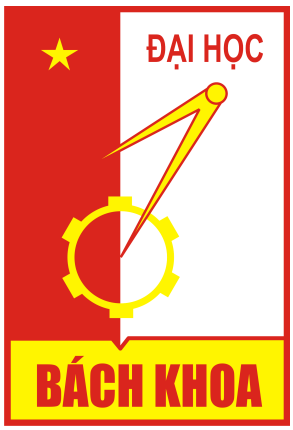


Research Project

NGUYÊN LÝ HỆ ĐIỀU HÀNH

ĐỀ TÀI

Quản lý bộ nhớ họ vi xử lý Intel

Cao Như Đạt - Trịnh Tùng Dương - Nguyễn Trung Hiếu - Nguyễn Huy Hoàng - Lê Quốc Hưng - Nguyễn Văn Thành
(Chương trình Tài năng Khoa học máy tính K64)

Mục lục

1 Tổng quan

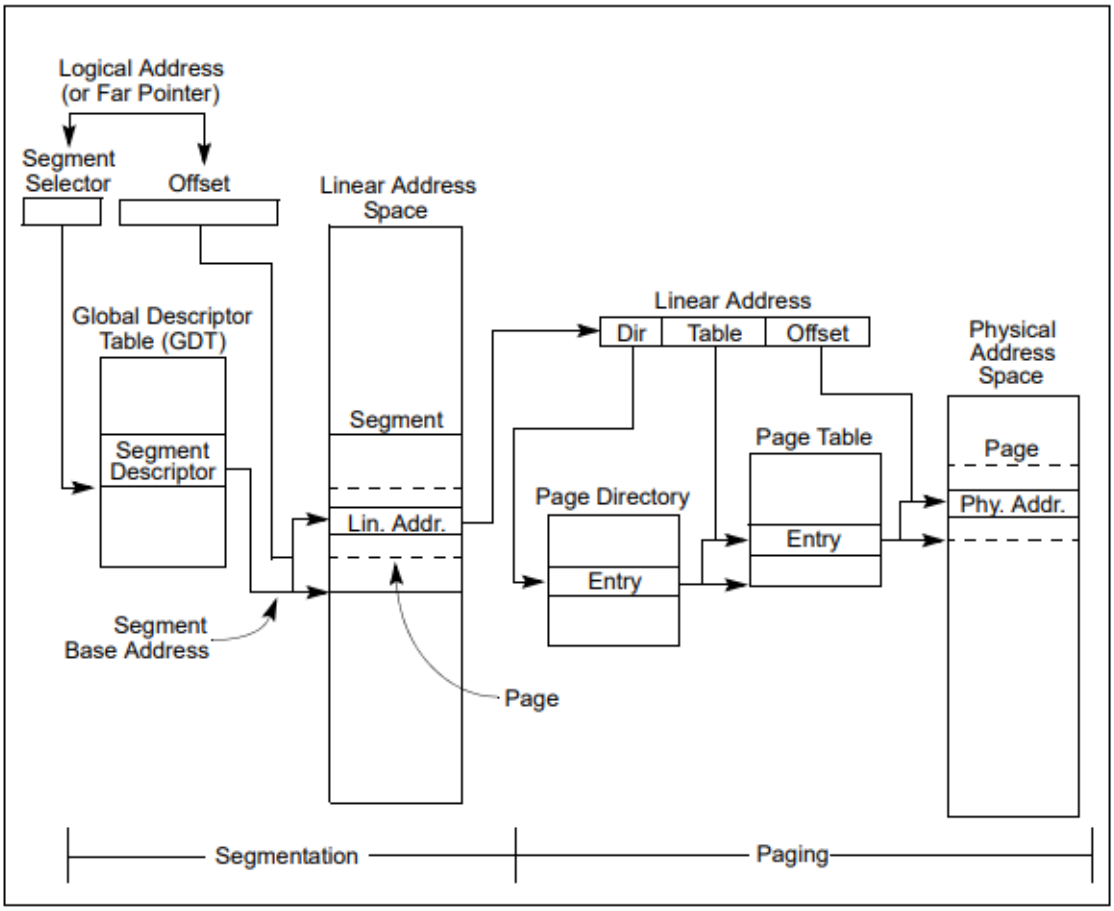
Bộ nhớ là tài nguyên quan trọng của hệ thống, dùng để lưu trữ dữ liệu lâu dài tạm thời (khi kết thúc phiên làm việc của máy tính thì dữ liệu không mất đi mất đi), nó được đặc trưng bởi kích thước và tốc độ truy nhập bao gồm: RAM, ROM, Cache, đĩa cứng, đĩa mềm, đĩa quang,... Các chương trình tồn tại trong các thiết bị lưu trữ ngoài, chúng phải tham gia hàng đợi đưa vào bộ nhớ trong và đặt trong một tiến trình để có thể thực hiện được.

Như vậy, một chương trình được thực hiện phải trải qua các giai đoạn bắt đầu từ khi được nạp vào bộ nhớ, thực hiện chương trình chính và kết thúc. Vấn đề đặt ra ở đây là chương trình có thể được nạp vào vị trí nào trong bộ nhớ? bất kỳ hay có một thứ tự định sẵn, hay khi thực hiện chương trình, các chuỗi địa chỉ bộ nhớ sinh ra sẽ được quản lý như thế nào? Đó chính là nhiệm vụ của việc quản lý bộ nhớ.

Quản lý bộ nhớ (memory management) là chức năng của hệ điều hành xử lý hoặc quản lý bộ nhớ chính và di chuyển các tiến trình qua lại giữa bộ nhớ chính và đĩa trong quá trình thực thi. Mục đích quan trọng của việc quản lý bộ nhớ là cung cấp những cách thức để cấp phát động các ô nhớ cho chương trình khi được yêu cầu và giải phóng các ô nhớ đó khi không cần dùng nữa. Đây là việc rất quan trọng đối với bất kỳ hệ thống máy tính cao cấp nào vì sẽ có nhiều công việc được tiến hành ở mọi thời điểm. Quản lý bộ nhớ theo dõi từng vị trí bộ nhớ, bất kể nó được cấp phát cho một số quy trình hay nó miễn phí. Nó kiểm tra lượng bộ nhớ được cấp cho các tiến trình. Nó quyết định tiến trình nào sẽ nhận được bộ nhớ vào thời điểm nào. Nó theo dõi bất khi nào một số bộ nhớ được giải phóng hoặc không được phân bổ và tương ứng nó sẽ cập nhật trạng thái. Các hệ điều hành đều mong muốn có nhiều tiến trình trong bộ nhớ chính. Hai chiến lược cơ bản nhất của quản lý bộ nhớ là phân trang (paging) và phân đoạn (segmentation), ngoài ra hệ điều hành cũng có thể kết hợp phân đoạn và phân trang để có một chiến lược quản lý bộ nhớ linh hoạt hơn.

Các chiến lược quản lý bộ nhớ của kiến trúc Intel được chia thành hai phần chính: phân đoạn và phân trang. Phân đoạn cung cấp một cơ chế cô lập mã, dữ liệu và mô-đun ngăn xếp riêng lẻ để nhiều chương trình (hoặc tác vụ) có thể chạy trên cùng một bộ xử lý mà không can thiệp vào nhau. Phân trang cung cấp một cơ chế triển khai hệ thống bộ nhớ ảo, phân trang theo yêu cầu thông thường, nơi các phần của môi trường thực thi của chương trình được ánh xạ vào bộ nhớ khi cần thiết. Phân trang cũng có thể được sử dụng để cung cấp sự cô lập giữa nhiều các nhiệm vụ. Khi hoạt động ở chế độ được bảo vệ, một số hình thức phân đoạn phải được sử dụng. Không có bit chế độ để vô hiệu hóa phân đoạn. Tuy nhiên, việc sử dụng phân trang là không bắt buộc. Hai cơ chế này (phân đoạn và phân trang) có thể được định cấu hình để hỗ trợ hệ

thống đơn chương trình (hoặc một tác vụ) đơn giản, hệ thống đa nhiệm hoặc hệ thống nhiều bộ xử lý đã sử dụng bộ nhớ dùng chung.



Hình 1: *Phân đoạn và phân trang*

Nhằm củng cố kiến thức về hệ điều hành, nắm vững các cấu trúc chương trình, các chiến lược quản lý bộ nhớ (chiến lược phân chương, chiến lược phân chương động, chiến lược phân đoạn, chiến lược phân trang) và đồng thời mở rộng những hiểu biết về các vấn đề liên quan, nhóm chúng em đã tìm hiểu về các chiến lược quản lý bộ nhớ của họ vi xử lý Intel. Sau đây nhóm chúng em xin trình bày phần tìm hiểu của mình.

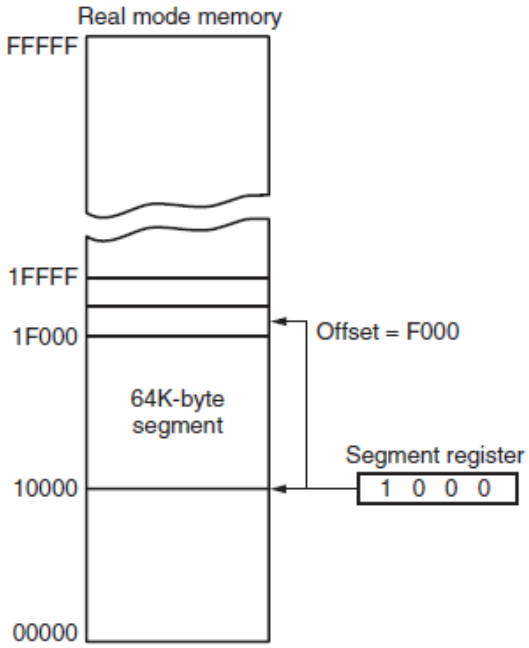
2 Quản lý bộ nhớ của vi xử lý họ Intel

2.1 Địa chỉ bộ nhớ chế độ thực

Intel 80286 trở lên hoạt động ở chế độ thực hoặc chế độ được bảo vệ. Chỉ 8086 và 8088 hoạt động độc quyền trong chế độ thực. Trong chế độ hoạt động 64-bit của Pentium 4 và Core2, không có hoạt động chế độ thực. Phần này nêu chi tiết hoạt động của bộ vi xử lý trong thực tế chế độ. Hoạt động ở chế độ thực cho phép bộ vi xử lý chỉ giải quyết 1M byte bộ nhớ đầu tiên không gian — ngay cả khi đó là bộ vi xử lý Pentium 4 hoặc Core2. Lưu ý rằng 1M byte bộ nhớ đầu tiên được gọi là bộ nhớ thực, bộ nhớ thông thường, hoặc hệ thống bộ nhớ DOS. Hệ điều hành DOS yêu cầu bộ vi xử lý hoạt động ở chế độ thực. Windows không sử dụng chế độ thực. Hoạt động ở chế độ thực cho phép phần mềm ứng dụng được viết cho 8086/8088, chỉ chứa 1 triệu byte bộ nhớ, hoạt động trong 80286 trở lên mà không cần thay đổi phần mềm. Hướng lên khả năng tương thích của phần mềm chịu trách nhiệm một phần cho sự thành công liên tục của bộ vi xử lý dòng Intel. Trong mọi trường hợp, mỗi bộ vi xử lý này bắt đầu hoạt động ở chế độ thực bằng mặc định bất cứ khi nào nguồn được cấp hoặc bộ vi xử lý được đặt lại. Lưu ý rằng nếu Pentium 4 hoặc Core2 hoạt động ở chế độ 64-bit, nó không thể thực thi các ứng dụng chế độ thực; do đó, các ứng dụng DOS sẽ không thực thi ở chế độ 64 bit trừ khi chương trình giả lập DOS được viết cho chế độ 64 bit.

