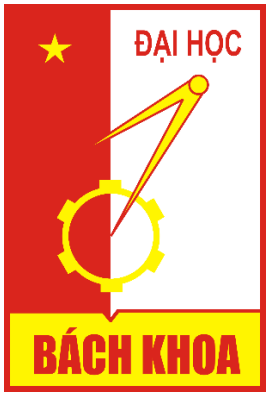
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI



MÔN HỌC: HỆ ĐIỀU HÀNH

TÌM HIỂU CÁCH QUẢN LÝ BỘ NHỚ TRONG HỆ ĐIỀU HÀNH LINUX VÀ VI XỬ LÝ INTEL 80386

SINH VIÊN: HỒ ANH TIẾN

MSSV: 20164861

LỚP: KSTN-CNTT-K61

Hà Nội, ngày 20 tháng 04 năm 2018

MỤC LỤC

1. Linux 3
1.1. Tổng quan về hệ điều hành Linux 3
1.2. Tổng quan về bộ nhớ trong Linux 3
1.3. Cơ chế quản lý bộ nhớ của LINUX 4
1.3.1. Một mô hình tóm tắt về bộ nhớ ảo 5
1.3.2. Vùng trung gian (Swapping) 6
1.3.3. Bộ nhớ ảo dùng chung (Shared virtual memory) 7
1.3.4. Caches 8
1.3.5. Các bảng quản lý trang trong LINUX 8
1.3.6. Vấn đề ánh xạ bộ nhớ 11
2. Vi xử lý Intel 80386 14
2.1. Tổng quan về Intel 80386 14
2.2. Quản lý bộ nhớ trên vi xử lí Intel 80386 15
2.2.1. Quản lý bộ nhớ Real mode 15
2.2.2. Quản lý bộ nhớ Protected mode 16
2.2.3. Cấu trúc record quản lý segment của Intel 17
2.2.4. Qui trình đổi địa chỉ ảo sang địa chỉ thật 17
2.2.5. Quản lý bộ nhớ 368 enchanced mode 18
TÀI LIỆU THAM KHẢO 20

1. Linux

Linux là 1 hệ điều hành mã nguồn mở rất phổ biến trên toàn thế giới hiện nay. Nhiều năm qua, **Linux** đã thực sự tạo ra một cuộc cách mạng trong lĩnh vực máy tính. Sự phát triển và những gì mà **Linux** mang lại thật đáng kinh ngạc: một hệ điều hành đa nhiệm, đa người dùng. **Linux** có thể chạy trên nhiều bộ vi xử lý khác nhau như: **Intel** , **Motorola** , **MC68K** , **Dec Alpha**…

Ngoài ra Linux còn tương tác tốt với các hệ điều hành của: **Apple** , **Microsoft** và **Novell**. Vì những ưu điểm của nó mà ngành công nghệ thông tin Việt Nam chọn **Linux** làm hệ điều hành nền cho các chương trình ứng dụng chủ đạo về kinh tế và quốc phòng. Với mã nguồn mở, việc sử dụng **Linux** an toàn hơn với các ứng dụng **Windows**. **Linux** đem đến cho chúng ta lợi ích về kinh tế với rất nhiều phần mềm miễn phí.

Giống như các hệ điều hành khác, Linux phải có các cơ chế và phương pháp khai thác và sử dụng tài nguyên máy hiệu quả, đặc biệt là tài nguyên bộ nhớ. Trong đồ án này, em sẽ trình bày một phần rất quan trọng trong hệ điều hành **Linux** cũng như các hệ điều hành khác là: cơ chế quản lý bộ nhớ trong của **Linux.**

1.1. Tổng quan về hệ điều hành Linux

Linux là một hệ điều hành họ UNIX miễn phí được sử dụng rộng rãi hiện nay. Được viết vào năm 1991 bởi Linus Toward, hệ điều hành Linux đã thu được nhiều thành công. Là một hệ điều hành đa nhiệm, đa người dùng, Linux có thể chạy trên nhiều nền phần cứng khác nhau. Với tính năng ổn định và mềm dẻo, Linux đang dần được sử dụng nhiều trên các máy chủ cũng như các máy trạm trong các mạng máy tính. Linux còn cho phép dễ dàng thực hiện việc tích hợp nó và các hệ điều hành khác trong một mạng máy tính như Windows, Novell, Apple ... Ngoài ra, với tính năng mã nguồn mở, hệ điều hành này còn cho phép khả năng tùy biến cao, thích hợp cho các nhu cầu sử dụng cụ thể.

1.2. Tổng quan về bộ nhớ trong Linux

Trong hệ thống máy tính, bộ nhớ là một tài nguyên quan trọng. Cho dù có bao nhiêu bộ nhớ đi chăng nữa thì vẫn không đáp ứng đủ nhu cầu của người sử dụng. Các máy tính cá nhân hiện nay đã được trang bị dung lượng bộ nhớ rất lớn. Thậm chí các máy chủ server có thể có đến hàng gigabyte bộ nhớ. Thế nhưng nhu cầu bộ nhớ vẫn không được thỏa mãn. Có rất nhiều chiến lược quản lí bộ nhớ được nghiên cứu và áp dụng, trong đó chiến lược sử dụng bộ nhớ ảo là hiệu quả nhất.Giống như các hệ điều hành khác, Linux sử dụng cơ chế bộ nhớ ảo để quản lí tài nguyên bộ nhớ trong máy tính.

Linux có cách tiếp cận và quản lý bộ nhớ rất rõ ràng. Các ứng dụng trên Linux không bao giờ được phép truy cập trực tiếp vào địa chỉ vật lý của bộ nhớ. Linux cung cấp cho các chương trình chạy dưới HĐH - còn gọi là tiến trình - một mô hình đánh địa chỉ phẳng không phân đoạn segment:offset như DOS.

Mỗi tiến trình chỉ thấy được một vùng không gian địa chỉ của riêng nó. Tất cả các phiên bản của UNIX đều cung cấp cách bảo vệ bộ nhớ theo cơ chế bảo đảm không có tiến trình nào có thể ghi đè lên vùng nhớ của tiến trình khác đang hoạt động hoặc vùng nhớ của hệ thống. Nói chung, bộ nhớ mà hệ thống cấp phát cho một tiến trình không thể nào đọc hoặc ghi bởi một tiến trình khác-tránh khả năng xung đột bộ nhớ. Trong hầu hết các hệ thống Linux, con trỏ được sử dụng là một số nguyên 32 bit trỏ đến một ô nhớ cụ thể. Với 32 bit, hệ thống có thể đánh địa chỉ lên đến 4 GB bộ nhớ.

Mô hình bộ nhớ phẳng này dễ truy xuất và xử lý hơn bộ nhớ phân đoạn segment:offset. Ngoài ra, một vài hệ thống còn sử dụng mô hình địa chỉ 64 bit, như vậy không gian địa chỉ có thể mở rộng ra đến hàng Terabyte.

