MÔN HỆ ĐIỀU HÀNH

Chương 5

QUẢN LÝ BỘ NHỚ

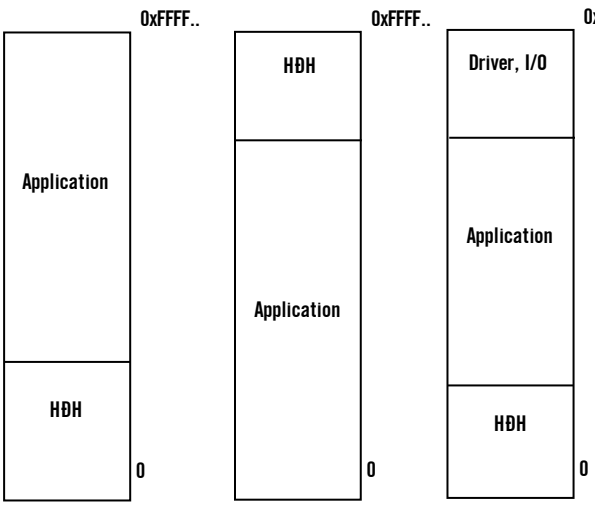
5.1 Tổng quát về quản lý bộ nhớ
5.2 Quản lý bộ nhớ thật
5.3 Quản lý bộ nhớ ảo
5.4 Quản lý bộ nhớ ảo phân trang
5.5 Quản lý bộ nhớ ảo phân đoạn
5.6 Quản lý bộ nhớ ảo phân đoạn và phân trang
5.7 Quản lý bộ nhớ của CPU Intel 80x86

Tài liệu tham khảo : chương 4, sách "Modern Operating Systems", Andrew S. Tanenbaum: , 2nd ed, Prentice Hall

5.1 Tổng quát về quản lý bộ nhớ

- Thường người lập trình muốn bộ nhớ mà chương trình truy xuất được có các tính chất :
  - dung lượng lớn
  - chạy nhanh
  - không bị mất thông tin.
- Thường máy tính sử dụng 3 loại bộ nhớ :
  - Cache : giá cao, dung lượng hạn chế, tốc độ cao
  - bộ nhớ chính DRAM : dung lượng trung bình, giá trung bình, tốc độ trung bình.
  - đĩa cứng : dung lượng rất lớn, chậm, giá rẻ.
- Module quản lý bộ nhớ phải tận dụng các ưu/khuyết điểm của các loại bộ nhớ máy tính để cung cấp cho người lập trình không gian làm việc thoả mãn càng nhiều yêu cầu càng tốt.

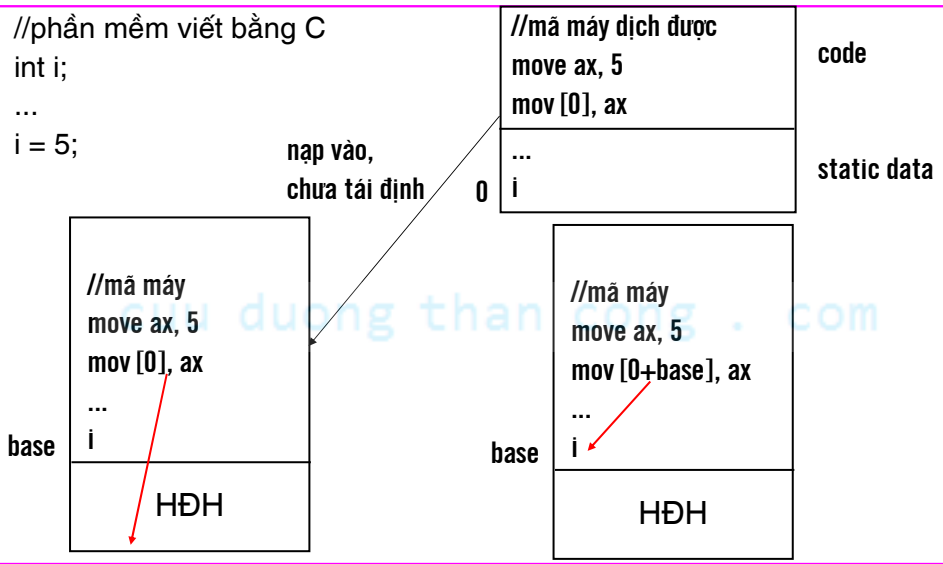
5.2 Quản lý bộ nhớ thật trên hệ đơn chương



