp trình ứng dụng bảng tính điện tử và dữ liệu vào trong bộ nhớ còn lại là 624 KB. Nếu như người dùng lại dùng phần mềm DELPHI thì phần mềm này và dữ liệu không thể chứa nổi vào bộ nhớ còn lại. Cứ theo cách nghĩ, khó lòng có thể chạy nổi chương trình trên theo như những điều đã mô tả. Song, hệ thống và cả

hai trình ứng dụng đó vẫn chạy suôn sẻ như thể bộ nhớ vẫn còn trống rất nhiều. Mọi chuyện xảy ra không phải ở bên trong 4 MB sẵn có của bộ nhớ vật lý, mà là xảy ra ở trong bộ nhớ ảo.

- ***Quản lý bộ nhớ ảo***

Bộ nhớ ảo của hệ thống được tạo lập nhờ RAM ở trong máy tímh và tập tin Windows tráo đổi trên đĩa cứng. Hệ điều hành quản lý tất cả bộ nhớ sẵn có bằng cách tráo đổi những phân đoạn chương trình và dữ liệu từ RAM vào tập tin tráo đổi. Những chỉ thị trong một phân đoạn mã đặc biệt được thi hành thì phân đoạn đó phải được nạp vào RAM, những phân đoạn mã khác vẫn có thể ở trên đĩa trong tập tin tráo đổi nếu không được dùng tới. Một vùng bộ đệm dữ liệu đĩa ở bên trong một phân đoạn dữ liệu phải ở trong RAM nếu chuyển đổi đĩa thành công. Mỗi khi một phân đoạn không nằm trong RAM thì hệ điều hành có thể đánh dấu vắng bóng phân đoạn đó bằng cách xóa bit Present nằm trong bộ mô tả phân đoạn thích hợp. Nếu truy xuất tới phân đoạn đó thì CPU x86 sẽ phát sinh một ngắt khiếm diện (not present interrupt) để báo cho hệ điều hành biết sự kiện này. Hệ thống sẽ dàn xếp để nạp đoạn bị thiếu đó vào một vùng sẵn có ở trong RAM, và rồi khởi động lại chương trình vốn đã gây ra ngắt đó.

Bộ xử lý từ 386 trở lên đã cải thiện mọi tác vụ trên bằng cách cho phép một kế hoạch phân trang bộ nhớ ảo khiến hệ điều hành có thể thi hành tất cả việc cấp phát bộ nhớ, giải phóng bộ nhớ và những tác vụ tráo đổi trong những đơn vị của những trang bộ nhớ. Trong bộ xử lý 386, một trang bộ nhớ là 4 KB, và mỗi phân đoạn bộ nhớ được tạo lập bằng một hay nhiều trang bộ nhớ 4 KB. Trong khi khởi tạo, hệ điều hành thoạt tiên chuyển bộ xử lý về chế độ bảo vệ, rồi làm tác vụ phân trang.. Khi đã phân trang xong, bộ xử lý 386 liền thay đổi các thông dịch địa chỉ 32 bit bằng cách thêm địa chỉ gốc từ bộ mô tả vào (offset) được chương trình phát sinh. Hệ điều hành giữ lấy toàn bộ cấu trúc bằng cách lưu trữ địa chỉ của thư mục bảng phân trang bộ nhớ dành cho chương trình hiện hành trong một thanh ghi đặc biệt tên là CR3. Mỗi lần chuyển đổi tác vụ, hệ điều hành lại nạp lại CR3 để báo cho chương trình mới biết thư

mục phân trang bộ nhớ (page directory). Để hỗ trợ những thao tác bộ nhớ ảo và hệ thống bảo vệ bộ nhớ x86, thì thư mục trang bộ nhớ và những nhập liệu bảng trang bộ nhớ phải chứa luôn cả một số bit cờ trạng thái. Ngoài ra, muốn để tương thích với những chương trình cũ viết bằng mã 16 bit, Windows lại dùng cả kỹ thuật *thunk* để chuyển đổi mã 32 bit thành 16 bit và ngược lại.