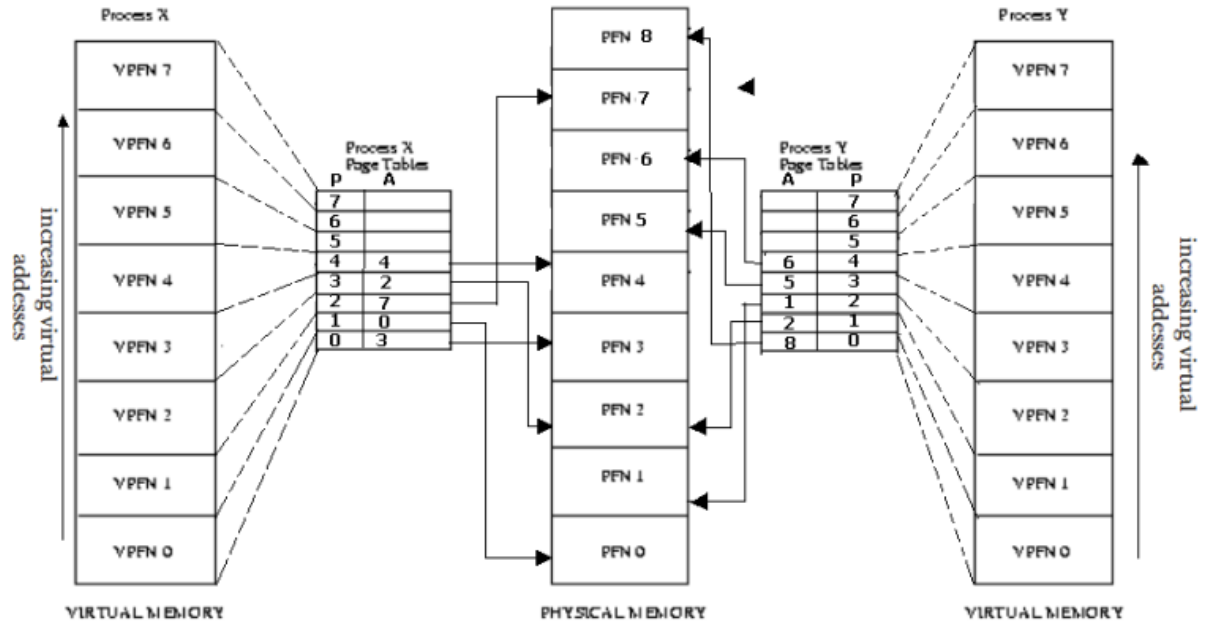
Để tăng dung lượng bộ nhớ sẵn có, Linux còn cài đặt chương trình phân trang đĩa tức là một lượng không gian hoán đổi nào đó có thể được phân bố trên đĩa. Khi hệ thống yêu cầu nhiều bộ nhớ vật lý, nó sẽ đưa các trang không hoạt động ra đĩa, nhờ vậy ta có thể chạy những ứng dụng lớn hơn và cùng lúc hỗ trợ nhiều người sử dụng. Tuy vậy, việc hoán đổi này không thay thế được bộ nhớ vật lý, nó chậm hơn vì cần nhiều thời gian để truy cập đĩa. Kernel cũng cài đặt khối bộ nhớ hợp nhất cho các chương trình người sử dụng và bộ đệm đĩa tạm thời (disk cache). Theo cách này, tất cả bộ nhớ trống dành để nhớ tạm và bộ nhớ đệm (cache) sẽ giảm xuống khi bộ xử lý chạy những chương trình lớn.

1.3. Cơ chế quản lý bộ nhớ của LINUX

Hệ thống con quản lý bộ nhớ là một trong các thành phần quan trọng của hệ điều hành. Máy tính luôn có nhu cầu cần nhiều không gian nhớ hơn không gian nhớ của bộ nhớ vật lý tồn tại trong một hệ thống. Nhiều chiến lược đã được phát triển để khắc phục vấn đề này và đa số đều thành công đó là sử dụng bộ nhớ ảo. Bộ nhớ ảo làm cho hệ thống có nhiều không gian nhớ hơn không gian nhớ thực tế bằng cách chia sẻ nó giữa các tiến trình khi các tiến trình này cần bộ nhớ. Bộ nhớ ảo làm cho không gian nhớ trên máy tính của ta rộng mở hơn. Hệ thống con quản lý bộ nhớ cung cấp: **Không gian địa chỉ rộng** (Large Address Spaces) Bộ nhớ ảo có thể có không gian nhớ lớn hơn nhiều lần bộ nhớ vật lý trong hệ thống. **Vấn đề bảo vệ bộ nhớ** (Protection) Mỗi một tiến trình trong hệ thống có không gian địa chỉ riêng của nó. Các không gian địa chỉ của các tiến trình hoàn toàn tách biệt nhau và khi một tiến trình đang chạy không thể ảnh hưởng đến tiến trình khác. Cũng vậy, kỹ thuật bộ nhớ ảo phần cứng cho phép các vùng trong bộ nhớ chống lại sự ghi đè. Kỹ thuật này bảo vệ mã lệnh và dữ liệu không bị ghi đè bởi các tiến trình không

hợp lệ. **Bản đồ bộ nhớ** (Memory Mapping) Bản đồ bộ nhớ được sử dụng để ánh xạ các file dữ liệu và các file ảnh vào một tiến trình đánh địa chỉ không gian nhớ. Nội dung của một file được liên kết trực tiếp với không gian địa chỉ của một tiến trình. **Phân phối bộ nhớ vật lý hợp lý** (Fair Physical Memory Allocation) Hệ thống con quản lý bộ nhớ cho phép mỗi một tiến trình đang chạy trong hệ thống được chia sẻ bộ nhớ vật lý của hệ thống một cách hợp lý. **Chia sẻ bộ nhớ ảo** (Shared Virtual Memory) Mặc dù bộ nhớ ảo cho phép các tiến trình có không gian địa chỉ riêng, xong nhiều khi ta vẫn cần tiến trình để chia sẻ bộ nhớ.

1.3.1. Một mô hình tóm tắt về bộ nhớ ảo



***Hình 1**. Mô hình tóm tắt ánh xạ từ bộ nhớ ảo tới bản đồ địa chỉ bộ nhớ vật lý.*

Khi bộ vi xử lý thi hành một chương trình, nó đọc lệnh từ bộ nhớ và giải mã. Trong khi giải mã lệnh, nó sẽ lấy hoặc cất nội dung của một ô nhớ trong bộ nhớ. Bộ vi xử lý sau khi thi hành lệnh sẽ chuyển đến lệnh tiếp theo trong chương trình, đảm bảo bộ vi xử lý luôn luôn truy cập bộ nhớ để nhận lệnh hoặc nhận và cất dữ liệu. Trong một hệ thống bộ nhớ ảo, tất cả các địa chỉ ảo được chuyển đổi thành địa chỉ vật lý do vi xử lý. Để thực hiện được việc chuyển đổi này, bộ nhớ ảo và bộ nhớ vật lý được chia thành các phần có kích thước hợp lý gọi là các trang *(pages)*. Tất cả các trang này có kích thước giống hệt nhau.

VD: Linux trên các hệ thống Alpha AXP sử dụng các trang có dung lượng 8 KB và trên các hệ thống Intel x86 nó sử dụng các trang có dung lượng là 4 KB. Mỗi trang này

được đánh số với chỉ số duy nhất, gọi là **số hiệu khung trang (page frame number - PFN).**

Trong mô hình đánh số trang này, một địa chỉ ảo gồm 2 phần: **địa chỉ offset** và **số hiệu khung trang (PFN).** VD: Nếu cỡ trang là 4 KB, trong 1 địa chỉ ảo thì các bit 0□11 của địa chỉ ảo sẽ thể hiện địa chỉ offset, các bit từ bit 12 đến bit cao hơn cho biết số hiệu khung trang ảo. Mỗi khi bộ vi xử lý xử lí địa chỉ ảo sẽ tách riêng địa chỉ offset và số hiệu khung trang, chuyển đổi số hiệu khung trang ảo thành số hiệu khung trang vật lý phù hợp rồi truy cập vào địa chỉ offset trong trang vật lý này nhờ ***bảng phân trang***.

Mỗi một phần tử trong bảng phân trang gồm các thông tin sau:

- Valid flag: Cờ xác định tính hợp lệ, nhận giá trị 0 hoặc 1.

- Số hiệu khung trang vật lý mà phần tử này đang diễn tả.

- Thông tin điều khiển truy cập của trang.