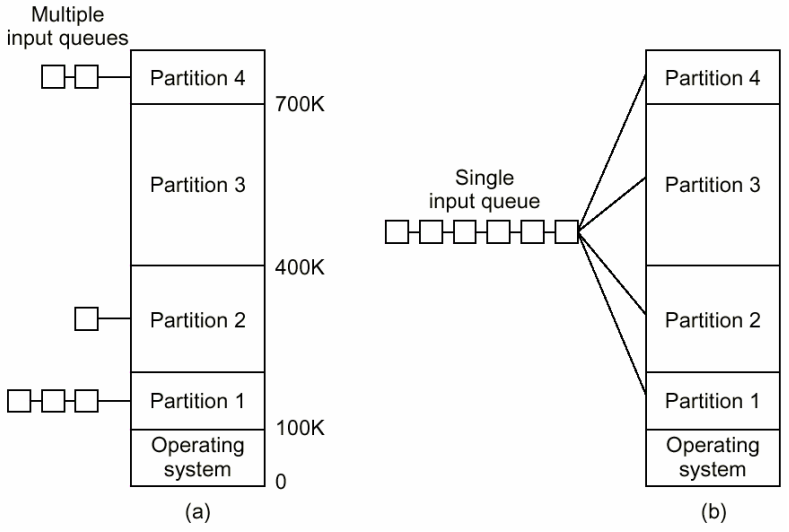
Một số vấn đề cần giải quyết :

- Tái định lại địa chỉ chương trình khi nạp fiel khả thi từ đĩa vào RAM.
- Bảo vệ bộ nhớ của HĐH từ việc truy xuất không hợp pháp của chương trình ứng dụng.
- Vấn đề không đủ chỗ cho chương trình lớn → sử dụng kỹ thuật Overlay để chia ứng dụng ra nhiều file overlay liên tiếp.

5.2 Quản lý bộ nhớ thật trên hệ đơn chương



5.3 Quản lý bộ nhớ thật trên hệ đa chương



5.3 Quản lý bộ nhớ thật trên hệ đa chương

Trong bối cảnh máy có dung lượng RAM khá lớn (512KB), còn các phần mềm cần chạy có kích thước khá nhỏ (10-100KB), ta có thể dùng 1 trong 3 kỹ thuật quản lý bộ nhớ :

1. Kỹ thuật phân vùng tĩnh dùng nhiều hàng chờ độc lập (hình a slide trước) :

- HĐH được load vào vùng bộ nhớ thấp của RAM.
- phần trống còn lại của RAM sẽ được chia làm nhiều phân vùng có kích thước tăng dần (10, 20, 40, 80, 160KB,...).
- mỗi phân vùng có 1 hàng chờ các ứng dụng cần chạy trên phân vùng tương ứng.
- khi cần chạy ứng dụng, người chạy ứng dụng phải chọn phân vùng có kích thước nhỏ nhất nhưng >= kích thước ứng dụng và sắp hàng ở hàng chờ tương ứng.
- HĐH sẽ phục vụ các ứng dụng trong từng hàng chờ theo thứ tự ai đến trước phục vụ trước.

Quản lý bộ nhớ thật trên hệ đa chương

Kỹ thuật dùng nhiều hàng chờ độc lập có 2 khuyết điểm chính :

1. Kích thước các phân vùng tĩnh thường không khớp với kích thước ứng dụng nên bị lãng phí.
2. sử dụng các phân vùng thường không đều gây ra lãng phí : nhiều phần mềm sắp hàng chạy trên phân vùng kích thước nhỏ, trong lúc phân vùng kích thước lớn không có ứng dụng chạy.

Để khắc phục khuyết điểm 2 ở trên, ta có thể dùng kỹ thuật sau :

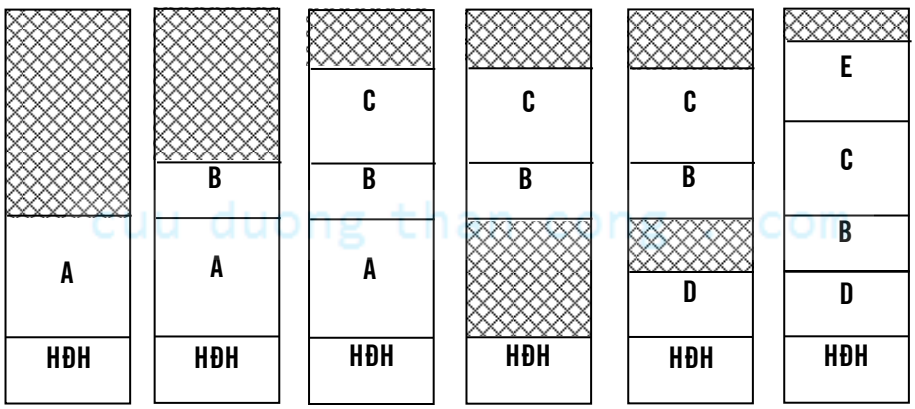
2. Kỹ thuật phân vùng tĩnh dùng 1 hàng chờ duy nhất (hình b slide trước) :

- HĐH được load vào vùng bộ nhớ thấp của RAM.
- phần trống còn lại của RAM sẽ được chia làm nhiều phân vùng có kích thước tăng dần (10, 20, 40, 80, 160KB,...).
- chỉ có 1 hàng chờ các ứng dụng cần chạy trên các phân vùng.
- khi 1 phân vùng rãnh, HĐH sẽ dò trong hàng chờ 1 ứng dụng có kích thước lớn nhất nhưng <= kích thước phân vùng và cho phép ứng dụng này chạy.