Đoạn và độ dời

Sự kết hợp của địa chỉ đoạn và địa chỉ dời giúp truy cập vào một địa chỉ bộ nhớ trong chế độ thực. Tất cả các địa chỉ bộ nhớ chế độ thực phải bao gồm địa chỉ đoạn cộng với địa chỉ độ dời. Địa chỉ đoạn, nằm ở một trong các thanh ghi đoạn (CS,DS, ES, SS, hay FS và GS từ 80386), xác định địa chỉ đầu của mọi đoạn nhớ 64KB. Địa chỉ dời lựa chọn bất kì vị trí nào trong đoạn nhớ 64KB. Đoạn nhớ trong chế độ thực luôn có độ dài 64KB (tức là kích thước thực tế của đoạn từ 1-65536 bit, do các đoạn có thể chồng lên nhau và các byte nằm trong vùng chồng lên nhau đó có thể được sử dụng từ thanh ghi chọn – selector của cả 2 đoạn). Hình 2 cho thấy cách phương thức định địa chỉ đoạn cộng với độ dời lựa chọn một vị trí nhớ. Thanh ghi đoạn trong hình chứa 1000H (1000 trong hệ cơ số 16 - hệ hexa), nhưng lại xác định một địa chỉ bắt đầu tại 10000H. Trong chế độ thực, mỗi thanh ghi đoạn được ngầm định thêm vào bên phải của nó 0H. Điều này tạo nên một địa chỉ nhớ 20 bit để truy cập vào 1MB đầu tiên của bộ nhớ. Do cơ chế ngầm định này, đoạn trong chế độ thực chỉ có thể bắt đầu tại một ranh giới 16B trong bộ nhớ (địa chỉ chia hết cho 16). Ranh giới 16B này còn được gọi là paragraph. Khi biết địa chỉ đầu đoạn, do mỗi đoạn có độ dài 64KB nên chỉ cần cộng thêm với FFFFH để tìm địa chỉ cuối đoạn.



Hình 2: *Phương thức định địa chỉ chế độ thực*

Phần địa chỉ dời được cộng thêm với vị trí đầu của đoạn để xác định một vị trí bộ nhớ. Ví dụ: nếu địa chỉ thanh ghi là 1000H và địa chỉ dời là 2000H, VXL xác định địa chỉ bộ nhớ 1000H * 10H + 2000H = 12000H. Cặp địa chỉ đoạn và địa chỉ dời có thể được viết thành 1000:2000. Trong VXL 80286 (với những mạch ngoài đặc biệt) và 80386 cho đến Pentium 4, một bộ nhớ 64KB trừ đi 16B có thể được xác định/truy cập khi địa chỉ đoạn là FFFFH và trình điều khiển thiết bị(driver) HIMEM.SYS dành cho DOS được cài đặt trên hệ thống. Vùng bộ nhớ này(từ 0FFFF0H – 10FFEFH) được gọi là bộ nhớ cao. Khi một địa chỉ được tạo ra sử dụng địa chỉ đoạn FFFFH, chân địa chỉ (address pin) A20 được kích hoạt (nếu được hỗ trợ bởi các hệ thống cũ) khi độ dời được cộng vào.

Segment Register	*Starting Address*	*Ending Address*
2000H	20000H	2FFFFH
2001H	20010H	3000FH
2100H	21000H	30FFFH
AB00H	AB000H	BAFFFH
1234H	12340H	2233FH

Hình 3: *Ví dụ định địa chỉ chế độ thực*

Đoạn mặc định và thanh ghi độ dời

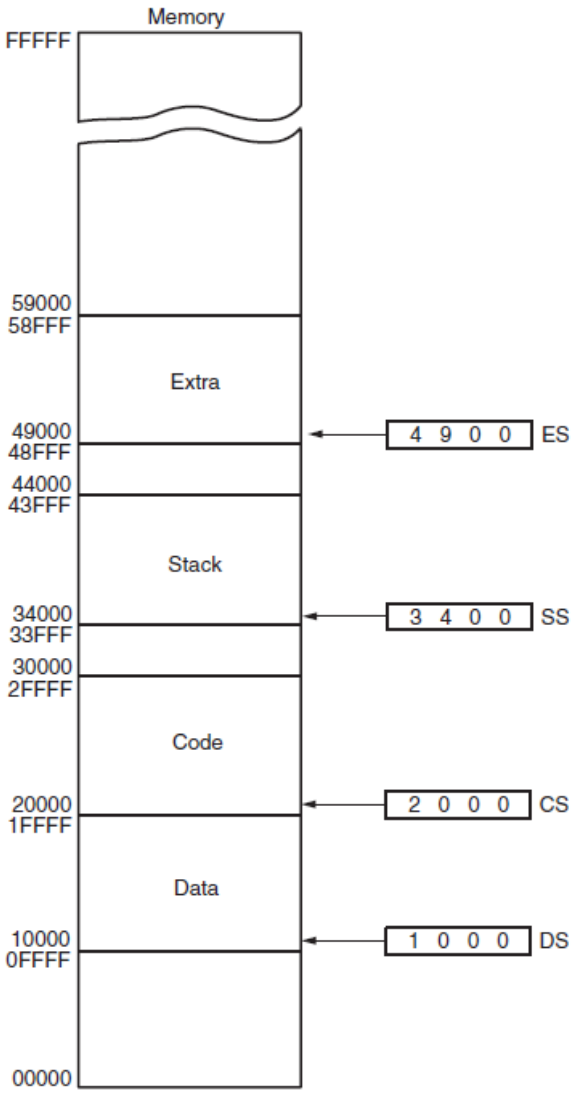
Vi xử lý có một tập các luật áp dụng cho đoạn chỉ xác định địa chỉ bộ nhớ. Những luật hay quy tắc này, áp dung trong cả chế độ thực và chế độ bảo vệ, định nghĩa tổ hợp thanh ghi đoạn và thanh ghi độ dời. Ví dụ: thanh ghi đoạn mã (CS – code segment) luôn được sử dụng cùng với thanh ghi con trỏ lệnh (IP – instruction pointer) để xác định địa chỉ lệnh tiếp theo trong chương trình được VXL thực hiện. Tổ hợp này là CS:IP hay CS:EIP, phụ thuộc vào chế độ vận hành của VXL.
Một tổ hợp mặc định khác là ngăn xếp (stack). Dữ liệu ngăn xếp được tham chiếu thông qua đoạn ngăn xếp tại địa chỉ bộ nhớ xác định bởi con trỏ ngăn xếp (SP/ESP – stack pointer) hoặc con trỏ gốc (BP/EBP – base pointer). Lưu ý rằng trong chế dộ thực, chỉ có 16 bit bên phải của thanh ghi địa chỉ mở rộng xác định vị trí trong một đoạn nhớ.
Các tổ hợp mặc định khác được thể hiện ở Hình 4 sử dụng cho việc định địa chỉ sử dụng VXL Intel với thanh ghi 16 bit. Hình 5 thể hiện tổ hợp mặc định đối với 80386 và thế hệ tiếp theo sử dụng thanh ghi 32 bit.

Segment	*Offset*	*Special Purpose*
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8- or 16-bit number	Data address
ES	DI for string instructions	String destination address

Hình 4: *Tổ hợp đoạn và độ dời 16 bit mặc định*

Segment	*Offset*	*Special Purpose*
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8- or 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

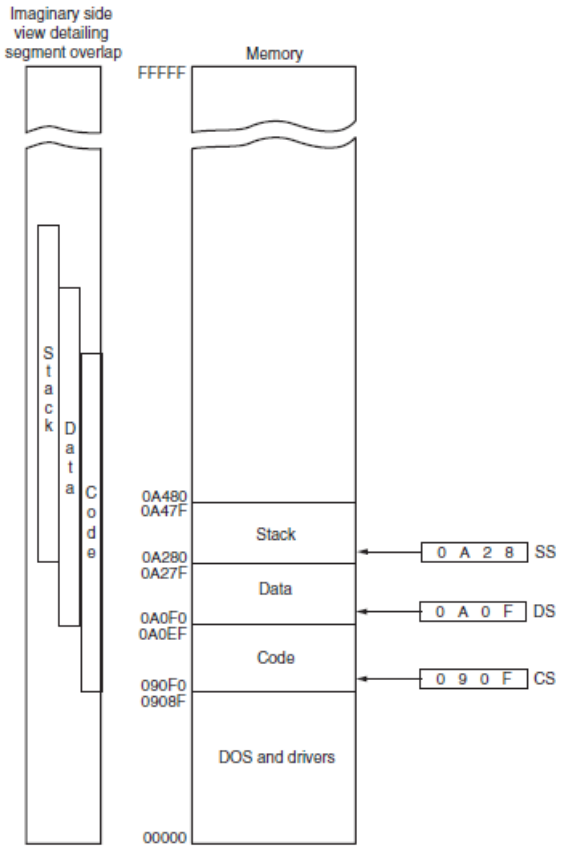
Hình 5: *Tổ hợp đoạn và độ dời 32 bit mặc định*



Hình 6: *4 đoạn nhớ trong hệ thống*

Các VXL 8086-80286 cho phép 4 đoạn nhớ và 80386-Core2 cho phép 6 đoạn nhớ. Hình 6 cho thấy một hệ thống gồm có 4 đoạn nhớ. Lưu ý rằng một đoạn nhớ có thể chồng lên đoạn khác nếu cả 64KB là không cần thiết cho 1 đoạn nhớ. Cũng lưu ý thêm rằng một chương trình có thể có nhiều hơn 4 hay 6 đoạn nhớ nhưng chỉ có thể truy cập 4 hoặc 6 đoạn nhớ cùng 1 lúc.

Giả thiết một chương trình ứng dụng yêu cầu 1000H bytes bộ nhớ cho đoạn mã, 190H bytes cho dữ liệu và 200H bytes cho ngăn xếp. Ứng dụng này không cần đoạn mở rộng (extra segment). Khi chương trình này được đưa vào bộ nhớ hệ thống bởi DOS, nó sẽ được nạp vào vùng chương trình tạm thời TPA (transient program area) tại vùng nhớ có sẵn đầu tiên phía trên vùng nhớ của driver và các chương trình TPA khác. Vùng này được chỉ định bởi một con trỏ tự do (free-pointer) được duy trì bởi DOS. Nạp chương trình được thực hiện tự động bởi bộ nạp chương trình (program loader) nằm trong DOS. Hình 7 cho thấy



Hình 7: *Cách một ứng dụng lưu trữ trong hệ thống*

cách một ứng dụng được lưu trữ trong bộ nhớ hệ thống. Các đoạn cho thấy sự chồng lên nhau vì dữ liệu trong mỗi đoạn không cần tới 64KB bộ nhớ. Phần bên cạnh trong hình cho chúng ta thấy chồng lên nhau này, đồng thời cũng cho thấy các đoạn có thể được di chuyển phía trên bất kì vùng nhớ nào bằng cách thay đổi địa chỉ bắt đầu. Các bộ nạp chương trình của DOS tính toán và gán các địa chỉ bắt đầu đoạn.