4.2.2. Hệ thống bảo vệ

Hệ điều hành Windows đã đề xuất những khả năng bảo vệ: bảo vệ dữ liệu cho người dùng, bảo vệ một chương trình này khỏi bị những chương trình khác lấn chiếm khi đang chạy trong máy, và bảo vệ những thiết bị vật lý không bị truy xuất trái phép.

- ***Bảo vệ bộ nhớ***

Mỗi khi một trình ứng dụng truy xuất một vị trí bộ nhớ không nằm trong ánh xạ bộ nhớ (memory map), bộ xử lý x86 liền phát sinh một ngắt và trao đổi cho hệ điều hành một tập thông tin liên hệ. Đôi khi, hệ điều hành phải dàn xếp để thêm một trang bộ nhớ thích hợp và ánh xạ bộ nhớ của trình ứng dụng đó. Với những phiên bản Windows 3.x, hệ điều hành cấp phát nhiều bộ nhớ cho trình ứng dụng. Nhưng thường nếu chạy nhiều trình ứng dụng tốn nhiều bộ nhớ thì hệ điều hành sẽ hiển thị một hộp thoại báo là bộ nhớ quá ít không thể tiếp tục thi hành, hoặc trình ứng dụng không thể làm được gì nữa. Vào trường hợp này, Windows 9x giải quyết bằng cách mở rộng số lượng những tài nguyên hệ điều hành sẵn có, đặc biệt khi những tài nguyên yêu cầu hệ thống được thỏa mãn do hệ điều hành cấp phát bộ nhớ từ vùng bộ nhớ động (memory pool) của chế độ bảo vệ 32 bit.

- ***Vành bảo vệ trong Windows***

Windows khai thác khả năng bộ xử lý Intel x86 để hỗ trợ những cấp độ ưu tiên phức tạp. Do tác vụ xử lý của những vành bảo vệ có khuynh hướng ảnh hưởng tới nhiều khía cạnh của cách thiết kế hệ thống, Windows xử lý bằng cách dùng những mức độ ưu tiên 0 và 3.

Những phần tử vành 0 là những gì người lập trình thường nghĩ tới như hệ điều hành. Phần mềm vành 0 là phần cơ sở của hệ điều hành và nó có mức ưu tiên cao nhất.

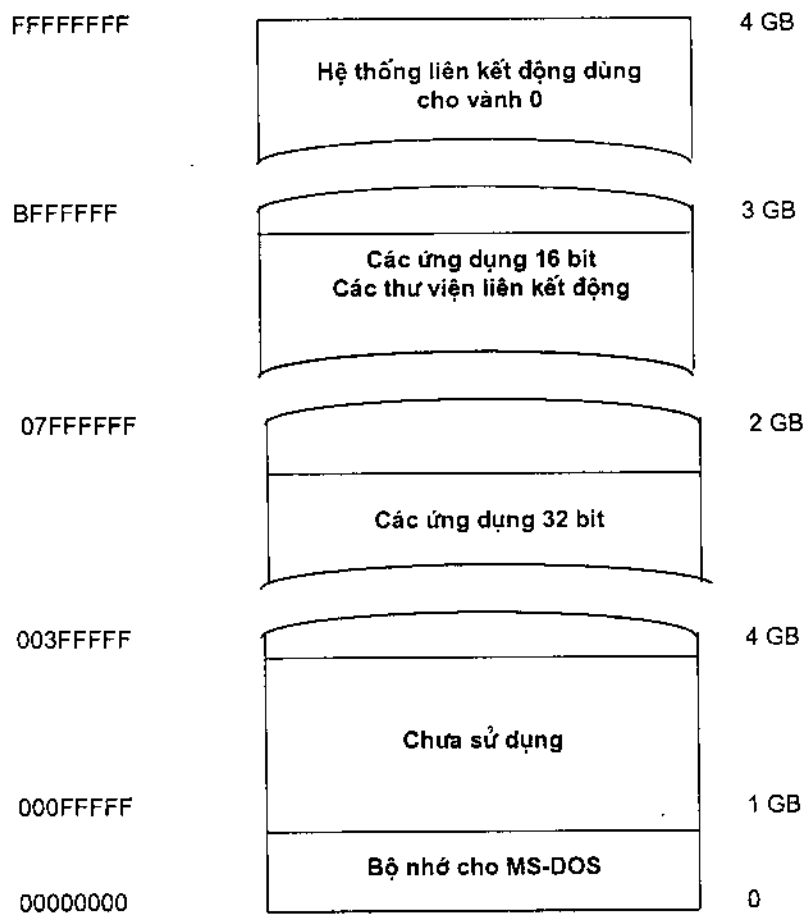
Những trình ứng dụng Windows và những trình ứng dụng hệ điều hành Microsoft luôn luôn chạy ở vành 3, do đó những quyền ưu tiên của nó thường bị hạn chế. Vì vậy, việc thâm nhập vào hệ thống rất khó mà phải có một số kỹ thuật lập trình đặc biệt với ngôn ngữ bậc thấp như Assembly hoặc ngôn ngữ bậc cao như C, Delphi... mới có thể thâm nhập được.