Quản lý bộ nhớ thật trên hệ đa chương

Để khắc phục khuyết điểm 1 ở silde trước, ta có thể dùng kỹ thuật sau :

3. Kỹ thuật phân vùng động :



Quản lý bộ nhớ thật trên hệ đa chương

3. Kỹ thuật phân vùng động :

a. HĐH được load vào vùng bộ nhớ thấp của RAM. Khi ứng dụng A cần chạy, HĐH tạo phần vùng động vừa đúng kích thước của phần mềm A và nạp A vào phân vùng vừa tạo. Phân bộ nhớ trống còn lại để dành cho các phần mềm khác sau này.
b. tương tự tạo phân vùng cho B chạy.
c. tương tự tạo phân vùng cho C chạy.
d. A kết thúc và trả lại phân vùng mình đã chiếm.
e. khi D chạy, HĐH tạo phần vùng động vừa đúng kích thước cho D.
f. khi E chạy, HĐH không tìm được vùng trống nào đủ lớn cho E cả. Trong trường hợp này, HĐH sẽ dời các phân vùng đang dùng bởi các ứng dụng lại kề nhau để tạo phân vùng trống duy nhất. Nếu kích thước của nó đủ chạy cho E thì sẽ tạo phân vùng động cho E chạy.

5.3 Quản lý bộ nhớ ảo trên hệ đa chương

Tất cả các phương pháp quản lý bộ nhớ đã trình bày trước đây đều có nhược điểm là dựa vào ý tưởng : nạp toàn bộ file phần mềm vào bộ nhớ trước khi chạy ứng dụng tương ứng.

Trong bối cảnh sử dụng máy hiện nay, máy chỉ có RAM kích thước vừa phải (<=4GB), nhưng phải chạy đồng thời nhiều ứng dụng, mỗi ứng dụng lại có nhu cầu bộ nhớ rất lớn, nhiều khi lớn hơn cả kích thước của RAM. Như vậy, các phương pháp trước đây đều thất bại không giải quyết nổi yêu cầu mới này.

Người ta đã tìm được phương pháp quản lý bộ nhớ đáp ứng được yêu cầu mới, đó là phương pháp quản lý bộ nhớ ảo. Ý tưởng nền tảng của phương pháp này là : không cần nạp hết chương trình vào bộ nhớ trước khi chạy mà khi ứng dụng chạy tới lệnh nào và truy xuất dữ liệu nào thì HĐH mới nạp phần chương trình chứa lệnh và dữ liệu cần chạy, sau đó khi cần thì giải phóng vùng nhớ để chứa phần code và dữ liệu khác.

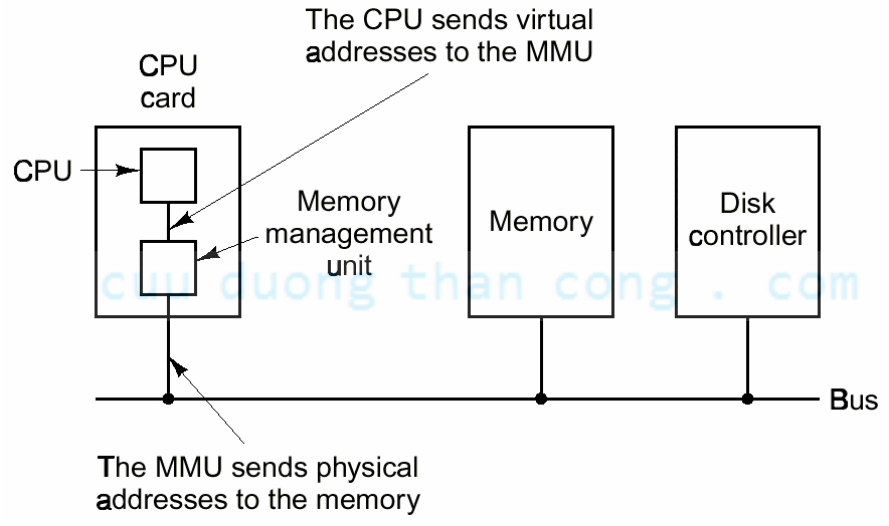
5.3 Quản lý bộ nhớ ảo trên hệ đa chương

Có 3 phương pháp quản lý bộ nhớ ảo khác nhau :

- Quản lý bộ nhớ phân trang (Paging)
- Quản lý bộ nhớ phân đoạn (Segmentation)
- Quản lý bộ nhớ phân đoạn và phân trang (Segmentation & Paging)

Để đạt được tốc độ cần thiết, người ta phải hiện thực các phương pháp quản lý bộ nhớ ảo bằng phần cứng. Đơn vị phần cứng quản lý bộ nhớ ảo được gọi là MMU (Memory Management Unit). Đơn vị MMU thường trong trong CPU.