2.2 Giới thiệu về chế độ bảo vệ

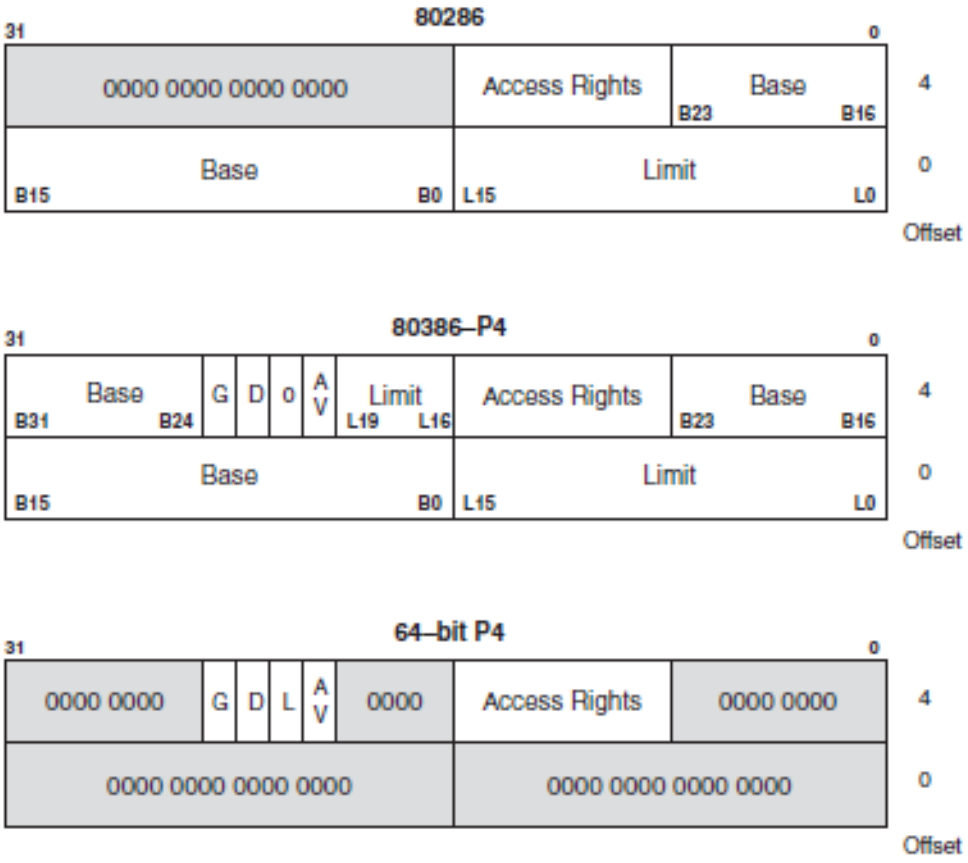
Xác định địa chỉ bộ nhớ trong chế độ bảo vệ (80286 và các thế hệ tiếp theo) cho phép truy cập vào dữ liệu và chương trình nằm ở phía trên vùng 1MB bộ nhớ đầu tiên cũng như chính vùng đó. Chế độ bảo vệ là nơi Windows vận hành. Định địa chỉ cho vùng mở rộng này của bộ nhớ yêu cầu một sự thay đổi trong phương thức định địa chỉ đoạn cộng độ dời sử dụng với xác định địa chỉ trong chế độ thực. Khi dữ liệu và chương trình được định địa chỉ ở trong vùng nhớ mở rộng, phần địa chỉ dời vẫn được sử dụng để truy cập thông tin nằm bên trong đoạn nhớ. Một sự khác biệt là địa chỉ đoạn, như đã được đề cập ở phần trước, không còn xuất hiện trong chế độ bảo vệ nữa. Thay vào đó, thanh ghi địa chỉ chứa một thanh ghi chọn (selector) giúp lựa chọn một descriptor từ bảng descriptor. Một bộ mô tả (descriptor) giúp mô tả vị trí, độ dài và quyền

truy cập của đoạn nhớ. Vì thanh ghi đoạn và địa chỉ dời vẫn truy cập vào bộ nhớ, lệnh của chế độ bảo vệ giống với chế độ thực. Thực tế, hầu hết các chương trình được viết để hoạt động trong chế độ thực sẽ hoạt động mà không thay đổi gì trong chế độ bảo vệ. Sự khác biệt giữa 2 chế độ nằm ở cách thanh ghi đoạn được thông dịch bởi VXL để truy cập vào đoạn nhớ. Một sự khác biệt nữa là, trong VXL 80386 và các thế hệ sau, địa chỉ dời có thể là 1 số 32 bit thay vì số 16 bit như trong chế độ bảo vệ. Địa chỉ dời 32 bit cho phép bộ VXL truy cập dữ liệu trong một đoạn có thể dài tối đa 4GB. Các chương trình được viết cho chế độ bảo vệ 32 bit có thể thực thi ở chế độ 64 bit của Pentium 4.

Thanh ghi chọn và bộ mô tả

Thanh ghi chọn, đặt ở trong thanh ghi đoạn, chọn 1 trong 8192 thanh ghi - chọn từ 1 trong 2 bảng các bộ mô tả. Bộ mô tả mô tả vị trí, độ dài và quyền truy cập vào đoạn nhớ. Một cách gián tiếp, thanh ghi đoạn vẫn lựa chọn một đoạn nhớ, nhưng không trực tiếp như trong chế độ thực. Ví dụ, như trong chế độ thực, nếu CS = 0008H, đoạn mã bắt đầu từ địa chỉ 00080H. Trong chế độ bảo vệ, số hiệu đoạn này có thể xác định bất kì vị trí nhớ nào trong toàn bộ hệ thống dành cho đoạn mã, như được giải thích dưới đây.

Có 2 bảng mô tả được sử dụng với thanh ghi đoạn: một bảng bao gồm các bộ mô tả toàn cục và bảng còn lại bao gồm các bộ mô tả cục bộ. Các bộ mô tả toàn cục (global descriptors) chứa các định nghĩa đoạn áp dụng cho tất cả các chương trình, trong khi đó bộ mô tả cục bộ (local descriptors) thường sẽ là độc nhất đối với mỗi ứng dụng. Có thể gọi bộ mô tả toàn cục là bộ mô tả hệ thống (system descriptor) và bộ mô tả cục bộ là thanh ghi chọn ứng dụng (application descriptor). Mỗi bảng bộ mô tả gồm 8192 bộ mô tả, do đó có tổng cộng 16384 bộ mô tả có sẵn cho một ứng dụng bất cứ lúc nào. Bởi vì bộ mô tả mô tả một đoạn bộ nhớ, điều này cho phép tối đa 16384 phân đoạn bộ nhớ được mô tả cho mỗi ứng dụng. Do độ dài đoạn nhớ có thể dài tối đa 4GB, một ứng dụng có thể có quyền truy cập vào 4GB x 16384 bộ nhớ hoặc 64TB
.
Hình 8 hiển thị định dạng của một bộ mô tả cho VXL 80286 đến Core2. Lưu ý rằng mỗi bộ mô tả có độ dài 8B, do đó, các bảng bộ mô tả toàn cục và cục bộ có độ dài tối đa 64KB. Các bộ mô tả cho 80286 và 80386->Core2 khác nhau đôi chút, nhưng bộ mô tả của 80286 có khả năng tương thích hướng lên (upward-compatible).
Phần địa chỉ cơ sở (base address) của bộ mô tả chỉ định vị trí bắt đầu của địa chỉ đoạn nhớ. Đối với VXL 80286, địa chỉ cơ sở là một địa chỉ 24 bit, do đó đoạn có thể bắt đầu từ bất cứ đâu trong vùng nhớ 16MB của nó (giới hạn ranh giới paragraph 16B được loại bỏ khi VXL vận hành trong chế độ bảo vệ nên đoạn có thể bắt đầu từ bất cứ địa chỉ nào). VXL 80386 và các thế hệ tiếp theo sử dụng một địa chỉ cơ sở 32 bit cho phép đoạn có thể bắt đầu từ bất cứ đâu trong vùng nhớ 4GB của nó. Lưu ý rằng địa chỉ cơ sở của bộ mô tả ở 80286



Hình 8: *Bộ mô tả từ 80286 đến Core2 (64 bit)*

được tương thích hướng lên với bộ mô tả ở 80386 cho đến Pentium 4 vì 16 bit cao nhất của nó là 0000H.

Phần giới hạn của đoạn (limit) bao gồm địa chỉ dời cuối cùng tìm thấy trong đoạn. Ví dụ, nếu đoạn bắt đầu tại vị trí F00000H và kết thúc tại vị trí F000FFH, địa chỉ cơ sở là F00000H và giới hạn là FFH. Đối với 80286, địa chỉ cơ sở là F00000H và giới hạn là 00FFH (16 bit). Đối với 80386 và các thế hệ tiếp theo, địa chỉ cơ sở là 00F00000H và giới hạn là 000FFH (20 bit). Do đó, 80286 có thể truy cập địa chỉ đoạn có độ dài từ 1B đến 64KB; còn 80386 và các thế hệ tiếp theo có thể truy cập đoạn nhớ có độ dài từ 1B đến 1MB hoặc từ 4KB đến 4GB.

Bit G hay bit chi tiết (granularity bit) chỉ có ở bộ mô tả từ 80386 trở đi. Nếu G = 0, phần giới hạn sẽ chỉ định một giới hạn đoạn từ 00000H đến FFFFFH. Nếu G = 1, giá trị của giới hạn được nhân với 4KB (thêm vào đằng sau FFFH). Giới hạn sẽ là từ 00000FFFFH đến FFFFFFFFFH, nếu G = 1. Điều nay cho phép đoạn có độ dài từ 4KB đến 4GB với bước nhảy 4KB. Lý do đoạn ở 80286 có độ dài 64KB là do địa chỉ dời luôn là 16 bit (do kiến trúc 16 bit bên trong của nó). 80386 và các thế hệ tiếp theo sử dụng kiến trúc 32 bit, cho phép một địa chỉ dời (trong hoạt động của chế độ bảo vệ) có độ lớn 32 bit và cho phép

độ dài đoạn 4GB. Các hệ điều hành hoạt động trên môi trường 16 bit hoặc 32 bit. Ví dụ, DOS sử dụng môi trường 16 bit, trong khi hầu hết các ứng dụng Windows sử dụng một môi trường 32 bit có tên WIN32.