Để chuyển đổi một địa chỉ ảo sang một địa chỉ vật lý, đầu tiên bộ vi xử lý phải tính toán số hiệu khung trang địa chỉ ảo và địa chỉ offset trong trang ảo đó. Giả sử kích thước của một trang ảo là 1000h bytes (là 4096 bytes theo hệ thập phân) và một địa chỉ có giá trị 1194h trong không gian địa chỉ ảo của tiến trình thì địa chỉ đó sẽ được chuyển đổi thành địa chỉ offset 0194h thuộc trang ảo có số hiệu khung trang là 1. Nếu số hiệu khung trang ảo là hợp lệ (nhỏ hơn hoặc bằng chỉ số max trong bảng phân trang) bộ vi xử lý sẽ nhận số hiệu khung trang vật lý của phần tử này. Ngược lại, chỉ số này sẽ là chỉ số của vùng không tồn tại trong bộ nhớ ảo, bộ vi xử lý thực hiện chuyển đổi quyền điều khiển tới hệ điều hành để hệ điều hành có thể sửa chữa lại địa chỉ: bộ vi xử lý thông báo cho hệ điều hành rằng tiến trình hợp lệ đã cố gắng truy cập địa chỉ ảo nhưng không thể chuyển đổi được địa chỉ vật lý hợp lệ, địa chỉ này được coi là một sự hỏng trang và hệ điều hành được thông báo là địa chỉ ảo không hợp lệ. Đây là một **sự lỗi trang**. Nếu phần tử cần tìm trong bảng phân trang là hợp lệ, bộ vi xử lý lấy số hiệu khung trang vật lý và nhân với kích thước trang được kết quả làm **địa chỉ cơ sở** (địa chỉ đầu) của trang trong bộ nhớ vật lý. Sau đó cộng địa chỉ cơ sở này với địa chỉ offset để được địa chỉ của lệnh hoặc dữ liệu cần truy cập. VD: số hiệu khung trang ảo của tiến trình là 1, heo giá trị phần tử tương ứng trong bảng phân trang ta có số hiệu khung trang vật lý là 4, kích thước trang là 4Kb (1000h) địa chỉ cơ sở của trang vật lý sẽ là 4000h (4 x 1000h), cộng với địa chỉ offset là 0194h được địa chỉ vật lý là 4194h. Theo phương pháp trên, bộ nhớ ảo có thể ánh xạ vào các trang vật lý của hệ thống một cách bất kì.

1.3.2. Vùng trung gian (Swapping)

Nếu một tiến trình cần nạp một trang ảo vào bộ nhớ vật lý và không có sẵn trang vật lý tự do cho nó, hệ điều hành phải cấp phát bộ nhớ cho trang này bằng cách loại bỏ trang khác ra khỏi bộ nhớ vật lý, đây gọi là kỹ thuật **đổi trang** hay **thay trang**.

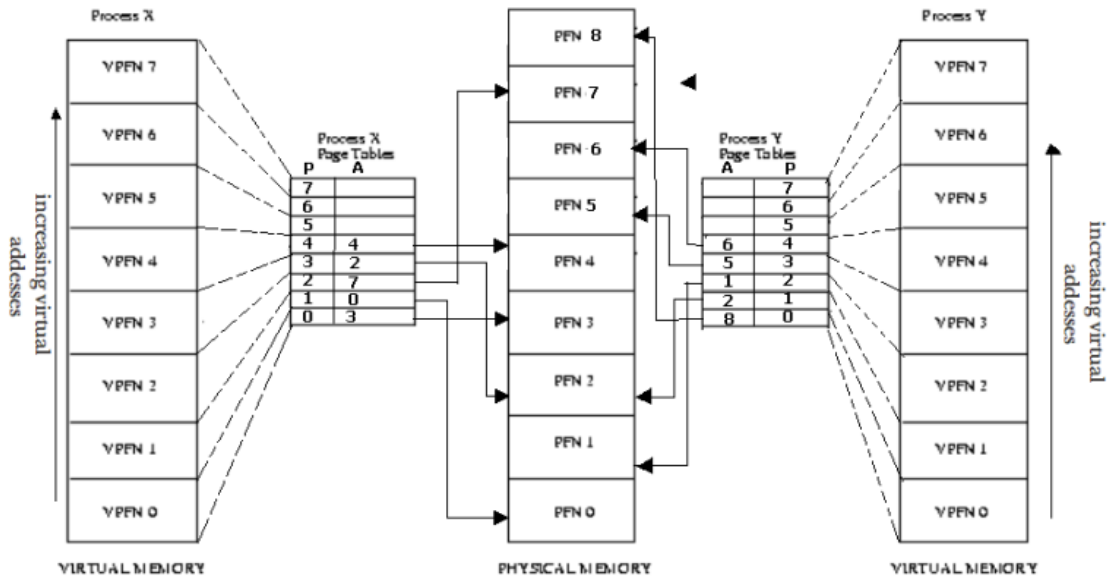
Nếu trang bị loại bỏ ra khỏi bộ nhớ vật lý thuộc nội dung của một file ảnh hoặc file dữ liệu và chưa từng bị thay đổi thì trang này không cần phải ghi lại lên đĩa. Để đổi trang chỉ cần loại bỏ trang này và nếu tiến trình cần nó một lần nữa, chỉ việc đưa trang trở lại bộ nhớ từ file ảnh hoặc dữ liệu.

Tuy nhiên, nếu trang đã bị thay đổi, hệ điều hành phải giữ nội dung của trang đó để truy cập về sau. Kiểu của trang này gọi là trang ***dirty***, khi nó bị di chuyển khỏi bộ nhớ sẽ được ghi vào một loại file đặc biệt là swap file (file trao đổi). Việc truy cập tới swap file liên quan rất nhiều tới tốc độ của bộ vi xử lý và bộ nhớ vật lý của hệ thống. Nhiều thuật toán đã được sử dụng để giải quyết vấn đề loại bỏ hoặc đưa ra swap file. Tập hợp các trang mà tiến trình đang sử dụng được gọi là *working set* (tập các trang đang làm việc). Một lược đồ thuật toán hoán đổi hiệu quả phải đảm bảo tất cả các tiến trình đều có working set hiệu quả trong bộ nhớ vật lý.

Linux sử dụng kỹ thuật "làm già trang" LRU (Least Recently Used - được sử dụng gần nhất) để lựa chọn trang có thể đưa ra khỏi bộ nhớ. Theo đó, các trang hoạt động trong hệ thống đều có "tuổi", "tuổi" của trang sẽ thay đổi khi trang được truy cập. Khi một trang được truy cập, tuổi của nó trở nên trẻ hơn, trang được truy cập ít hơn sẽ già hơn và nếu lâu không được truy cập nó có thể trở thành trang già nhất. Các trang già là các đối tượng tốt cho việc hoán đổi.

1.3.3. Bộ nhớ ảo dùng chung (Shared virtual memory)

Việc sử dụng bộ nhớ ảo làm cho các tiến trình có thể dễ dàng dùng chung bộ nhớ. Việc truy cập bộ nhớ được thực hiện thông qua các bảng phân trang và mỗi một tiến trình có một bảng phân trang riêng. Để 2 tiến trình dùng chung một trang vật lý trong bộ nhớ, số hiệu khung trang vật lý đó phải được đưa vào trong một phần tử thuộc cả 2 bảng phân trang của 2 tiến trình đó.



VD: Hình 1 cho thấy tiến trình X có khung trang ảo 3 và tiến trình Y có khung trang ảo 1 dùng chung trang vật lý 2.