Quản lý bộ nhớ ảo trên hệ đa chương



5.4 Quản lý bộ nhớ ảo phân trang

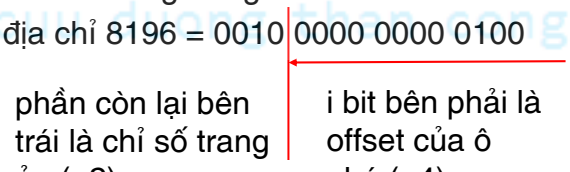
Nguyên lý hoạt động :

- khi lập trình, các lệnh truy xuất các địa chỉ trong không gian phẳng có dùng lượng rất lớn (4GB). Không gian này được gọi là luận lý (ảo). Nếu phần mềm hạn chế truy xuất trong không gian này thì HĐH sẽ đảm bảo nó chạy tốt cho dù kích thước thật của RAM nhỏ hơn nhiều.
- Để quản lý việc swap (nạp vào/ghi ra) bộ nhớ ảo, HĐH chia bộ nhớ ảo của ứng dụng ra thành nhiều đơn vị quản lý có kích thước đồng nhất, mỗi đơn vị được gọi là trang ảo. Kích thước trang ảo = 2^i (256, 512, 1K, 2K, 4K, 8K,...).
- Bộ nhớ RAM cũng được chia thành nhiều đơn vị quản lý, mỗi đơn vị được gọi là trang thật (page frame). Kích thước thật = kích thước trang ảo.

Quản lý bộ nhớ ảo phân trang

Nguyên lý hoạt động (tt) :

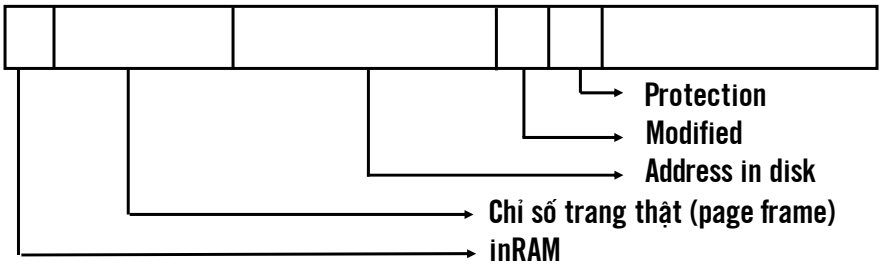
- Trang thật là nơi chứa trang ảo khi cần thiết, tại từng thời điểm, mỗi trang thật chứa tốt đa 1 trang ảo, nhưng theo thời gian nó có thể chứa nhiều trang ảo khác nhau.
- Khi ứng dụng truy xuất 1 ô nhớ theo địa chỉ tuyến tính số nguyên, HĐH biết ngay ô nhớ đó thuộc trang ảo nào, nằm ở Offset nào trong trang ảo.
- ô ở địa chỉ 8196 = 0010 0000 0000 0100



Quản lý bộ nhớ ảo phân trang

Nguyên lý hoạt động (tt) :

- Để quản lý quá trình ánh xạ các trang ảo của chương trình vào các trang thật, HĐH dùng 1 bảng đặc tả trang ảo cho mỗi chương trình, bảng này có số phần tử = số trang ảo của chương trình tương ứng, mỗi phần tử của bảng là 1 record chứa các thông số quản lý trang ảo tương ứng :



Quản lý bộ nhớ ảo phân trang

Qui trình đổi địa chỉ ảo sang địa chỉ thật :

1. từ địa chỉ mà chương trình truy xuất (addr), hệ thống tách thành 2 thành phần : page (i) và offset.
2. Truy xuất record quản lý trang ảo i trong bảng đặc tả trang. Nếu field inRAM=1 thì địa chỉ thật tương ứng là :