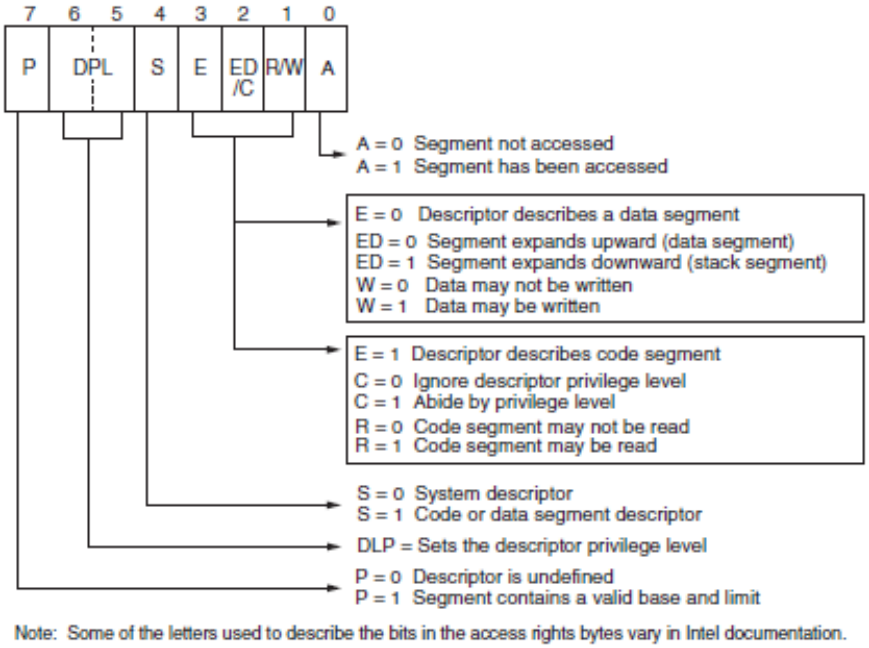
Trong bộ mô tả 64 bit, bit L lựa chọn địa chỉ 64 bit trong Pentium 4 hoặc Core2 khi L = 1 và lựa chọn địa chỉ chế độ tương thích 32 bit khi L = 0. Khi vận hành trong chế độ bảo vệ 64 bit, thanh ghi đoạn mã vẫn được sử dụng để lựa chọn 1 phần đoạn mã từ trong bộ nhớ. Lưu ý rằng bộ mô tả 64 bit không có phần giới hạn và phần địa chỉ cơ sở của đoạn, dù không được đặt trong bộ mô tả, là 00 0000 0000H. Điều này có nghĩa là mọi đoạn mã đều bắt đầu tại địa chỉ 0 với hoạt động 64 bit. Không có kiểm tra giới hạn đối với đoạn mã 64 bit.

Bit AV hay bit có sẵn, trong bộ mô tả của 80386 và các thế hệ sau, được sử dụng bởi một số hệ điều hành để chỉ định rằng đoạn đang có sẵn (AV = 1) hoặc không có sẵn (AV = 0).

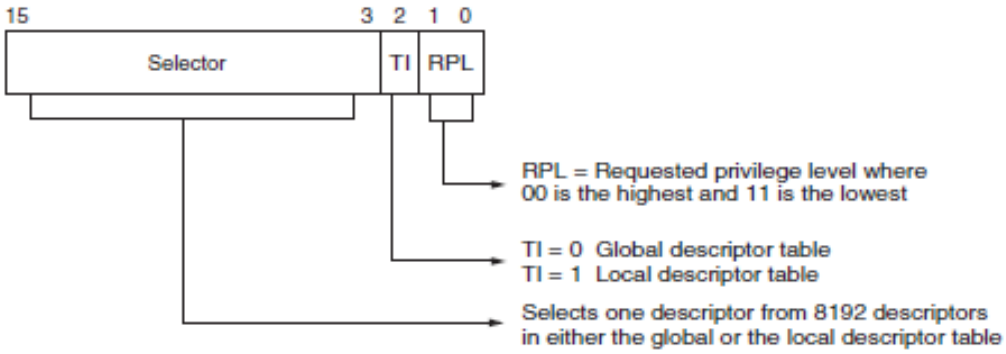
Bit D thể hiện cách các lệnh của 80386 đến Core2 truy cập vào thanh ghi và dữ liệu nhớ trong chế độ bảo vệ hoặc chế độ thực. Nếu D = 0, các lệnh có kích thước 16 bit, tương thích với VXL 8086-80286. Điều này có nghĩa là lệnh sử dụng địa chỉ dời 16 bit và thanh ghi 16 bit theo mặc định. Chế độ này thường được gọi là chế độ lệnh 16 bit hay chế độ DOS. Nếu D = 1, các lệnh có kích thước 32 bit. Lưu ý rằng kích thước thanh ghi mặc định và địa chỉ dời mặc định được ghi đè trong cả chế độ lệnh 16 bit và 32 bit. Cả hệ điều hành MSDOS và PCDOS yêu cầu lệnh luôn được sử dụng trong chế độ lệnh 16 bit. Windows 3.1 và các ứng dụng được viết cho nó, cũng yêu cầu chế độ lệnh 16 bit được sử dụng. Lưu ý chế độ lệnh chỉ có thể được truy cập trong các hệ thống chế độ bảo vệ như Windows Vista.

Byte quyền truy cập (access rights byte) trong hình 9 điều khiển truy cập tới đoạn của chế độ bảo vệ. Byte này mô tả cách đoạn hoạt động trong hệ thống. Byte quyền truy cập cho phép toàn quyền điều khiển trên đoạn. Nếu đoạn là một đoạn dữ liệu, hướng mở rộng được chỉ định. Nếu đoạn mở rộng ra ngoài giới hạn, chương trình hệ điều hành của VXL bị ngắt, biểu thị lỗi bảo vệ chung. Người dùng có thể chỉ định một đoạn dữ liệu là có thể viết hoặc được bảo vệ không cho viết (write-protected). Đoạn mã cũng được điều khiển tương tự như vậy và có thể bị cấm đọc để bảo vệ phần bit, chỉ có một đoạn mã và không có loại bộ mô tả đoạn nào khác. Một chương trình 64 bit mô hình phẳng (flat model) bao gồm đoạn dữ liệu và ngăn xếp của nó trong đoạn mã.

Bộ mô tả được lựa chọn từ bảng bộ mô tả bởi thanh ghi đoạn. Hình 10 cho thấy cách thanh ghi đoạn hoạt động trong chế độ bảo vệ.
Thanh ghi đoạn bao gồm 1 trường lựa chọn (selector field) 13 bit, 1 bit chọn bảng, và 1 trường mức độ ưu tiên được yêu cầu (requested privilege level field).



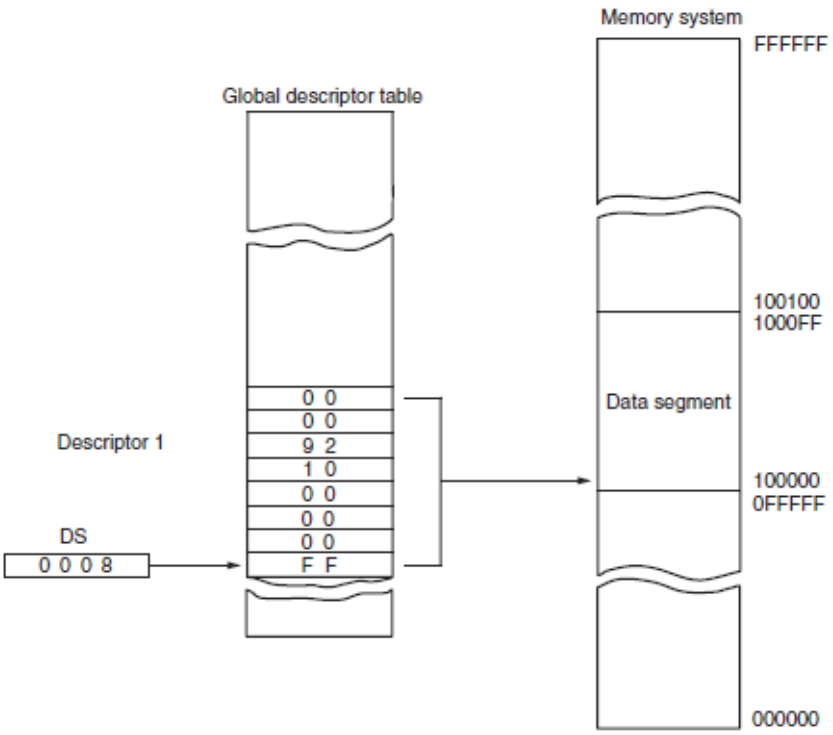
Hình 9: *Byte quyền truy cập của bộ mô tả từ 80286 đến Core2*



Hình 10: *Thanh ghi đoạn trong chế độ bảo vệ từ VXL 80286 đến Core2*

Bộ/thanh ghi chọn 13 bit chọn 1 trong 8192 bộ mô tả từ bảng bộ mô tả.Bit TI lựa chọn bảng bộ mô tả toàn cục (TI = 0) hoặc bảng bộ mô tả cục bộ (TI = 1). Mức độ ưu tiên được yêu cầu (requested privilege level – RPL) yêu cầu mức độ ưu tiên trong quyền truy cập vào đoạn nhớ. Mức độ ưu tiên cao nhất là 00 và thấp nhất là 11. Nếu mức độ ưu tiên được yêu cầu khớp với hoặc cao hơn mức độ ưu tiên được đặt bởi byte quyền truy cập, quyền truy cập được cấp. Quyền truy cập được sử dụng ở các môi trường nhiều người sử dụng. Windows sử dụng mức 00 (vòng 0) cho nhân (kernel) và trình điều khiển thiết bị (driver), mức 11 (vòng 3) cho các ứng dụng, và không sử dụng mức 01 hay 10. Nếu mức ưu tiên bị vi phạm, hệ thống thông thường chỉ định lỗi vi phạm mức ưu tiên.

Hình 11 thể hiện cách thanh ghi đoạn, chứa một thanh ghi chọn, chọn một đoạn từ bộ nhớ.



Hình 11: *Sử dụng thanh ghi DS để chọn một mô tả từ bảng mô tả chung. Ở đây, thanh ghi DS truy cập các vị trí bộ nhớ 00100000H – 001000FFH dưới dạng một đoạn dữ liệu.*

Phần mở đầu của bảng bộ mô tả toàn cục lựa chọn một đoạn trong bộ nhớ hệ thống. Trong ảnh minh họa, thanh ghi DS chứa giá trị 0008H, truy cập vào bộ mô tả số 1 trong bảng sử dụng mức ưu tiên được yêu cầu 00. Bộ mô tả số 1 chứa một bộ mô tả định nghĩa địa chỉ cơ sở là 00100000H với giới hạn đoạn là 000FFH. Có nghĩa là giá trị 0008H nạp vào thanh ghi DS khiến cho VXL sử dụng vị trí nhớ 00100000H – 001000FFH cho đoạn dữ liệu. Lưu ý bộ mô tả không hay null (null descriptor) chứa toàn các số 0 và có thể không được sử dụng để truy cập bộ nhớ.