1.3.4. Caches

Các phương pháp quản lý bộ nhớ trên tuy vậy lại không thực sự hiệu quả. Do vậy người thiết kế hệ điều hành và người thiết kế các bộ vi xử lý đều cố gắng nâng cao hiệu năng hệ thống. Xét về khía cạnh chế tạo vi xử lý, bộ nhớ, .v..v. để hệ thống hiệu quả hơn tốt nhất là cần duy trì các vùng nhớ đệm cho dữ liệu và thông tin hữu ích làm cho các thao tác nhanh hơn. Trong Linux sử dụng một số loại vùng nhớ đệm ***cache*** sau:

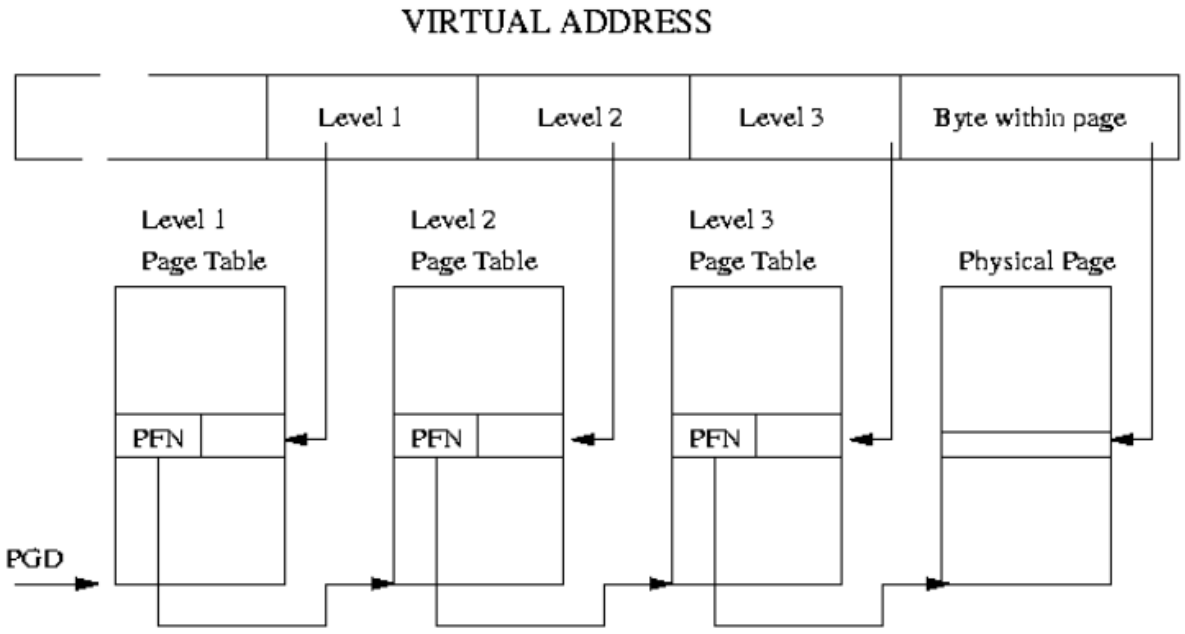
Vùng bộ đệm (Buffer Cache): chứa các bộ đệm dữ liệu (data buffers) được sử dụng bởi các trình điều khiển thiết bị khối.

Vùng trang nhớ (Page Cache): Vùng trang nhớ được sử dụng để tăng tốc độ truy cập các trang và dữ liệu trên đĩa.

Vùng lưu trữ trung gian (Swap Cache): Chỉ có các trang đã bị thay đổi (có kiểu *dirty*) mới được ghi vào file trung gian (swap file).

Các vùng đệm phần cứng (Hardware Caches)

1.3.5. Các bảng quản lý trang trong LINUX



Hình 2. *Các bảng quản lý trang 3 mức.*

Linux áp dụng các bảng quản lý trang 3 mức. Số hiệu khung trang của 1 bảng quản lý trang đều chứa thông tin bảng quản lý trang ở mức tiếp theo. Để chuyển đổi một địa chỉ ảo thành một địa chỉ vật lý, bộ vi xử lý lấy nội dung địa chỉ ảo của từng trường lưu trữ, chuyển đổi nó thành địa chỉ offset trong trang vật lý đang chứa bảng quản lý trang và đọc số hiệu khung trang của bảng quản lý trang mức tiếp theo. Việc này lặp lại 3 lần đến khi số hiệu khung trang của trang vật lý chứa địa chỉ ảo được tìm thấy, đồng thời nội dung của trường cuối cùng trong địa chỉ ảo (địa chỉ offset) sẽ được sử dụng để tìm dữ liệu trong trang.

1.3.5.1. Việc phân phối và thu hồi trang

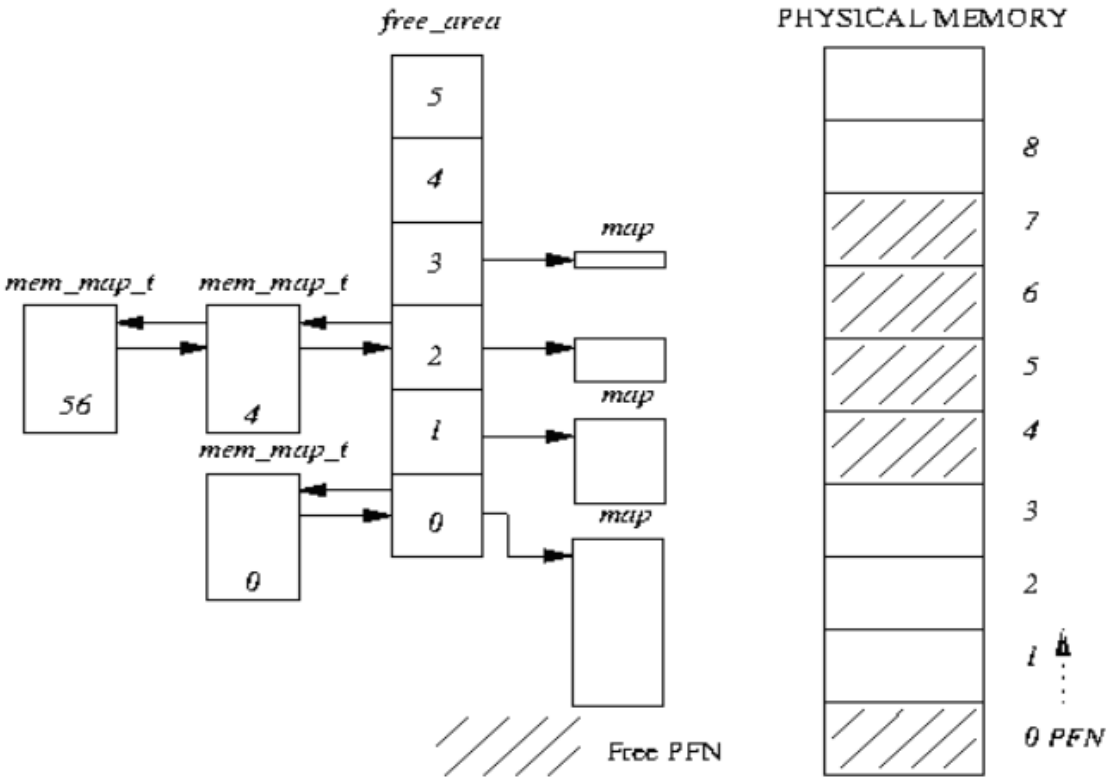
Có nhiều yêu cầu với các trang vật lý trong hệ thống. Ví dụ, khi một trang được nạp vào trong bộ nhớ, hệ điều hành cần phân phối các trang. Các trang này sẽ tự do khi các tiến trình xử lý chúng hoàn thành và các trang được loại bỏ khỏi hệ thống (với các trang vật lý cũng tương tự). Các kỹ thuật và các cấu trúc dữ liệu được sử dụng cho việc phân phối trang và thu hồi trang này có thể coi là tối ưu nhất trong việc duy trì 1 hệ thống con quản lý bộ nhớ ảo hiệu quả.

1.3.5.2. Việc phân phối trang

Linux sử dụng thuật toán Buddy để phân phối và thu hồi các khối của các trang, nó sẽ cố gắng phân phối một khối gồm một hoặc nhiều trang vật lý. Các trang được phân phối trong khối có số lượng là lũy thừa của 2 ví dụ như một khối gồm 1 trang, 2 trang,

4 trang, .v..v. miễn là số lượng này nhỏ hơn số trang tự do trong hệ thống. *free_area* là 1 mảng lưu giữ các khối trang. Hệ thống sẽ duyệt *free_area* để tìm một khối có số lượng trang đáp ứng yêu cầu. Mỗi phần tử của *free_area* có một sơ đồ các khối đã phân phối và các khối còn tự do cũng như kích thước các khối (theo đơn vị trang - page). Ví dụ phần tử 2 của mảng có sơ đồ nhớ diễn tả các khối còn tự do và các khối đã cấp phát, mỗi khối gồm 4 trang (4 pages).