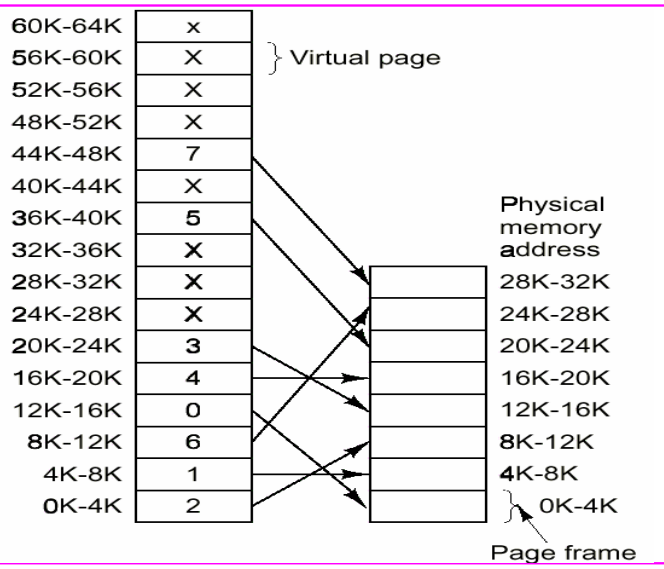
page frame	Offset

   và qui trình kết thúc.
3. Nếu inRAM =0, hệ thống sẽ tìm 1 trang thật rãnh (k), nếu không có phải tìm cách giải phóng 1 trang thật ít gây phiền hà nhất (k), dựa vào thông tin trong field "inDisk" để mở file và đọc trang ảo vào trang thật k.
4. Hiệu chỉnh lại field inRAM = 1 và field page frame = k rồi quay lại bước 2.

Quản lý bộ nhớ ảo phân trang

Thí dụ về trạng thái hệ thống, RAM có 32KB được chia làm 8 trang thật chỉ số từ 0 -7, mỗi trang 4KB. Chương trình dài 64KB được chia làm 16 trảng ảo, chỉ số từ 0 -15.

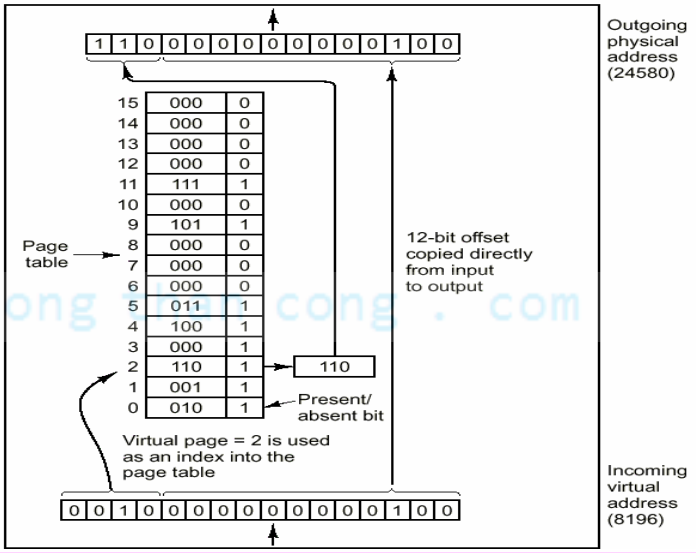
Hiện có 8 trang ảo được nạp vào và chiếm hết RAM.



Quản lý bộ nhớ ảo phân trang

Thí dụ về bảng đặc tả trang và qui trình đổi địa chỉ ảo sang địa chỉ thật.

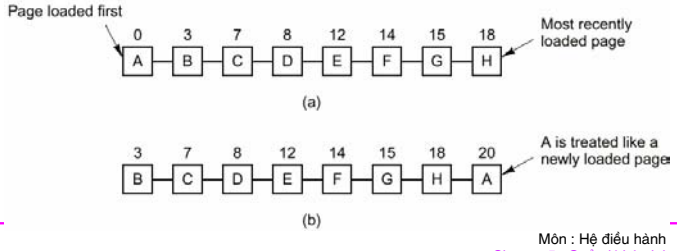
Theo hình bên, phần mềm truy xuất ô nhớ ảo ở địa chỉ 8196 thì máy truy xuất RAm ở địa chỉ 24580.



Các phương pháp giải phóng trang thật

Trong qui trình đổi địa chỉ ảo sang địa chỉ thật ở các slide trước, ta thấy mỗi khi cần nạp trang ảo mới, máy sẽ tìm trang thật trống, nhưng ít khi tìm được, do đó máy sẽ phải tìm 1 trang thật đang dùng nào đó và giải phóng nó. Có nhiều phương pháp giải phóng trang thật khác nhau, mỗi phương pháp có giá/kết quả khác nhau, tùy thuộc vào mục tiêu xây dựng HĐH cụ thể, ta có thể chọn 1 trong các phương pháp này hay kết hợp chúng lại.

1. Phương pháp FIFO : dùng 1 danh sách liên kết chứa các trang thật được dùng, trang nào được dùng mới nhất sẽ được đưa vào cuối danh sách → đầu danh sách là trang được dùng cũ nhất → chọn nó để giải phóng khi cần.