2.3 Phân trang bộ nhớ

Cơ chế phân trang bộ nhớ nằm trong 80386 trở lên cho phép bất kỳ bộ nhớ vật lý nào vị trí được chỉ định cho bất kỳ địa chỉ tuyến tính nào. Địa chỉ tuyến tính được xác định là địa chỉ được tạo bởi một chương trình. Địa chỉ vật lý là vị trí bộ nhớ thực được một chương trình truy cập. Với đơn vị phân trang bộ nhớ, địa chỉ tuyến tính được dịch ẩn thành bất kỳ địa chỉ vật lý nào, cho phép một ứng dụng được viết để hoạt động tại một địa chỉ cụ thể được di dời thông qua cơ chế phân trang. Nó cũng cho phép đặt bộ nhớ vào những vùng không có bộ nhớ. Một ví dụ là các khối bộ nhớ trên được cung cấp bởi EMM386.EXE

trong hệ thống DOS. Chương trình EMM386.EXE chỉ định lại bộ nhớ mở rộng, trong các khối 4K, cho hệ thống bộ nhớ giữa BIOS video và ROMS BIOS hệ thống cho các khối bộ nhớ trên. Nếu không có cơ chế phân trang, việc sử dụng vùng bộ nhớ này là không thể.

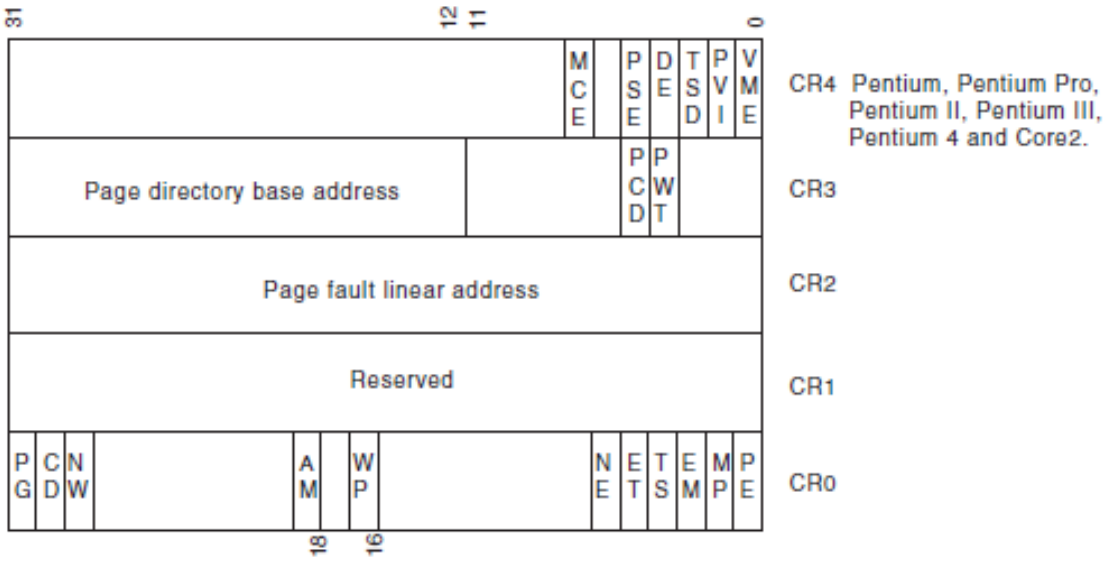
Trong Windows, mỗi ứng dụng được phép một không gian địa chỉ tuyến tính 2G từ vị trí 00000000H – 7FFFFFFFH mặc dù có thể không có đủ bộ nhớ hoặc bộ nhớ khả dụng tại các địa chỉ này. Thông qua phân trang vào ổ đĩa cứng và phân trang vào bộ nhớ thông qua đơn vị phân trang bộ nhớ, bất kỳ ứng dụng Windows nào cũng có thể được thực thi.

Thanh ghi phân trang

Đơn vị phân trang được điều khiển bởi nội dung của các thanh ghi điều khiển của bộ vi xử lý. Hình 12 cho thấy nội dung của thanh ghi điều khiển CR0 đến CR4. Lưu ý rằng các thanh ghi này có sẵn cho 80386 thông qua bộ vi xử lý Core2. Bắt đầu với Pentium, một điều khiển bổ sung thanh ghi các phần mở rộng điều khiển CR4 được gắn nhãn đối với kiến trúc cơ bản được cung cấp trong Pentium hoặc bộ vi xử lý mới hơn. Một trong những tính năng này là trang 2M hoặc 4M byte được kích hoạt bằng cách điều khiển CR4.

Các thanh ghi quan trọng đối với đơn vị phân trang là CR0 và CR3. Vị trí bit (PG) ngoài cùng bên trái của CR0 chọn phân trang khi được đặt ở mức logic 1. Nếu bit PG bị xóa (0), tuyến tính địa chỉ do chương trình tạo ra sẽ trở thành địa chỉ vật lý được sử dụng để truy cập bộ nhớ. Nếu Bit PG được đặt (1), địa chỉ tuyến tính được chuyển thành địa chỉ vật lý thông qua cơ chế phân trang. Cơ chế phân trang hoạt động ở cả chế độ thực và chế độ được bảo vệ.

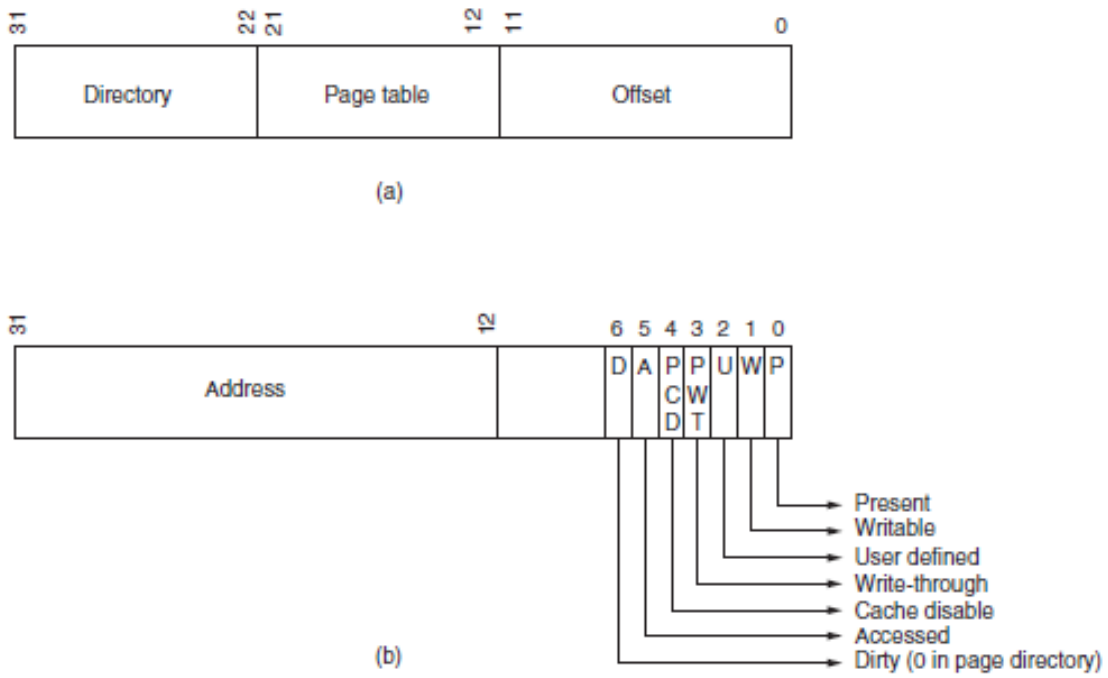
CR3 chứa cơ sở hoặc địa chỉ gốc của thư mục trang và các bit PCD và PWT. PCD và các bit PWT điều khiển hoạt động của các chân PCD và PWT trên bộ vi xử lý. Nếu PCD được thiết lập (1), chân PCD trở thành chân logic trong các chu kỳ bus không được phân trang. Điều này cho phép bên ngoài phần cứng để điều khiển bộ nhớ đệm cấp 2. (Lưu ý rằng bộ nhớ đệm cấp 2 là bộ nhớ trong [trên các phiên bản hiện đại của Pentium] bộ nhớ tốc độ cao có chức năng như một bộ đệm giữa bộ vi xử lý và hệ thống bộ nhớ DRAM chính.) Bit PWT cũng xuất hiện trên PWT pin trong các chu kỳ bus không được phân trang để điều khiển bộ đệm ghi qua trong hệ thống. Trang địa chỉ cơ sở thư mục định vị thư mục cho đơn vị dịch trang. Lưu ý rằng địa chỉ này định vị thư mục trang ở bất kỳ ranh giới 4K nào trong hệ thống bộ nhớ vì nó được nối vào bên trong với 000H. Thư mục trang chứa 1024 mục nhập thư mục, mỗi mục 4 byte. Mỗi trang mục nhập thư mục địa chỉ một bảng trang chứa 1024 mục nhập.
Địa chỉ tuyến tính, vì nó được tạo ra bởi phần mềm, được chia thành ba phần được sử dụng để truy cập mục nhập thư mục trang, mục nhập bảng trang và địa chỉ bù trang bộ nhớ. Hình 13 cho thấy địa chỉ tuyến tính và cấu trúc của



Hình 12: *Cấu trúc thanh ghi điều khiển của vi xử lý*

nó để phân trang. Chú ý cách 10 bit ngoài cùng bên trái giải quyết một mục trong thư mục trang. Đối với địa chỉ tuyến tính 00000000H – 003FFFFFH, trang đầu tiên thư mục được truy cập. Mỗi mục nhập thư mục trang đại diện hoặc đăng lại một phần 4M của bộ nhớ hệ thống. Nội dung của thư mục trang chọn một bảng trang được lập chỉ mục bởi 10 bit tiếp theo của địa chỉ tuyến tính (vị trí bit 12–21). Điều này có nghĩa là địa chỉ 00000000H – 00000FFFH sẽ chọn mục nhập thư mục trang là 0 và mục nhập bảng trang là 0. Lưu ý rằng đây là dải địa chỉ 4K byte. Sự bù đắp một phần của địa chỉ tuyến tính (vị trí bit 0-11) tiếp theo chọn một byte trong trang bộ nhớ 4K-byte. Trong Hình 2–12, nếu mục nhập bảng trang 0 chứa địa chỉ 00100000H, thì địa chỉ thực là 00100000H-00100FFFH cho địa chỉ tuyến tính 00000000H – 00000FFFH. Điều này có nghĩa là khi chương trình truy cập một vị trí trong khoảng từ 00000000H đến 00000FFFH, bộ vi xử lý vật lý địa chỉ vị trí 00100000H – 00100FFFH.

Vì hành động sao chép phần bộ nhớ 4K-byte yêu cầu quyền truy cập vào thư mục trang và một bảng trang, cả hai đều nằm trong bộ nhớ, Intel đã kết hợp một loại bộ nhớ cache được gọi là TLB (bộ đệm nhìn sang một bên bản dịch). Trong bộ vi xử lý 80486, bộ nhớ đệm nắm giữ 32 địa chỉ dịch trang gần đây nhất. Điều này có nghĩa là bản dịch bảng 32 trang cuối cùng được lưu trữ trong TLB, vì vậy nếu cùng một vùng bộ nhớ được truy cập, địa chỉ đó đã hiện diện trong TLB và không bắt buộc phải truy cập vào thư mục trang và bảng trang. Tốc độ này Thực hiện chương trình. Nếu bản dịch không có trong TLB, thư mục trang và bảng trang phải được truy cập, yêu cầu thêm thời gian thực hiện. Bộ vi xử lý Pentium – Pentium 4 chứa các TLB riêng biệt cho từng lệnh và bộ nhớ đệm dữ liệu của chúng.