4.2.3. Ánh xạ bộ nhớ trong Windows

Bộ xử lý x86 quản lý được không gian địa chỉ ảo ltừ 1 GB trở lên. Trong không gian địa chỉ ảo này, những phần tử hệ thống khác nhau và những trình ứng dụng đều chiếm giữ những vùng có những giới hạn nhất định. Hình 4.2 cho ta thấy ánh xạ bộ nhớ cơ bản cho hệ thống là ánh xạ không gian địa chỉ ảo 4 GB vào trong bộ nhớ vật lý có sẵn.

Trong ánh xạ bộ nhớ hệ thống, vùng 1 MB thấp nhất của không gian địa chỉ ảo được dùng để thao tác máy ảo của hệ điều hành hiện hành. Mỗi một máy ảo cũng có riêng một ánh xạ bộ nhớ hợp thức trong vùng từ 2 tới 3 GB. Sự ánh xạ này cho phép hệ thống được gán địa chỉ cho một máy ảo bất kể máy đó hoạt động hay không. Nhưng khi một máy ảo của hệ điều hành Microsoft chạy, nó liền được ánh xạ vào đáy của 1 MB.

Trong không gian địa chỉ ảo của trình ứng dụng Windows 32 bit, những bộ công cụ chuẩn đều dùng 4 MB như là dung lượng mặc định và chúng được nạp ngay vào vùng từ 4 MB tới 2 GB. Địa chỉ nạp trình ứng dụng loại 4 MB sẽ làm phù hợp được với địa chỉ Windows NT được dùng để nạp những trình ứng dụng 32 bit ngay trong phiên bản sản phẩm đầu tiên. Vùng nhớ thấp nhất 16 KB của mỗi không gian địa chỉ của trình ứng dụng 32 bit (những địa chỉ ảo từ 0 tới 3FFFh) đều không hợp lệ.



Hình 4.2. **Tạo một máy ảo để tải file chạy vào bộ nhớ**

4.3. QUẢN LÝ FILE CỦA WINDOWS

- ***Vùng tiêu đề của file chạy (Portable Executable file - PE file)***

Điều quan trọng đầu tiên cần phải biết về vùng tiêu đề của file chạy (Portable Executable file - PE file) là một chương trình trên đĩa chính là hình ảnh của một module sau khi đã được Windows tải vào bộ nhớ. Ta sử dụng từ "module" để nói đến phần *code*, *data*, và *resource* của một file chạy hoặc thư viện liên kết động (DLL) được tải vào bộ nhớ. Trình nạp của Windows không cần phải làm gì nhiều để tạo ra một *process* từ file

trên đĩa, nó sử dụng cơ chế ánh xạ file trên bộ nhớ để ánh xạ các phần của file trên đĩa. Trong môi trường Windows 32 bit, phần bộ nhớ được sử dụng bởi một module là liên tục, ta chỉ cần phải xác định một điều là trình nạp sẽ ánh xạ file vào địa chỉ nào trên bộ nhớ.