Các phương pháp giải phóng trang thật

Phương pháp FIFO có nhược điểm là thường giải phóng nhằm các trang chứa HĐH (vì các module chức năng của HĐH được nạp vào RAM đầu tiên). Để khắc phục nhược điểm này, ta có thể dùng phương pháp sau :

2. Phương pháp cho cơ hội lần 2 : kết hợp mỗi trang thật 1 bit trạng thái, bit R = 0/1 (Recent). Định kỳ bit R của các trang sẽ bí xóa về 0, mỗi khi trang bị truy xuất thì set R = 1. Khi cần giải phóng trang, chọn phần tử đầu danh sách. Nếu bit R = 0 thì giải phóng nó, còn nếu R = 1 thì set lại R=0, tha nó và đưa nó về đuôi danh sách để xử lý sau. Như vậy, với phương pháp này, ta chỉ giải phóng trang được dùng cũ nhất và không được truy xuất lại trong quá khứ gần đây.

Các phương pháp giải phóng trang thật

Phương pháp cơ hội lần 2 còn tốn nhiều chi phí cho mỗi lần xử lý. Cụ thể để đưa được phần tử đầu của danh sách về cuối danh sách, ta cần thực hiện 4 lệnh gán sau :

```
tail->next = head;     //đưa phần tử đầu danh sách về cuối danh sách
tail = head;           //hiệu chỉnh lại pointer tail của danh sách
head = head->next;     //đưa phần tử đầu danh sách về phần tử kế tiếp
tail->next = null;     //xóa vùng next của phần tử cuối danh sách
```

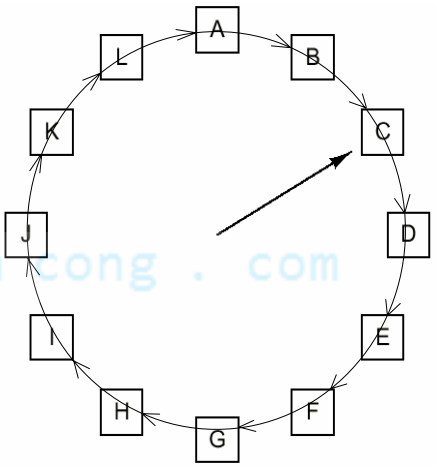
Để giảm nhẹ hơn nữa chi phí quản lý danh sách, ta có thể dùng danh sách liên kết vòng thay vì danh sách liên kết đơn. Đây là ý tưởng của phương pháp "Clock" để giải phóng trang.

Các phương pháp giải phóng trang thật

3. Phương pháp "Clock" :

Để đưa được phần tử đầu của danh sách về cuối danh sách, ta chỉ cần thực hiện 1 lệnh gán sau :

```
//đưa phần tử đầu danh sách về phần tử kế tiếp
head = head->next;
```



Các phương pháp giải phóng trang thật

4. Phương pháp "Not Recently-Used" - NRU : Để giải phóng trang ít gây phiền hà hơn, ta sẽ kết hợp mỗi trang thật 1 bit trạng thái nữa. Cụ thể ta có 2 bit miêu tả trạng thái của từng trang thật như sau :
   - bit R = 0/1 (Recent). Định kỳ bit R của các trang sẽ bí xóa về 0, mỗi khi trang bị truy xuất thì set R = 1.
   - bit M = 0/1(Modified). Mỗi lần nạp trang ảo, bit M của trang thật được xóa 0. Mỗi lần bị thay đổi nội dung, bit M được set lên 1.

   Như vậy mỗi trang sẽ ở 1 trong 1 trạng thái sau :
   1. S0 : R = M = 0 : trong quá khứ gần, trang chưa được truy xuất và trang chưa bị thay đổi nội dung.
   2. S1 : R = 0, M = 1 : trong quá khứ gần, trang chưa được truy xuất nhưng trang đã bị thay đổi nội dung.
   3. S2 : R = 1, M = 0 : trong quá khứ gần, trang được truy xuất, nhưng trang chưa bị thay đổi nội dung.
   4. S3 : R = 1, M = 1 : trong quá khứ gần, trang được truy xuất và trang đã bị thay đổi nội dung.

   Khi cần giải phóng trang, ta chọn trang theo thứ tự ưu tiên từ S0 -> S3.

Các phương pháp giải phóng trang thật

Trong phương pháp NRU ở slide trước, nếu có nhiều trang S0 thì ta phải dùng cơ chế ngẫu nhiên để chọn 1 trong các trang đó chưa chính xác lắm. Để khắc phục ngược điểm này, ta dùng phương pháp sau :

5. Phương pháp "Least Recently-Used" - LRU : kết hợp mỗi trang 1 vùng thông tin miêu tả mộc thời gian. Mỗi lần trang được truy xuất, ta ghi thời điểm truy xuất vào mộc thời gian của trang. Mỗi khi cần giải phóng trang, ta chọn trang có mộc thời gian nhỏ nhất (trang được truy xuất lần cuối lâu nhất).