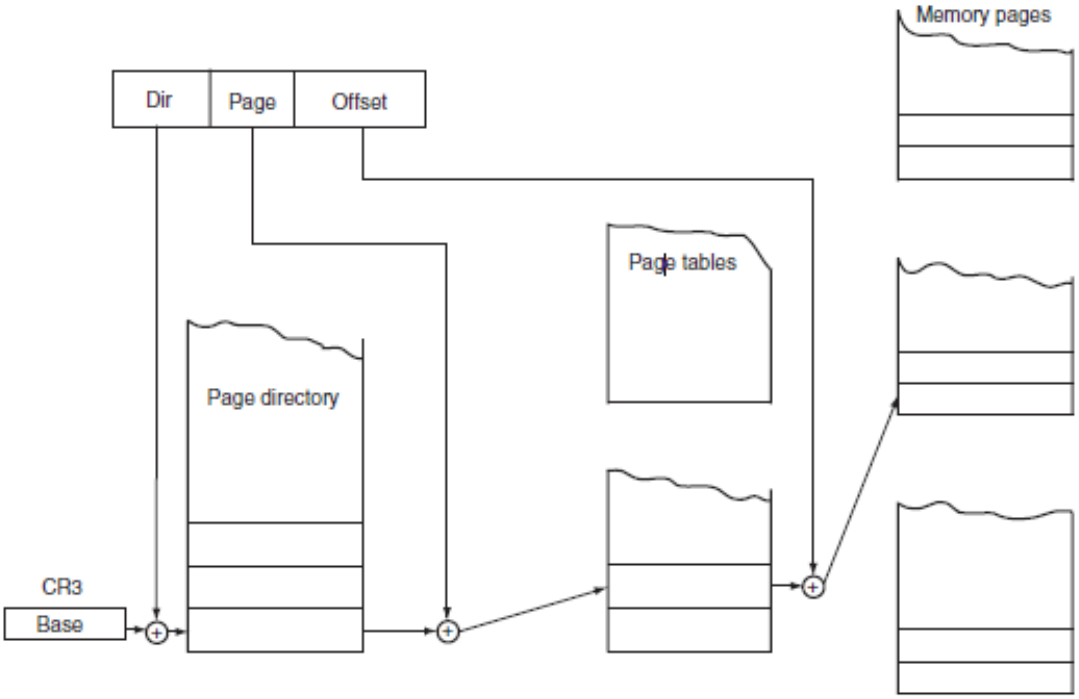


Hình 13: *Định dạng cho địa chỉ tuyến tính (a) và một thư mục trang hoặc mục nhập bảng trang (b).*

Thư mục trang và bảng trang

Hình 14 cho thấy thư mục trang, một vài bảng trang và một số trang bộ nhớ. Chỉ có một thư mục trang trong hệ thống. Thư mục trang chứa 1024 địa chỉ từ kép định vị tối đa 1024 bảng trang. Thư mục trang và mỗi bảng trang có độ dài 4K byte. Nếu toàn bộ byte bộ nhớ 4G được phân trang, hệ thống phải phân bổ 4K byte bộ nhớ cho thư mục trang và 4K nhân với 1024 hoặc 4M byte cho 1024 bảng trang. Điều này thể hiện sự đầu tư đáng kể vào tài nguyên bộ nhớ.
Hệ thống DOS và EMM386.EXE sử dụng bảng trang để xác định lại vùng bộ nhớ giữa các vị trí C8000H – EFFFFH như các khối bộ nhớ trên. Điều này được thực hiện bằng cách tái tạo bộ nhớ mở rộng để lấp đầy phần này của hệ thống bộ nhớ thông thường để cho phép DOS truy cập vào bộ nhớ bổ sung. Giả sử rằng chương trình EMM386.EXE cho phép truy cập vào 16 triệu byte bộ nhớ mở rộng và thông thường thông qua phân trang và các vị trí C8000H – EFFFFH phải được định vị lại đến các vị trí 110000–138000H, với tất cả các vùng khác của bộ nhớ được phân trang như vị trí thông thường. Một sơ đồ như vậy được mô tả trong Hình 15

.
Ở đây, thư mục trang chứa bốn mục nhập. Nhớ lại rằng mỗi mục trong thư mục trang tương ứng với 4M byte bộ nhớ vật lý. Hệ thống cũng chứa bốn bảng trang với 1024 mục nhập mỗi bảng. Nhớ lại rằng mỗi mục nhập trong bảng trang chiếm 4K byte bộ nhớ vật lý. Lược đồ này yêu cầu tổng cộng 16K bộ nhớ



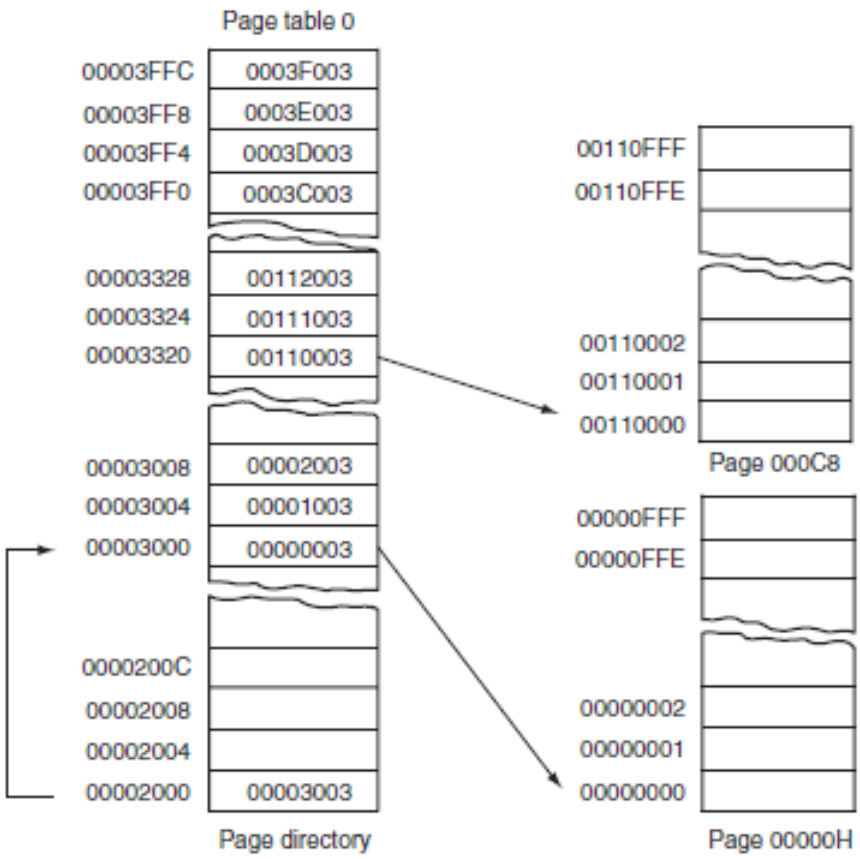
Hình 14: *Cơ chế phân trang trong 80386 thông qua bộ vi xử lý Core2*

cho bốn bảng trang và 16 byte bộ nhớ cho thư mục trang.

Như với DOS, chương trình Windows cũng lưu trữ lại hệ thống bộ nhớ. Hiện tại, Windows phiên bản 3.11 chỉ hỗ trợ phân trang cho 16 triệu byte bộ nhớ vì dung lượng bộ nhớ cần thiết để lưu trữ các bảng trang. Các phiên bản Windows mới hơn đóng gói toàn bộ hệ thống bộ nhớ. Trên bộ vi xử lý Pentium – Core2, các trang có thể có độ dài 4K, 2M hoặc 4M byte. Trong các biến thể 2M và 4M, chỉ có thư mục trang và trang bộ nhớ, nhưng không có bảng trang.

2.4 Kết hợp phân đoạn và phân trang

Cơ chế phân đoạn và phân trang được cung cấp trong kiến trúc IA-32 hỗ trợ nhiều cách tiếp cận khác nhau để quản lý bộ nhớ. Khi phân đoạn và phân trang được kết hợp, các phân đoạn có thể được ánh xạ tới các trang theo một số cách. Ví dụ: để triển khai môi trường địa chỉ phẳng (không phân đoạn), tất cả các mô-đun mã, dữ liệu và ngăn xếp có thể được ánh xạ tới một hoặc nhiều phân đoạn lớn (lên đến 4 GByte) có cùng dải địa chỉ tuyến tính. Ở đây, các phân đoạn về cơ bản là vô hình đối với các ứng dụng và hệ điều hành hoặc điều hành. Nếu phân trang được sử dụng, cơ chế phân trang có thể ánh xạ một không gian địa chỉ tuyến tính duy nhất (chứa trong một phân đoạn duy nhất) vào bộ nhớ ảo. Hoặc, mỗi chương trình (hoặc nhiệm vụ) có thể có không gian địa chỉ tuyến tính lớn của riêng nó (chứa trong phân đoạn riêng của nó), được ánh xạ vào bộ nhớ ảo thông qua thư mục trang và tập hợp các bảng trang của chính nó.

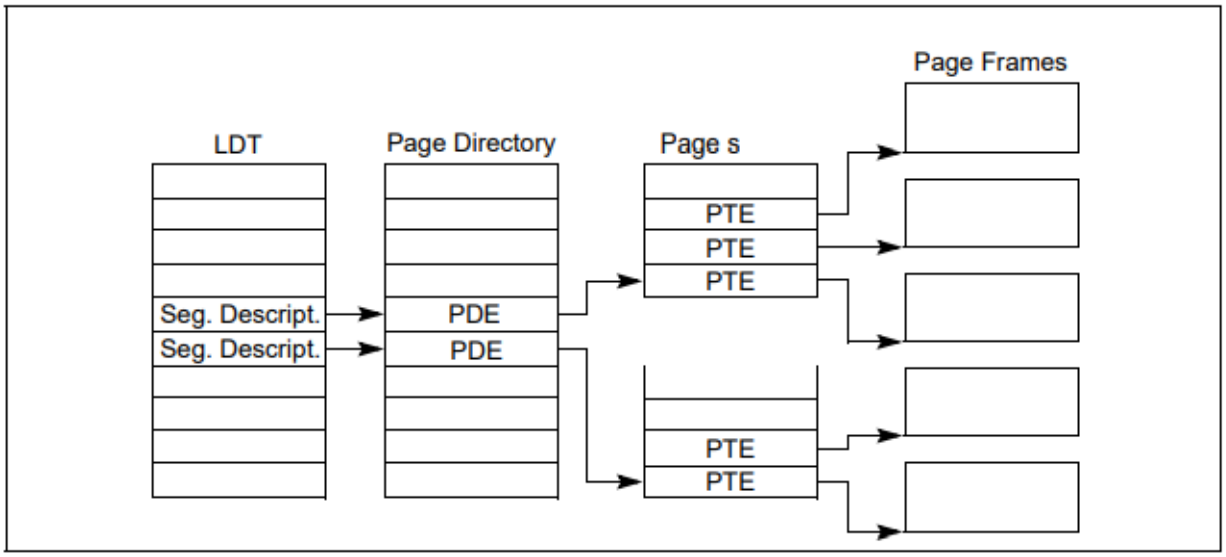


Hình 15: *Trang thư mục, bảng trang 0 và hai trang bộ nhớ.*

Các phân đoạn có thể nhỏ hơn kích thước của một trang. Nếu một trong những phân đoạn này được đặt trong một trang không được chia sẻ với một phân đoạn khác, thì bộ nhớ bổ sung sẽ bị lãng phí. Ví dụ: cấu trúc dữ liệu nhỏ, chẳng hạn như semaphore 1 byte, ccupies 4K byte nếu chính nó được đặt trong một trang. Nếu nhiều semaphores được sử dụng, sẽ hiệu quả hơn nếu đóng gói chúng thành một trang duy nhất.