Một vấn đề nữa cần chú ý là *địa chỉ ảo tương đối* (the Relative Virtual Address - RVA) vì trong nhiều trường file PE phải sử dụng RVA này. Một RVA đơn giản là một địa chỉ OFFSET của các đối tượng, địa chỉ OFFSET này có quan hệ với địa chỉ ánh xạ file trong bộ nhớ. Lấy ví dụ, ta nói trình nạp ánh xạ file trong bộ nhớ bắt đầu tại địa chỉ 0x10000 trong *vùng địa chỉ ảo*, nếu một thành phần nào đó của file bắt đầu tại địa chỉ 0x10464 thì RVA của nó sẽ là 0x464. Công thức tính toán địa chỉ này rất đơn giản:

Địa chỉ ảo (Virtual Address) - Địa chỉ cơ sở (Base Address) = RVA

0x10464 - 0x10000 = 0x00464

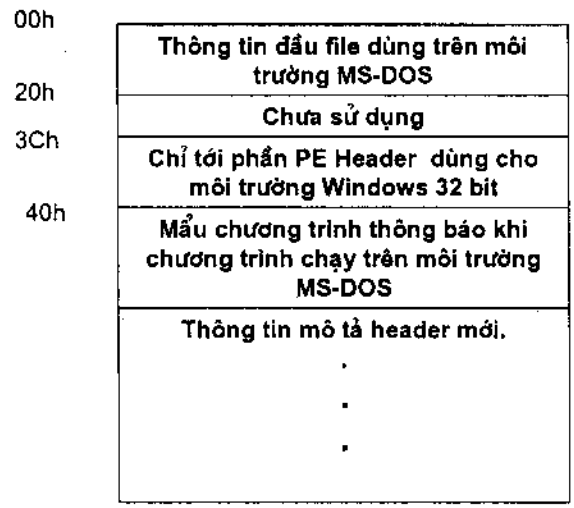
Địa chỉ cơ sở là địa chỉ bắt đầu của file EXE (hoặc của thư viện DLL) ánh xạ trong bộ nhớ. Trong Win32, gọi địa chỉ cơ sở của một module là HINSTANCE (instance handle) ở một mức nào đó có vẻ như không hợp lý bởi nó có nguồn gốc từ Win16. Mỗi một *instance* của ứng dụng trong Windows 16 bit có một mảng dữ liệu riêng biệt lập (gắn với nó là một handle toàn cục) để phân biệt với những *instance* khác của ứng dụng đó, do vậy xuất hiện cụm từ "instance handle". Các ứng dụng trong Win32 không cần phải được phân biệt giữa các instance bởi vì chúng không dùng chung vùng nhớ nào. Tuy nhiên, HINSTANCE được sử dụng để chỉ ra tính kế thừa từ Win16 của Win32. Đối với Win32, ta có thể dùng hàm *GetModuleHandle* cho bất cứ thư viện DLL nào mà chương trình sử dụng, do vậy có thể có được một con trỏ cho phép truy nhập đến các thành phần của module đó.

Trong file PE, một khái niệm cũng rất quan trọng là *Section*. Section có thể chứa hoặc là mã chương trình, hoặc là dữ liệu của chương trình. Không giống như các segment, section là một khối bộ nhớ liên tục không giới hạn về độ lớn. Một số section chứa mã hoặc dữ liệu mà chương trình trực tiếp sử dụng, một số khác được tạo ra do trình liên kết

(linker) chứa các thông tin cần thiết cho hệ điều hành. Trong một số tài liệu, section được đề cập đến như là các đối tượng.

Vùng tiêu đề PE không nằm ngay ở đầu file, khoảng 100 byte đầu tiên của một file PE chứa một mẩu chương trình MS-DOS nhỏ có nhiệm vụ in ra một thông tin ngắn như "*This program cannot be run in MS-DOS mode*" khi ta chạy chương trình Win32 trong môi trường không hỗ trợ Win32.

Hình 4.3 minh họa 4 phần đầu tiên của header file.