Tối ưu hóa qui trình đổi địa chỉ ảo sang thật

Qui trình đổi địa chỉ ảo sang thật được trình bày trong các slide trước có 2 vấn đề lớn sau đây cần chú ý và giải quyết :

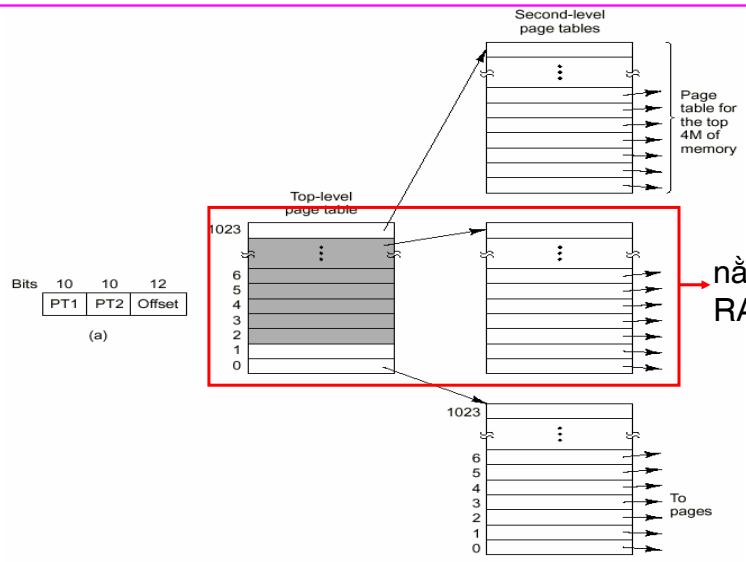
1. bảng đặc tả trang cho chương trình chiếm rất nhiều chỗ. Thí dụ không gian ảo của chương trình là 4GB, trang ảo là 4KB. Như vậy, ta cần có bảng đặc tả trang cho chương trình chứa 4G/4K = 1M phần tử, mỗi phần tử là 1 record (td. dài 16 byte). Kết quả là bảng đặc tả trang sẽ chiếm 16MB và về nguyên tắc phải nằm thường trực trong RAM → quá tốn chỗ!
2. nếu thuật giải đổi địa chỉ ảo sang thật được viết bằng phần mềm, nó sẽ gồm hàng ngàn lệnh máy, như vậy mỗi lần chương trình thực hiện 1 lệnh máy có truy xuất bộ nhớ, máy sẽ dừng lại chạy thuật giải đổi địa chỉ gồm hàng ngàn lệnh máy mới có kết quả giúp lệnh máy của chương trình chạy tiếp → quá kém hiệu quả!

Tối ưu hóa qui trình đổi địa chỉ ảo sang thật

Để khắc phục nhược điểm 1 trong slide trước, người ta dùng cơ chế phân trang nhiều cấp thay vì 1 cấp. Thí dụ Windows dùng cơ chế phân trang 2 cấp như sau :

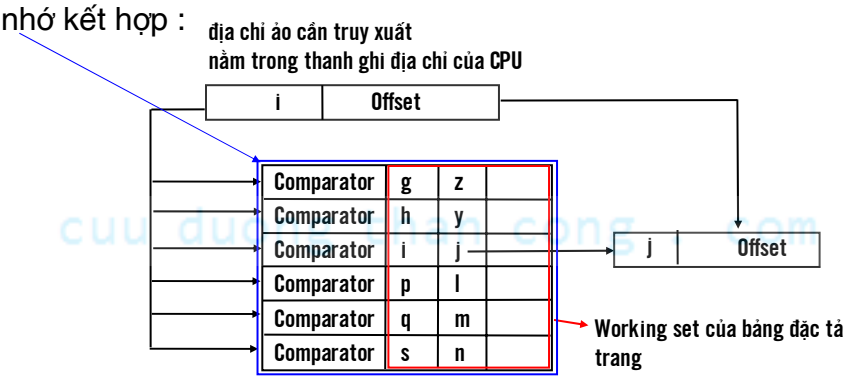
- chia không gian ảo của phần mềm (4GB) thành n (=1024) trang ảo cấp 1 (Dir),
- mỗi trang ảo cấp 1 có dung lượng rất lớn (=4MB) được chia nhỏ thành m (=1024) trang ảo cấp 2, mỗi trang ảo cấp 2 có kích thước 4KB và là đối tượng swap vật lý giữa RAM/đĩa cứng.
- Như vậy bảng đặc tả trang cho chương trình chỉ chứa n (=1024) phần tử, rất nhỏ so với trước đây (1M phần tử). Tuy nhiên mỗi trang ảo cấp 1 có bảng đặc tả trang riêng dài m (=1024) phần tử. May mắn là các bảng đặc tả trang cấp 2 không cần nằm thường trực trong RAM, chúng được chứa trên đĩa. Khi truy xuất vào trang ảo cấp 1 nào, bảng đặc tả trang cấp 2 của nó mới được nạp vào RAM.