Kiến trúc IA-32 không thực thi sự tương ứng giữa nhiều trang và phân đoạn. Một trang có thể chứa phần cuối của một phân đoạn và phần đầu của một phân đoạn khác. Tương tự như vậy, một phân đoạn có thể chứa phần cuối của một trang và phần đầu của trang khác. Phần mềm quản lý bộ nhớ có thể đơn giản và hiệu quả hơn nếu nó thực thi một số liên kết giữa ranh giới trang và phân đoạn. Ví dụ: nếu một phân đoạn có thể vừa với một trang được đặt trong hai trang, có thể có gấp đôi chi phí phân trang để hỗ trợ quyền truy cập vào phân đoạn đó.

Một cách tiếp cận để kết hợp phân trang và phân đoạn giúp đơn giản hóa phần



Hình 16: *Quy ước quản lý bộ nhớ chỉ định một bảng trang đến từng phân đoạn*

mềm quản lý bộ nhớ là cung cấp cho mỗi phân đoạn một bảng trang riêng của nó, như thể hiện trong Hình 16. Quy ước này cung cấp cho phân đoạn một mục nhập duy nhất trong thư mục trang cung cấp thông tin kiểm soát truy cập để phân trang toàn bộ phân đoạn.

3 Quản lý bộ nhớ trong Window

Hầu hết các chương trình yêu cầu một số hình thức quản lý bộ nhớ động. Nhu cầu này phát sinh bất cứ khi nào có nhu cầu tạo cấu trúc dữ liệu có kích thước hoặc số lượng không được biết vào thời gian xây dựng chương trình. Cây tìm kiếm, bảng biểu tượng và danh sách liên kết là các ví dụ phổ biến về cấu trúc dữ liệu động trong đó chương trình tạo các phiên bản mới trong thời gian chạy.

Windows cung cấp các cơ chế linh hoạt để quản lý bộ nhớ động của chương trình. Windows cũng cung cấp các tệp được ánh xạ bộ nhớ để liên kết địa chỉ của tiến trình với không gian trực tiếp với một tệp, cho phép HĐH quản lý tất cả chuyển động dữ liệu giữa tệp và bộ nhớ để lập trình viên không bao giờ cần phải xử lý ReadFile, WriteFile,SetFilePointer, hoặc các chức năng vào ra với tệp khác. Với các tệp được ánh xạ bộ nhớ, chương trình có thể duy trì cấu trúc dữ liệu động một cách thuận tiện trong các tệp cố định và các thuật toán dựa trên bộ nhớ có thể xử lý tệp dữ liệu. Hơn nữa, ánh xạ bộ nhớ có thể tăng tốc độ xử lý tệp một cách đáng kể, và nó cung cấp cơ chế chia sẻ bộ nhớ giữa các tiến trình.

Thư viện liên kết động (DLL) là một trường hợp đặc biệt cần thiết của ánh xạ tệp và bộ nhớ được chia sẻ trong đó các tệp (chủ yếu là các tệp mã chỉ đọc) được ánh xạ vào không gian địa chỉ tiến trình để thực thi. Phần này mô tả quản lý bộ nhớ Windows và ánh xạ tệp các chức năng, minh họa lợi thế sử dụng và hiệu suất của chúng với một số và mô tả cả DLL được liên kết ngầm định và rõ ràng(implicity and explicity linked DLLs).

3.1 Kĩ thuật quản lý bộ nhớ Windows

Win32 là một API dành cho họ hệ điều hành Windows 32-bit. “32-bitness” tự thể hiện trong bộ nhớ địa chỉ và con trỏ (LPCTSTR, LPDWORD, v.v.) là các đối tượng 4 byte (32-bit). API Win64 cung cấp không gian địa chỉ ảo lớn hơn nhiều với 8 byte, 64 bit con trỏ và là một sự phát triển tự nhiên của Win32. Tuy nhiên, hãy cẩn thận để đảm bảo rằng các ứng dụng của bạn có thể được nhắm mục tiêu ở cả hai nền tảng; tất cả các ví dụ đã được thử nghiệm trên cả hệ thống 64-bit và 32-bit và các tệp thực thi 32-bit và 64-bit được có sẵn trong tệp Ví dụ. Với các chương trình ví dụ, có những nhận xét về những thay đổi được yêu cầu để hỗ trợ Win64.

Do đó, mọi tiến trình Windows đều có không gian địa chỉ ảo riêng của nó 4GB hoặc 16EB. Win32 tạo ra ít nhất một nửa trong số này (2–3GB; 3GB phải được bật lúc khởi động) có sẵn cho một quy trình. Các phần còn lại của không gian địa chỉ ảo được phân bổ cho dữ liệu và mã được chia sẻ, mã hệ thống, trình điều khiển, v.v. Các chi tiết về phân bổ bộ nhớ này, mặc dù rất thú vị, nhưng không quan trọng ở đây. Từ quan điểm của lập trình viên, hệ điều hành cung

cấp một lượng lớn không gian địa chỉ cho mã, dữ liệu và các tài nguyên khác. Phần tiếp theo tập trung vào khai thác quản lý bộ nhớ Windows mà không cần quan tâm đến hệ điều hành thực hiện. Tuy nhiên, sau đây là một tổng quan ngắn.

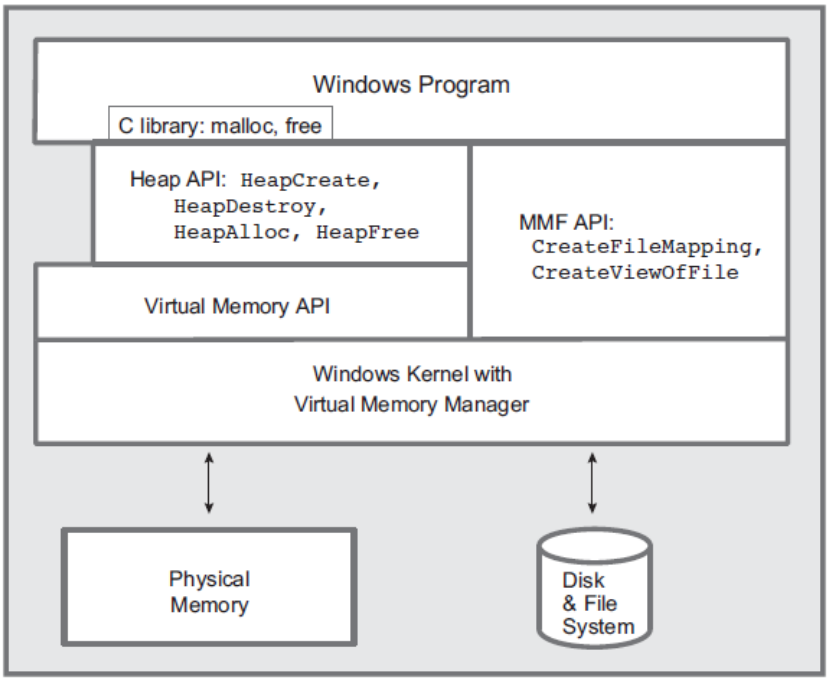
Hệ điều hành quản lý tất cả các chi tiết của ánh xạ bộ nhớ ảo với bộ nhớ vật lý và cơ chế hoán đổi trang, phân trang theo yêu cầu, và những thứ tương tự. Chủ đề này được thảo luận kỹ lưỡng trong các văn bản hệ điều hành. Đây là một bản tóm tắt ngắn gọn:

• Máy tính có dung lượng bộ nhớ vật lý tương đối nhỏ; 1GB là tối thiểu thiết thực cho Windows XP 32 bit và bộ nhớ vật lý lớn hơn nhiều là những điển hình.

• Mọi tiến trình và có thể có một số tiến trình của người dùng và hệ thống đều có riêng không gian địa chỉ ảo, có thể lớn hơn nhiều so với bộ nhớ vật lý có sẵn. Ví dụ: không gian địa chỉ ảo của quy trình 4GB lớn hơn gấp 2 lần so với bộ nhớ vật lý 2GB và có thể có nhiều tiến trình như vậy chạy đồng thời.

• Hệ điều hành ánh xạ địa chỉ ảo với địa chỉ thực.

• Hầu hết các trang ảo sẽ không nằm trong bộ nhớ vật lý, vì vậy hệ điều hành sẽ phản hồi lại trang lỗi (tham chiếu đến các trang không có trong bộ nhớ) và tải dữ liệu từ đĩa từ tệp hoán đổi hệ thống hoặc từ tệp bình thường. Lỗi trang có tác động đáng kể đến hiệu suất và các chương trình nên được thiết kế để giảm thiểu lỗi.



Hình 17: *Kĩ thuật quản lý bộ nhớ Windows*

Hình 17 cho thấy lớp API quản lý bộ nhớ Windows trên Trình quản lý bộ nhớ ảo. API Windows bộ nhớ ảo (VirtualAlloc, VirtualFree , VirtualLock, VirtualUnlock, v.v.) giao tiếp với toàn bộ các trang. API Heap của Windows quản lý bộ nhớ theo các đơn vị do người dùng xác định.

3.2 Vùng nhớ HEAPS

Windows duy trì các nhóm bộ nhớ trong heaps. Một quy trình có thể chứa một số heaps và bạn cấp phát bộ nhớ từ các heap này. Một heaps thường là đủ, nhưng có những lý do chính đáng, được giải thích bên dưới, cho nhiều heaps. Nếu một heap duy nhất là đủ, chỉ cần sử dụn hàm quản lý bộ nhớ thư viện C (malloc, free, calloc, realloc). Heaps là các đối tượng Windows; do đó, chúng có các thẻ. Tuy nhiên, heaps không phải là đối tượng nhân. Cần xử lý heap khi bạn đang phân bổ bộ nhớ. Mỗi tiến trình có heap mặc định của riêng nó và hàm tiếp theo nhận được thẻ của nó.

```
HANDLE GetProcessHeap (VOID)
```

Return: The handle for the process's heap; `NULL` on failure.