Đầu tiên thuật toán duyệt các khối có kích thước phù hợp với số trang yêu cầu. Nếu không có khối tự do nào có đủ số trang yêu cầu, các khối có kích thước tiếp theo (là các khối có kích thước gấp 2 lần số trang yêu cầu) sẽ được tìm kiếm. Quá trình này được thực hiện cho đến khi tất cả các phần tử của *free_area* được duyệt hoặc khi tìm được một khối có số trang thỏa mãn. Nếu khối đã tìm thấy có số trang lớn hơn số trang yêu cầu, nó phải tách đôi khối này thành các khối nhỏ cho đến khi có một khối có số trang phù hợp theo yêu cầu. Các khối tự do được xếp thành một hàng đợi và các khối đã phân phối cùng các trang của nó được trả về cho đối tượng gọi nó.



***Hình 3**. Cấu trúc dữ liệu free_area*

Ví dụ, trong hình 3 nếu một khối gồm 2 trang được yêu cầu, khối đầu chỉ có 1 phần tử sẽ bị bỏ qua, khối thứ 2 có kích thước 4 trang sẽ được chọn (bắt đầu tại khung trang số 4). Nó sẽ được tách làm 2 khối mỗi khối 2 trang. Khối đầu tiên bắt đầu từ khung trang số 4 sẽ được trả về cho đối tượng gọi nó khi các trang này được phân phối và khối thứ 2 bắt đầu tại khung trang số 6 sẽ được xếp thành một khối gồm 2 trang vào phần tử 1 của mảng *free_area*.

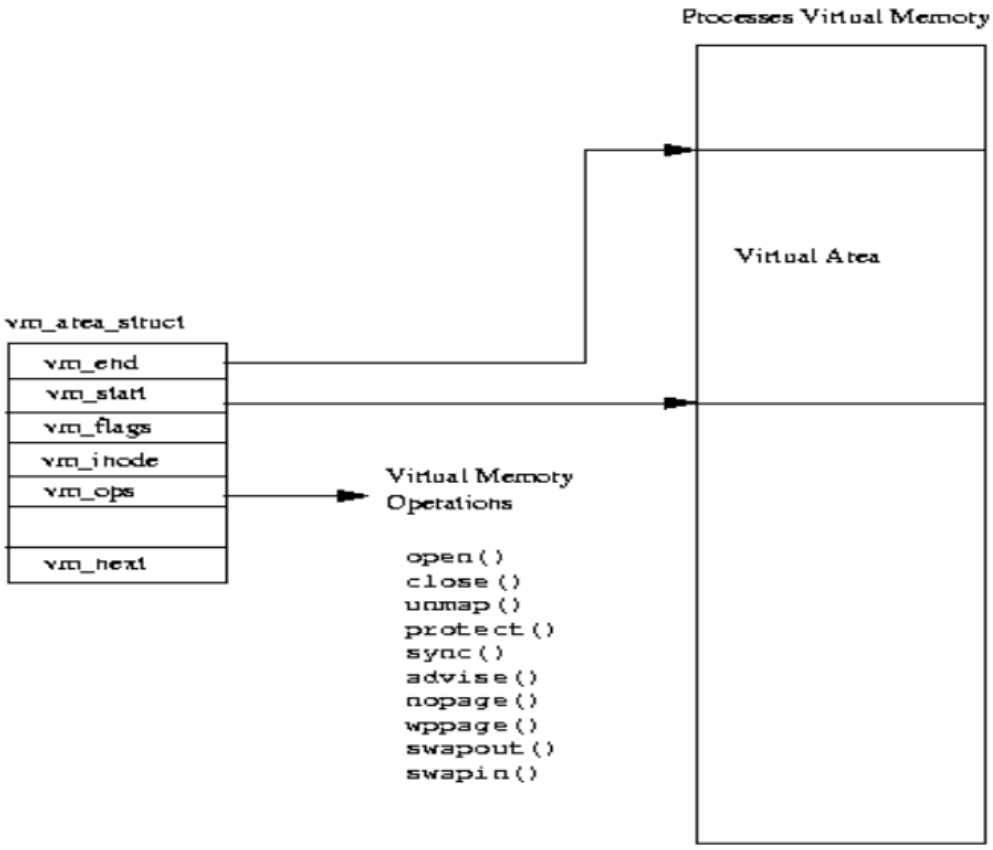
1.3.5.3. Thu hồi trang đã phân phối

Việc phân phối các khối nhớ gồm nhiều trang dẫn đến sự phân mảnh bộ nhớ do các khối nhớ tự do có số trang lớn hơn yêu cầu phải tách thành các khối nhỏ hơn. Mã lệnh thu hồi trang sẽ gộp các khối có số trang nhỏ thành các khối có số trang tự do lớn hơn bất cứ khi nào có thể.

Bất cứ khi nào một khối gồm các trang tự do cũng kiểm tra khối liền kề có kích thước tương tự xem các trang có tự do hay không, nếu nó tự do, nó sẽ được gộp với khối đó tạo thành một khối tự do mới có số lượng trang lớn hơn gấp đôi. Mỗi lần 2 khối được kết hợp lại thành một khối có số trang tự do lớn hơn. mã thu hồi trang cố gắng kết hợp khối đó thành một khối lớn hơn so với thời điểm hiện tại. Theo phương pháp này, các khối gồm các trang tự do ngày càng lớn hơn và sẽ có thể bộ nhớ sẽ trở thành một khối nhớ duy nhất. Ví dụ, trong hình 3 nếu khung trang số 1 vừa được giải phóng (được trả lại thành tự do), thì nó sẽ kết hợp với với khung trang số 0 đã tự do rồi và được xếp hàng vào phần tử số 1 của mảng *free_area* như một khối gồm 2 trang.

1.3.6. Vấn đề ánh xạ bộ nhớ

Khi một trang được thi hành, các nội dung của nó phải được truy cập thông qua không gian địa chỉ ảo của tiến trình. File có thể thi hành thực sự không đưa vào trong bộ nhớ vật lý mà chỉ được liên kết với bộ nhớ ảo của tiến trình. Khi các phần của chương trình được chạy, trang được đưa vào bộ nhớ từ file có thể thi hành. Việc liên kết một file vào không gian địa chỉ ảo gọi là ánh xạ bộ nhớ.