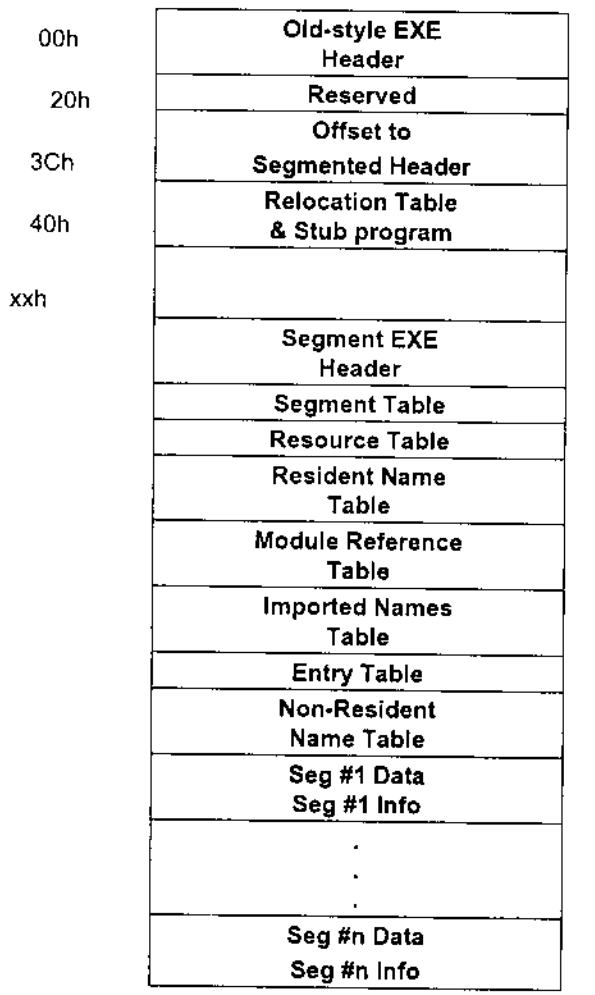
Hình 4.3. **Minh họa 4 phần đầu tiên của tiêu đề**

T rong phần tiêu đề MS-DOS, nếu word tại offset 18h có giá trị là 40h hoặc lớn hơn, thì giá trị của word ở offset 3Ch sẽ chỉ ra offset tới tiêu đề Windows. MS-DOS sẽ sử dụng mẩu chương trình để hiển thị thông báo khi thi hành chương trình trên môi trường MS-DOS. Các thông tin cụ thể về tiêu đề MS-DOS được trình bày ở phần tiêu đề của các file chạy trên môi trường MS-DOS.

Segment EXE Header bao gồm nhiều loại như PE, NE, LE, có nội dung miêu tả, kích thước, vị trí tải file lên bộ nhớ, số section trong file,... Tiếp theo là các section miêu tả về vị trí kích thước của section trên file và vị trí, kích thước của section khi tải lên bộ nhớ.

- ***Phần tiêu đề PE***

Phần header này chứa các thông tin về vị trí, kích thước của phần code và dữ liệu, hệ điều hành file cần để chạy,... Phần tiêu đề PE gồm có 3 phần: phần dấu hiệu nhận dạng (gồm 4 byte có giá trị "PE\0\0"), phần IMAGE_FILE HEADER và phần IMAGE_OPTIONAL_HEADER. Dưới đây là bảng liệt kê đầy đủ các trường trong phần tiêu đề PE, các vị trí được tính tương đối, bắt đầu từ byte đầu tiên của phần tiêu đề PE (không phải là byte đầu tiên tính từ đầu file). Tên các trường giữ nguyên theo định nghĩa của Microsoft để tiện tham khảo về sau.