NULL là giá trị trả về để chỉ ra thất bại thay vì INVALID_HANDLE_VALUE, được trả lại bởi CreateFile. Một chương trình cũng có thể tạo ra các heaps riêng biệt. Đôi khi nó là thuận tiện để có riêng heaps để phân bổ các cấu trúc dữ liệu riêng biệt. Lợi ích của việc tách biệt heaps bao gồm những điều sau đây:

• Công bằng: Nếu các luồng chỉ phân bổ bộ nhớ từ một heap duy nhất được gán cho luồng, khi đó không luồng đơn lẻ nào có thể lấy được nhiều bộ nhớ hơn mức được cấp cho heaps. Đặc biệt, lỗi rò rỉ bộ nhớ do chương trình bỏ qua việc giải phóng các phần tử dữ liệu không còn cần thiết, sẽ chỉ ảnh hưởng đến một luồng của quy trình.
• Hiệu suất đa luồng: Bằng cách cung cấp cho mỗi luồng riêng heaps của nó, sự tranh chấp giữa các chủ đề được giảm bớt, điều này có thể cải thiện đáng kể hiệu suất.
• Hiệu quả phân bổ: Phân bổ các phần tử dữ liệu có kích thước cố định trong một heap nhỏ có thể hiệu quả hơn việc phân bổ các phần tử có nhiều kích thước khác nhau trong một heap lớn. Sự phân mảnh cũng được giảm bớt. Hơn nữa, cung cấp cho mỗi chủ đề một heap duy nhất để lưu trữ chỉ được sử dụng trong chuỗi đơn giản hóa việc đồng bộ hóa, dẫn đến hiệu quả bổ sung.
• Hiệu quả thu hồi: Toàn bộ heap và tất cả các cấu trúc dữ liệu chứa có thể

được giải phóng bằng một lệnh gọi hàm duy nhất. Lời gọi này cũng sẽ miễn phí bất kể cấp phát bộ nhớ bị rò rỉ trong heap.
• Vị trí của hiệu quả tham chiếu: Duy trì cấu trúc dữ liệu nhỏ heap đảm bảo rằng các phần tử sẽ được giới hạn ở một số lượng tương đối nhỏ của các trang, có khả năng làm giảm các lỗi trang do các phần tử cấu trúc dữ liệu xử lý.

Giá trị của những ưu điểm này khác nhau tùy thuộc vào ứng dụng, và nhiều người lập trình chỉ sử dụng process heap và thư viện C. Tuy nhiên, một sự lựa chọn như vậy ngăn chương trình khai thác khả năng tạo ngoại lệ của các hàm quản lý bộ nhớ của Windows (được mô tả cùng với các hàm).

3.3 Tệp ánh xạ bộ nhớ

Bộ nhớ động trong heap phải được cấp phát vật lý trong một tệp thay trang. Hệ điều hành quản lý bộ nhớ kiểm soát chuyển động của trang giữa bộ nhớ vật lý và tệp thay trang và cũng ánh xạ không gian địa chỉ ảo của quy trình với tệp thay trang. Khi quá trình kết thúc, không gian vật lý trong tệp được thu hồi.

Chức năng tệp ánh xạ bộ nhớ của Windows cũng có thể ánh xạ bộ nhớ ảo không gian trực tiếp đến các tệp bình thường. Điều này có một số lợi thế:

• Không cần thực hiện I / O tệp trực tiếp (đọc và ghi).
• Các cấu trúc dữ liệu được tạo trong bộ nhớ sẽ được lưu trong tệp để sử dụng sau này bởi cùng hoặc các chương trình khác. Hãy cẩn thận về việc sử dụng con trỏ.
• Các thuật toán trong bộ nhớ thuận tiện và hiệu quả (sắp xếp, cây tìm kiếm, chuỗi xử lý, v.v.) có thể xử lý dữ liệu tệp mặc dù tệp có thể nhiều lớn hơn bộ nhớ vật lý có sẵn. Hiệu suất vẫn sẽ bị ảnh hưởng bởi hành vi phân trang nếu tệp lớn.
• Hiệu suất xử lý tệp thường nhanh hơn nhiều so với việc sử dụng và các chức năng truy cập tệp.
• Không cần quản lý bộ đệm và dữ liệu tệp mà chúng chứa. Hệ điều hành thực hiện công việc khó khăn này và thực hiện nó một cách hiệu quả với mức độ tin cậy cao.
• Nhiều tiến trình có thể chia sẻ bộ nhớ bằng cách ánh xạ ảo của chúng địa chỉ không gian cho cùng một tệp hoặc tệp thay trang (bộ nhớ liên xử lý chia sẻ là lý do chính để ánh xạ tới tệp thay trang).
• Không cần tiêu tốn dung lượng tệp thay trang.

Bản thân hệ điều hành sử dụng ánh xạ bộ nhớ để triển khai các tệp DLL và tải và thực thi các tệp thực thi (.exe). Khi đọc hoặc viết một tệp được ánh xạ, bạn nên sử dụng SEH để bắt bất kỳ trường hợp ngoại lệ nào. Các chương trình

tệp ví dụ tất cả đều làm được điều này, nhưng SEH bị bỏ qua khỏi danh sách chương trình cho ngắn gọn.

3.4 Thư viện liên kết động

Bây giờ chúng ta đã thấy rằng quản lý bộ nhớ và ánh xạ tệp là quan trọng và các kỹ thuật hữu ích trong nhiều loại chương trình. Bản thân hệ điều hành cũng sử dụng bộ nhớ quản lý và DLLs là cách sử dụng quan trọng và dễ thấy nhất vì ứng dụng Windows sử dụng rộng rãi các tệp DLLs. DLLs cũng rất cần thiết cho các công nghệ cấp cao hơn, chẳng hạn như COM, và nhiều thành phần phần mềm được cung cấp dưới dạng DLLs. Bước đầu tiên là xem xét các phương pháp xây dựng thư viện khác nhau của các chức năng thường dùng.

Thư viện liên kết tĩnh và động

Cách trực tiếp nhất để xây dựng một chương trình là thu thập mã nguồn của tất cả các hàm, biên dịch chúng và liên kết mọi thứ thành một hình ảnh thực thi duy nhất. Các hàm phổ biến, chẳng hạn như ReportError, có thể được đưa vào thư viện để đơn giản hóa xây dựng quy trình. Kỹ thuật này đã được sử dụng với tất cả các chương trình mẫu được trình bày cho đến nay, mặc dù chỉ có một số hàm, hầu hết chúng để báo lỗi.

Mô hình nguyên khối, một hình ảnh này rất đơn giản, nhưng nó có một số nhược điểm:
• Hình ảnh thực thi có thể lớn, tiêu tốn dung lượng đĩa và bộ nhớ vật lý tại thời điểm chạy và cần thêm nỗ lực để quản lý và cung cấp cho người dùng.
• Mỗi bản cập nhật chương trình yêu cầu xây dựng lại chương trình hoàn chỉnh ngay cả khi các thay đổi nhỏ hoặc được địa phương hóa.
• Mỗi chương trình trong máy tính sử dụng các hàm sẽ có một bản sao của các hàm, có thể là các phiên bản khác nhau, trong hình ảnh thực thi của nó. Sự sắp xếp này tăng mức sử dụng dung lượng ổ đĩa và có lẽ quan trọng hơn là sử dụng bộ nhớ khi một số chương trình như vậy đang chạy đồng thời.
• Có thể yêu cầu các phiên bản riêng biệt của chương trình, sử dụng các kỹ thuật khác nhau để đạt được hiệu suất tốt nhất trong các môi trường khác nhau.

DLLs giải quyết những vấn đề này và các vấn đề khác khá gọn gàng:
• Các hàm thư viện không được liên kết tại thời điểm xây dựng. Đúng hơn, chúng được liên kết tại chương trình thời gian tải (liên kết ngầm định) hoặc tại thời điểm chạy (liên kết rõ ràng). Kết quả là, hình ảnh chương trình có thể nhỏ hơn nhiều vì nó không bao gồm các hàm của thư viện.
• Các tệp DLL có thể được sử dụng để tạo các thư viện dùng chung. Nhiều chương trình chia sẻ một thư viện ở dạng DLLs và chỉ một bản sao duy nhất được tải vào bộ nhớ. Tất cả các chương trình ánh xạ mã DLLs tới không gian địa chỉ quy trình của chúng, mặc dù mỗi tiến trình có một bản sao riêng biệt

của các biến toàn cầu của DLLs. Ví dụ, hàm ReportError đã được sử dụng bởi hầu hết các chương trình ví dụ; một DLL duy nhất việc thực hiện có thể được chia sẻ bởi tất cả các chương trình.
• Các phiên bản mới hoặc triển khai thay thế có thể được hỗ trợ đơn giản bởi cung cấp phiên bản mới của DLLs và tất cả các chương trình sử dụng thư viện có thể sử dụng phiên bản mới mà không cần sửa đổi.
• Thư viện sẽ chạy trong các quá trình giống như chương trình đang gọi.

DLLs, đôi khi ở dạng hạn chế, được sử dụng trong hầu hết mọi hệ điều hành. Ví dụ, UNIX sử dụng thuật ngữ "thư viện được chia sẻ" cho cùng một khái niệm. Windows sử dụng DLLs để triển khai các giao diện hệ điều hành, trong số những thứ khác. Toàn bộ API Windows là được hỗ trợ bởi một DLLs gọi hạt nhân Windows cho các dịch vụ bổ sung. Nhiều tiến trình Windows có thể chia sẻ mã DLLs, nhưng mã, khi được gọi, chạy như một phần của quá trình gọi và luồng. Do đó, thư viện sẽ có thể sử dụng các tài nguyên của quá trình gọi, chẳng hạn như các trình xử lý tệp và sẽ sử dụng lệnh gọi ngăn xếp của chuỗi. Do đó, các tệp DLL nên được viết là an toàn cho luồng. Một DLL cũng có thể xuất các biến cũng như các điểm nhập hàm.

Liên kết ngầm định (Implicit Linking)

Liên kết ngầm định, trong đó hệ điều hành tải DLLs cùng lúc với tệp thực thi sử dụng nó. Máy khách thực thi gọi các chức năng đã xuất của DLLs theo cách giống như khi các chức năng được liên kết tĩnh và chứa trong tệp thực thi. Liên kết ngầm đôi khi được gọi là liên kết động tải tĩnh hoặc thời gian tải.

Liên kết rõ ràng (Explicit Linking)

Liên kết rõ ràng, trong đó hệ điều hành tải DLLs theo yêu cầu trong thời gian chạy. Một tệp thực thi sử dụng DLLs theo liên kết rõ ràng phải tải và dỡ DLLs một cách rõ ràng. Nó cũng phải thiết lập một con trỏ chức năng để truy cập từng chức năng mà nó sử dụng từ DLLs. Không giống như các lệnh gọi đến các hàm trong thư viện được liên kết tĩnh hoặc một DLLs được liên kết ngầm, chương trình thực thi của ứng dụng khách phải gọi các hàm được xuất trong một DLLs được liên kết rõ ràng thông qua các con trỏ hàm. Liên kết rõ ràng đôi khi được gọi là tải động hoặc liên kết động trong thời gian chạy.

4 Tài liệu tham khảo

1. Bài giảng Nguyên lý hệ điều hành - Phạm Đăng Hải
2. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1
3. Barry B. Brey (2008),The Intel Microprocessors, Eighth Edition.
4. Johnson M. Hart, Windows System Programming (Fourth Edition).
5. Microsoft C++, C, and Assembler documentation.
6, Các tài liệu khác trên Internet.