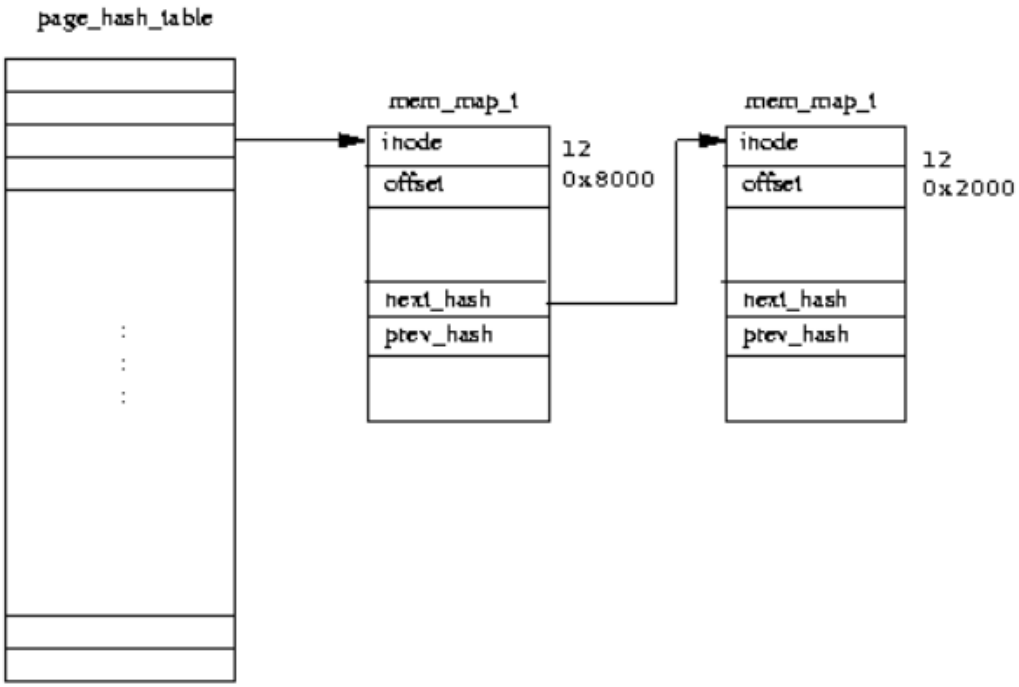
Hình 4*. Các vùng của bộ nhớ ảo*

Mỗi một bộ nhớ ảo của tiến trình được diễn tả bởi một cấu trúc dữ liệu là *mm_struct*. Cấu trúc này chứa thông tin về file hiện thời nó đang thi hành và có các con trỏ trỏ tới các cấu trúc dữ liệu *vm_area_struct,* cấu trúc dữ liệu này diễn tả vị trí bắt đầu và kết thúc của bộ nhớ ảo và các tiến trình có quyền truy cập tới vùng nhớ đó và thao tác với chúng. Các thao tác chính là tập các thủ tục mà Linux phải sử dụng khi tác động tới vùng nhớ ảo này.

Khi tiến trình cố gắng truy cập bộ nhớ ảo nhưng địa chỉ nhớ thực tế không tồn tại trong bộ nhớ vật lý (trường hợp truy cập bị lỗi trang), nếu một trong các thao tác với vùng nhớ ảo được thực hiện, thao tác này gọi là thao tác *nopage*. Thao tác *nopage* được sử dụng khi yêu cầu Linux đánh số các trang của một file thi hành trong bộ nhớ.

Khi một file có khả năng thi hành được ánh xạ vào một địa chỉ ảo tiến trình, một tập các cấu trúc dữ liệu *vm_area_struct* được tạo. Mỗi cấu trúc dữ liệu *vm_area_struct* diễn tả một phần của file có khả năng thi hành: mã có khả năng thi hành, dữ liệu ban đầu (biến), v.v.. Linux hỗ trợ một số thao tác bộ nhớ ảo chuẩn và khi cấu trúc dữ liệu *vm_area_struct* được tạo, một tập chuẩn các thao tác với bộ nhớ ảo được gắn kết với chúng.

1.3.6.1. Vùng trang đệm của Linux (The Linux Page Cache)



***Hình 5**. Vùng trang đệm của Linux*

Vai trò của vùng trang đệm của Linux là làm tăng tốc độ truy cập tới các file trên đĩa. Các file đã ánh xạ bộ nhớ được đọc một trang tại một thời điểm và các trang đó được lưu trữ trong vùng trang đệm. Vùng trang đệm có 1 vector là *page_hash_table*, có các con trỏ trỏ tới các cấu trúc dữ liệu *mem_map_t*.

Mỗi file trong Linux được định danh bởi một cấu trúc dữ liệu VFS *inode*, mỗi VFS *inode* là duy nhất và chỉ diễn tả một file. Chỉ số trong bảng quản lý trang được bắt nguồn từ VFS *inode* của file và địa chỉ offset trong file. Khi một trang được đọc từ một file đã ánh xạ bộ nhớ, nó cần được đưa vào bộ nhớ khi được yêu cầu, trang được đọc thông qua vùng trang đệm. Nếu trang đã có sẵn trong vùng trang đệm, một con trỏ trỏ tới cấu trúc dữ liệu *mem_map_t* đang chứa thông tin của nó sẽ được trả về cho mã xác định lỗi trang. Nếu không thì trang phải được nạp vào bộ nhớ từ hệ thống file đang lưu trữ. Linux sẽ phân phối một trang vật lý và đọc trang từ file trên đĩa.

Nếu có thể thực hiện được, Linux sẽ bắt đầu việc đọc trang tiếp theo trong file. Nếu tiến trình đang truy cập chuỗi các trang trong file, trang tiếp theo sẽ được đợi sẵn trong bộ nhớ chờ tiến trình xử lý.

Theo thời gian vùng trang đệm trở nên lớn hơn khi các ảnh được đọc và được thi hành. Các trang sẽ được loại bỏ khỏi vùng đệm khi chúng có "tuổi" lớn nhất để giảm kích thước của vùng trang đệm.

1.3.6.2. Hoán đổi vùng đệm (The Swap Cache)

Linux sử dụng bộ nhớ cache hoán đổi để theo dõi các trang. Bộ nhớ cache hoán đổi là một danh sách các mục của bảng trang, mỗi một trang vật lí trong hệ thống. Đây là một mục nhập của bảng trang cho một trang được hoán đổi và mô tả các tệp trao đổi tập tin đó đang được giữ trong cùng với vị trí của nó trong file swap. Nếu một mục trao đổi bộ nhớ cache khác không, nó đại diện cho một trang hiện đang có một tập tin hoán đổi mà chưa được sửa đổi. Nếu trang được sửa đổi sau (bằng cách ghi vào) , mục nhập của nó sẽ được xóa khỏi bộ nhớ cache hoán đổi.

Khi Linux cần phải trao đổi một trang vật lý ra ngoài để một tập tin hoán đổi tham chiếu các bộ nhớ cache hoán đổi, và nếu có một mục hợp lệ cho trang này, nó không cần phải ghi trang cho tập tin hoán đổi. Điều này là do các trang trong Bộ nhớ chưa được sửa đổi vì nó đã được đọc lần cuối từ tập tin hoán đổi.

1.3.6.3. Hoán đổi các trang đã nạp (Swapping pages in)

Các trang kiểu *dirty* đã ghi trong swap file có thể cần đến một lần nữa, ví dụ khi một ứng dụng ghi tới một vùng trong bộ nhớ ảo mà nội dung của nó đã được lưu trữ trong một trang vật lý lúc trước đã được tráo đổi ra ngoài. Việc truy cập một trang thuộc bộ nhớ ảo không được lưu trữ trong bộ nhớ vật lý là nguyên nhân gây ra lỗi trang.

Mã xử lý lỗi trang tìm kiếm phần tử trong bảng quản lý trang đối với địa chỉ ảo lỗi. Nếu phần tử mà nó tìm thấy cho biết trang đã tráo đổi ra ngoài bộ nhớ, Linux phải tráo đổi trang này trở lại bộ nhớ vật lý. Linux cần các thông tin vị trí swap file chứa phần tử cần tìm để nạp lại trang vào bộ nhớ vật lý.

Nó phân phối một trang vật lý tự do và nạp trang đã tráo đổi ra ngoài trở lại bộ nhớ từ swap file. Thông tin cho biết trang này nằm đâu đó trong swap file được lấy từ phần tử quản lý trang có thuộc tính invalid trong bảng quản lý trang.

Nếu việc truy cập với thao tác ghi là nguyên nhân gây lỗi trang thì trang bị bỏ lại trong vùng đệm tráo đổi và phần tử quản lý trang đó sẽ bị loại bỏ quyền được ghi. Về sau nếu trang này được ghi lên, một lỗi trang khác sẽ xảy ra, lúc này trang được đánh dấu là dirty, phần tử quản lý nó bị đẩy khỏi vùng đệm tráo đổi. Nếu trang không được ghi tới và nó cần được tráo đổi một lần nữa, Linux có thể ngăn ngừa việc ghi trang này tới swap file do trang đã tồn tại trong swap file. Nếu việc truy cập bằng thao tác ghi khiến trang được nạp lại vào bộ nhớ từ swap file, nó sẽ bị di chuyển khỏi vùng đệm tráo đổi và phần tử quản lý nó trong bảng quản lý trang được đánh dấu là dirty và sẽ được thêm quyền cho phép ghi file.