Hình 4.4. Minh họa tiêu đề EXE

4 byte đầu của phần tiêu đề PE chứa chuỗi nhận dạng *"PE\0\0"*, nếu 4 byte này là *"NE\0\0"* thì có nghĩa là file chạy trong môi trường Windows 16 bit, nếu là *"LE\0\0"* chỉ ra đó là một VxD của Windows 3.x, còn *"LX\0\0"* dành cho file của OS/2 2.0. Tiếp theo chuỗi nhận dạng là phần IMAGE_FILE_HEADER, các trường trong phần này chứa những thông tin cơ bản nhất về file (đã được chỉ ra ở trên). Ngoài ra, còn một số lưu ý sau:

Offset	*Tên trường*	*Ý nghĩa*
00h - 03h	Signature	Dấu hiệu nhận dạng ("PE\0\0")
IMAGE_FILE_HEADER		
04h - 05h	Machine	Yêu cầu CPU cho file
06h - 07h	NumberOfSections	Số section của file
08h - 0Bh	TimeDateStamp	Thời điểm lúc file được tạo ra
0Ch - 0Fh	PointerToSymbolTable	Vị trí của bảng symbol COFF trên file, chỉ được dùng trong file OBJ và file PE với thông tin debug COFF. Thường không sử dụng
10h - 13h	NumberOfSymbols	Số symbol có trong bảng symbol COFF. Thường không sử dụng
14h - 15h	SizeOfOptionalHeader	Kích thước của phần Header mở rộng (Optional Header)
16h - 17h	Characteristics	Cờ chứa các thông tin về file
IMAGE_OPTIONAL_HEADER		
18h - 19h	Magic	Thường có giá trị 10Bh
1Ah	MajorLinkerVersion	
1Bh	MinorLinkerVersion	Version của trình liên kết

1Ch	SizeOfCode	Hầu hết các file chỉ có 1 code section, do đó trường này chứa kích thước của section.text
20h - 23h	SizeOfInitializedData	Tổng kích thước của các section (ngoại trừ code segment) chứa dữ liệu đã được khởi tạo khi chương trình bắt đầu
24h - 27h	SizeOfUninitializedData	Kích thước của các section mà trình nạp cho phép chiếm một khoảng trong bộ nhớ, tuy vậy chúng không chiếm 1 khoảng trong file trên đĩa. Các section này không cần phải có giá trị xác định khi chương trình khởi động. Thường các dữ liệu loại này nằm trong section.bss
28h - 2Bh	AddressOfEntryPoint	Địa chỉ (RVA) chứa câu lệnh đầu tiên được thực hiện
2Ch - 2Fh	BaseOfCode	Địa chỉ (RVA) nơi Code section bắt đầu
30h - 33h	BaseOfData	Địa chỉ (RVA) bắt đầu của Data section
34h - 37h	ImageBase	Địa chỉ (RVA) ánh xạ file trong bộ nhớ khi file được gọi (thường có giá trị là 0x400000)
38h - 3Bh	SectionAlignment	Khi đã ánh xạ file vào bộ nhớ, mỗi section của file được bảo đảm bắt đầu từ một địa chỉ là bội số của giá trị này (với mục đích phân trang, giá trị ngầm định là 0x1000)
3Ch - 3Fh	FileAlignment	Giá trị ngầm định là 0x200. Trong PE file, các section thường có kích thước là bội số của giá trị này
40h - 41h	MajorOperatingSystemVersion	
42h - 43h	MinorOperatingSystemVersion	Version tối thiểu của hệ điều hành để thực hiện file này
44h - 45h	MajorImageVersion	
46h - 47h	MinorImageVersion	Đây là trường do người sử dụng định nghĩa, có thể đặt giá trị cho trường này thông qua chỉ thị /VERSION của trình liên kết. Ví dụ: LINK /VERSION : 2.0 myobj.obj