2. Vi xử lý Intel 80386

2.1. Tổng quan về Intel 80386

Intel 80386 (còn có các tên gọi là **80386**, **80386DX**, **386**, hay **386DX**) là bộ vi xử lý 32-bit của hãng Intel có từ năm 1985 được dùng trong các máy IBM (PS/2) và bản sao của máy IBM PC. Đối với lập trình viên nó còn gọi là kiến trúc **i386** hay x86_32.Intel gọi nó là IA-32. Chip này có thanh ghi 32-bit, với bus dữ liệu 32-bit và có khả năng địa chỉ hóa đến 32-bit. Giống 80286, nó có khả năng hoạt động trong hai chế độ thực (*real mode*) và bảo toàn (*protected mode*). Trong chế độ thực (chế độ này tương thích được với MS-DOS) dung lượng địa chỉ có thể xử lý là 1 MB. Trong chế độ bảo toàn, nó có thể địa chỉ hóa tới 4 GB. Ngoài ra, 80386, còn có khả năng vận hành trong chế độ 8086 ảo (*virtual 8086 mode*), chế độ này cho phép hệ điều hành chia con 80386DX ra thành nhiều phần mà mỗi phần tương đương với một bộ vi xử lý 8086, chúng có riêng 1 MB không gian nhớ và cho phép các *8086* này chạy chương trình riêng của nó. Từ Pentium 4 trở đi, Intel mở rộng thêm kiến trúc IA-32 thêm mode IA-32e. IA-32e chia ra thêm 2 mode nhỏ là compality mode và 64 bit mode.

Ảnh minh họa: *Intel 80386 DX và 33MHz đằng trước*

2.2. Quản lý bộ nhớ trên vi xử lí Intel 80386

Với mục tiêu phải tương thích ngược với các CPU đời cũ hơn, các CPU 80x86 (x>=3) cung cấp 3 cơ chế quản lý bộ nhớ :
1. real mode : đã có trong CPU 8088, CPU được dùng để xây dựng máy IBM PC đầu tiên. Đây là cơ chế quản lý bộ nhớ thật dùng kỹ thuật phân đoạn (segmentation).
2. protected mode : đã có trong CPU 80286. Đây là cơ chế quản lý bộ nhớ ảo dùng kỹ thuật phân đoạn (segmentation).
3. 386 enchanced mode : mới thêm vào cho các CPU từ 80386 trở lên. Đây là cơ chế quản lý bộ nhớ tổng hợp vừa phân đoạn vừa phân trang.

2.2.1. Quản lý bộ nhớ Real mode

Nguyên lý hoạt động :

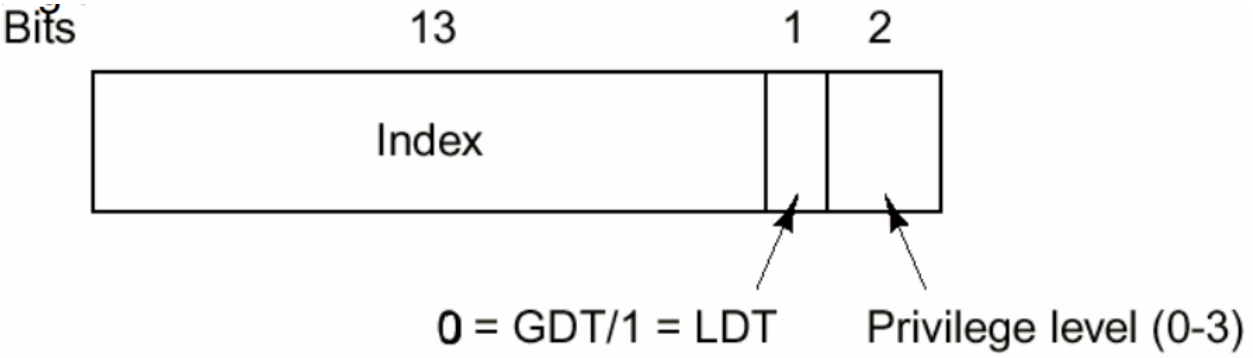
- Không gian bộ nhớ của chương trình là 1 tập các segment.
- Mỗi địa chỉ truy xuất được xác định bởi chương trình gồm 2 tham số : chỉ số segment + offset, mỗi tham số dài 16 bit. Ở góc nhìn lập trình, mỗi phần mềm có 216 segment, mỗi segment có 216 byte → mỗi chương trình dài maximum 4GB!
- Thường thì chương trình sẽ truy xuất tuần tự các ô nhớ nên tham số segment sẽ được chứa vào 1 trong các thanh ghi segment (CS, DS, ES, SS), mỗi lệnh máy chỉ cần miêu tả offset của ô nhớ cần truy xuất.
- Máy sẽ đổi địa chỉ ảo sang địa chỉ thật theo công thức sau :

  *physical address = segment * 16 + offset,*
- bit A20 của kết quả hoặc bị bỏ đi (trong chế độ A20 disable), hoặc được giữ lại và dùng (trong chế độ A20 enable). Chế độ A20 được thiết lập trong ROM BIOS.
- Theo cách đổi địa chỉ ảo như trên, ta thấy các segment phần mềm khác nhau có thể giao nhau, độ lệch tối thiểu của 2 segment là 16 ô nhớ (1 paragraph).
- Thí dụ các ô nhớ 0:80H ≡1:70H ≡ 2:60H ≡ 3:50H ≡ 4:40H ≡ 5:30H 6:20H ≡ 7:10H ≡ 8:0H đều chiếm cùng 1 ô nhớ RAM.
- không gian RAM mà chương trình truy xuất được thực tế là 1MB (A20 disable) hay 1MB + 65520 B (A20 enable). Ta gọi phần trên 1MB là HIGH MEMORY.

2.2.2. Quản lý bộ nhớ Protected mode

Nguyên lý hoạt động :

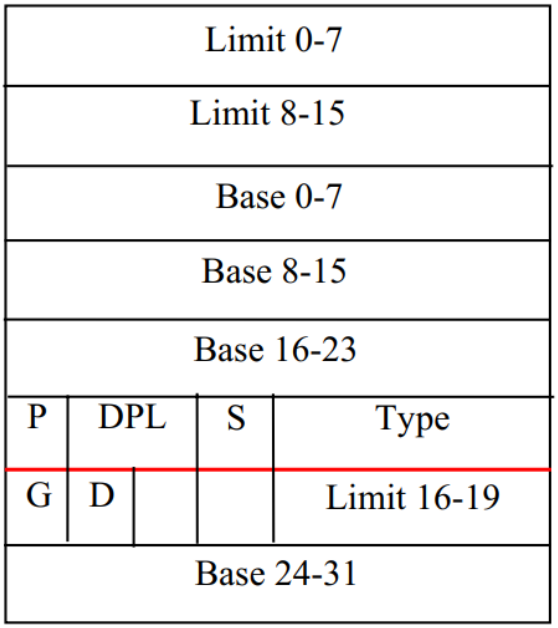
- Không gian bộ nhớ của chương trình là 1 tập các segment ảo.
- Mỗi địa chỉ truy xuất được xác định bởi chương trình gồm 2 tham số : chỉ số segment + offset, tham số segment dài 16 bit y như chế độ real mode, còn tham số offset có thể dài 32 bit. Như vậy, ở góc nhìn lập trình, mỗi phần mềm có 216 segment, mỗi segment có 232 byte → mỗi chương trình dài maximum 248 = 256TB!
- Thường thì chương trình sẽ truy xuất tuần tự các ô nhớ nên tham số segment sẽ được chứa vào 1 trong các thanh ghi segment (CS, DS, ES, SS), mỗi lệnh máy chỉ cần miêu tả offset của ô nhớ cần truy xuất.
- Nội dung trong thanh ghi segment không phải là chỉ số segment cần truy xuất mà nó được hiểu là “segment selector” gồm 3 thông tin :



- Như vậy, không gian bộ nhớ của chương trình có kích thước maximum là 8K segment toàn cục (global) và 8K segment cục bộ (local). Mỗi segment dài tối đa 4GB (dùng format 2- bit cho offset) → bộ nhớ chương trình tổng cộng là 64 TB.

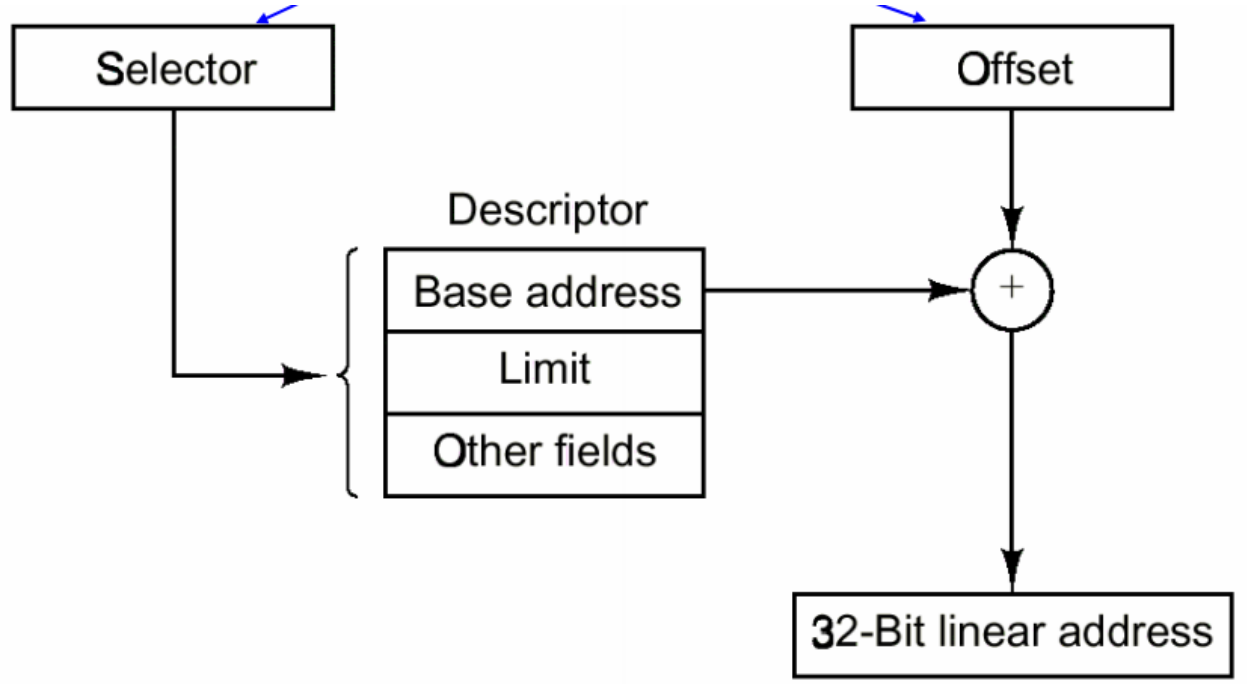
2.2.3. Cấu trúc record quản lý segment của Intel

- Base0-31 : địa chỉ RAM chứa segment.
- Limit0-19 : độ lớn segment (đơn vị tính là byte/page).
- D = 0 : Limit tính theo byte / D = 1: tính theo page 4KB.
- G = 0 : segment 16-bit / G = 1 : segment 32-bit.
- P = 0 : segment chưa nạp vào RAM / P = 1 : nạp vào RAM rồi.
- S = 0 : System / S = 1 : Application
- DPL : mức độ phân quyền từ 0 – 3.
- Type : kiểu segment và bảo vệ segment.

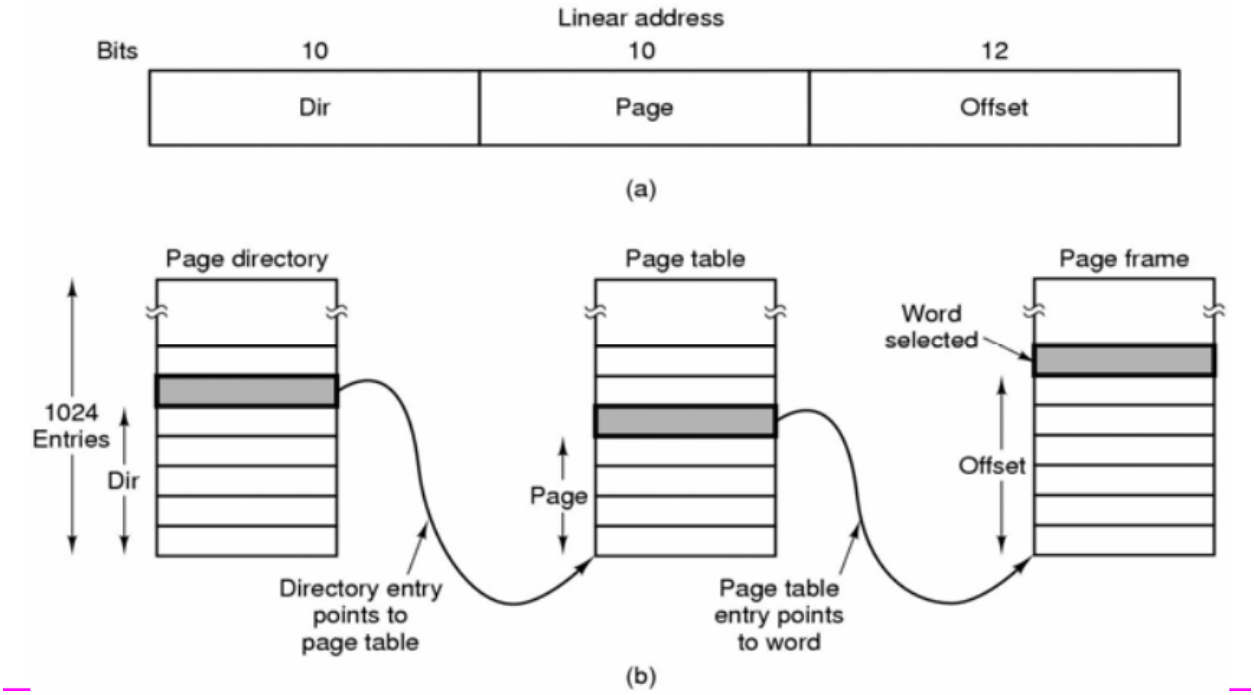


2.2.4 Qui trình đổi địa chỉ ảo sang địa chỉ thật

Địa chỉ luận lý có dạng segment:Offset, địa chỉ thật tương ứng là địa chỉ 32 bit, kết quả của phép cộng số học trong hình sau :



Địa chỉ ảo tuyến tính 32 bit được đổi sang địa chỉ thật của RAM bằng cơ chế phân trang 2 cấp như hình sau:



2.2.5. Quản lý bộ nhớ 368 enchanced mode

Nguyên lý hoạt động :

- Không gian bộ nhớ của chương trình là 1 tập các segment ảo.

- Mỗi địa chỉ truy xuất được xác định bởi chương trình gồm 2 tham số : chỉ số segment + offset, tham số segment dài 16 bit y như chế độ real mode, còn tham số offset có thể dài 32 bit. Như vậy, ở góc nhìn lập trình, mỗi phần mềm có 216 segment, mỗi segment có 232 byte → mỗi chương trình dài maximum 248 = 256TB!
- Thường thì chương trình sẽ truy xuất tuần tự các ô nhớ nên tham số segment sẽ được chứa vào 1 trong các thanh ghi segment (CS, DS, ES, SS), mỗi lệnh máy chỉ cần miêu tả offset của ô nhớ cần truy xuất.

TÀI LIỆU THAM KHẢO

[1] https://vi.wikipedia.org/wiki/Intel_80386

[2] https://vi.wikipedia.org/wiki/Linux