48h - 49h	MajorSubsystemVersion	
4Ah - 4Bh	MinorSubsystemVersion	Version tối thiểu của hệ thống con chạy chương trình. Giá trị đặc trưng là 3.10 (có nghĩa là Windows NT 3.1)
4Ch - 4Fh	Reserved1	Dành riêng
50h - 53h	SizeOfImage	Tổng kích thước của file bắt đầu từ ImageBase cho đến hết section cuối cùng (kích thước của mỗi section được làm tròn dựa vào SectionAlignment, ví dụ nếu kích thước của section.data trong bộ nhớ là 230h thì kích thước của section trên file sẽ được làm tròn lên 400h nếu SectionAlignment = 200h)
54h - 57h	SizeOfHeaders	Kích thước của toàn bộ phần header và bảng các section
58h - 5Bh	CheckSum	CRC checksum của file
5Ch - 5Dh	Subsystem	
5Eh - 5Fh	DllCharacteristics	Cờ xác định trong trường hợp nào thì một hàm khởi tạo DLL (ví dụ như DllMain) được gọi
60h - 63h	SizeOfStackReserve	Giá trị ngầm định là 0x100000 (1MB). Số lượng bộ nhớ ảo dành cho ngăn xếp của luồng khởi tạo, không phải tất cả vùng nhớ này đều được chấp nhận
64h - 67h	SizeOfStackCommit	Giá trị ngầm định là 0x1000 (1 trang) đối với Microsoft Linker, còn đối với TLINK32 là 2 trang
68h - 6Bh	SizeOfHeapReserve	Số lượng bộ nhớ ảo dành cho vùng heap của tiến trình khởi tạo, có thể dùng hàm GetProcessHeap để tham khảo
6Ch - 6Fh	SizeOfHeapCommit	Giá trị ngầm định là 1 trang
70h - 73h	LoaderFlags	
74h - 77h	NumberOfRvaAndSizes	Số lượng các Entry trong mảng DataDirectory (= 10h)

- Đối với trường Mechine, ta có một số giá trị để xác định CPU:

Giá trị	*Yêu cầu CPU*
0x14d	Intel i860
0x14c	Intel I386 (same ID used for 486 and 586)
0x162	MIPS R3000
0x166	MIPS R4000
0x183	DEC Alpha AXP

- Đối với trường Characteristics, có một số giá trị quan trọng sau

Giá trị	*Ý nghĩa*
0x0001	Không có sự tái định vị trong file
0x0002	Là file chương trình chứ không phải là OBJ hay LIB
0x2000	Là thư viện liên kết động chứ không phải là file chương trình

Tiếp theo phần IMAGE_HEADER_FILE là phần tiêu đề PE mở rộng có cấu trúc IMAGE_OPTIONAL_HEADER, phần này cũng đã được mô tả ở trên. Trường cuối cùng của phần tiêu đề PE là NumberOfRvaAndSizes hiện nay luôn có giá trị ngầm định là 10h. Đây là số phần tử của mảng các phần tử có cấu trúc IMAGE_DATA_DIRECTORY. Khai báo của mảng này như sau:

IMAGE_DATA_DIRECTORY

DATADIRECTORY[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]

Mảng này nằm ngay sau phần tiêu đề PE mở rộng. Các phần tử của mảng chứa RVA và kích cỡ của các thành phần quan trọng trong file, chúng có độ lớn là 8 byte. Một số phần tử ở phía cuối của mảng hiện nay chưa được sử dụng. Phần tử đầu tiên của mảng luôn chứa địa chỉ và kích thước của bảng các hàm Export (nếu có). Phần tử thứ hai chứa địa

chỉ và kích thước của bảng các hàm Import... Mảng phần tử này cho phép trình nạp nhanh chóng tìm ra các section riêng biệt của file mà không cần phải duyệt qua từng section. Dưới đây là bảng liệt kê 16 phần tử của mảng này :

Offset	*Nội dung*
78h - 7Fh	Export table address and size
80h - 87h	Import table address and size
88h - 8Fh	Resource table address and size
90h - 97h	Exception table address and size
98h - 9Fh	Certificate table address and size
A0h - A7h	Base relocation table address and size
A8h - AFh	Debugging information starting address and size
B0h - B7h	Architecture-specific data address and size
B8h - BFh	Global pointer register relative virtual address
C0h - C7h	Thread local storage (TLS) table address and size
C8h - CFh	Load configuration table address and size
D0h - D7h	Bound Import table address and size
D8h - DFh	Import address table address and size
E0h - E7h	Delay import descriptor address and size
E8h - Efh	COM runtime descriptor address and size
F0h - F7h	Reserved

Tiếp sau là bảng các section chứa các section header, mỗi section header có 40 byte có cấu trúc IMAGE_SECTION_HEADER, gồm các trường (tên trường giữ nguyên theo định nghĩa của Microsoft để tiện tham khảo về sau):