Tối ưu hóa qui trình đổi địa chỉ ảo sang thật



Tối ưu hóa qui trình đổi địa chỉ ảo sang thật

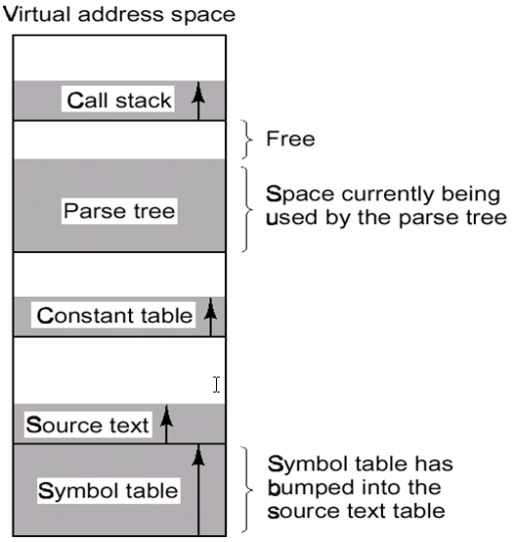
Để khắc phục nhược điểm 2 trong slide trước, thay vì dùng giải pháp phần mềm để đổi địa chỉ, ta dùng phương pháp phần cứng dựa vào bộ nhớ kết hợp :



Toàn bộ các mạch điện tử trên đều là mạch tổ hợp nên thời gian đổi địa chỉ ảo sang thật hầu như = 0, quá tốt!

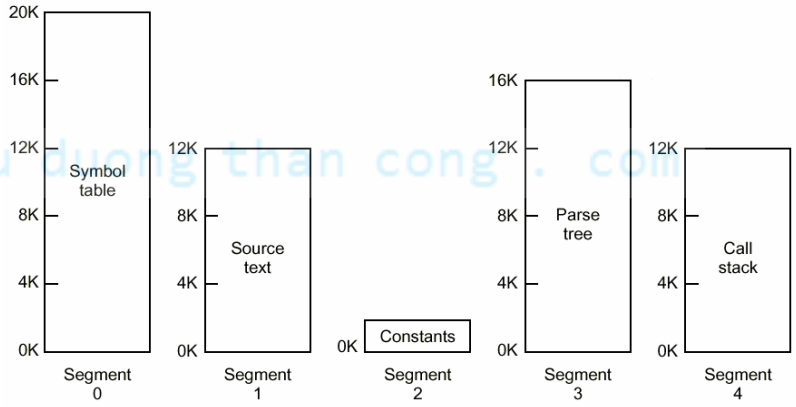
5.5 Quản lý bộ nhớ ảo phân đoạn

Trong kỹ thuật phân trang, mặc dù không gian ảo mà chương trình có thể truy xuất có kích thước rất lớn (4GB), nhưng nó là không gian phẳng nên 1 số chương trình lớn vẫn còn gặp phiền hà sau đây : chương trình tự chia không gian ảo của mình ra thành nhiều partition khác nhau để chứa những thông tin độc lập cần được xử lý, trong quá trình chạy, nếu 1 trong các partition không đủ chỗ chứa thông tin thì chương trình sẽ bị dừng độ ngột.



5.5 Quản lý bộ nhớ ảo phân đoạn

Do đó, thay vì cấp phát cho chương trình 1 không gian phẳng duy nhất, nếu hệ thống cấp phát cho chương trình các không gian bộ nhớ độc lập có kích thước thay đổi động theo nhu cầu (miễn sao tổng kích thước của chúng bị hạn chế trên nào đó) như hình sau thì chương trình sẽ không gặp vấn đề tràn bộ nhớ như slide trước :



5.5 Quản lý bộ nhớ ảo phân đoạn

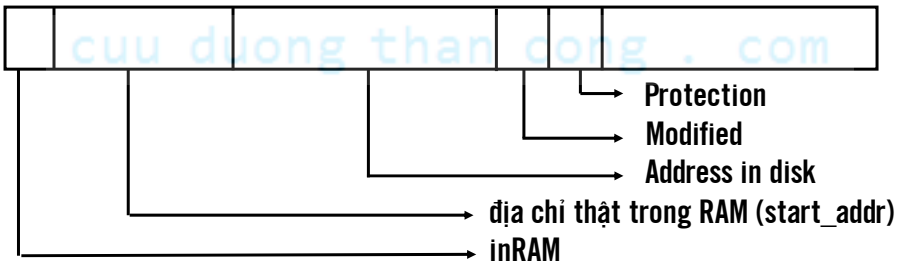
Nguyên lý hoạt động :

- khi lập trình, chương trình được phép truy xuất dữ liệu trong nhiều không gian khác nhau, mỗi không gian được gọi là segment. Mỗi segment có kích thước thay đổi được theo thời gian, ô nhớ đầu tiên của mỗi segment luôn bắt đầu từ 0.
- Bộ nhớ RAM có kích thước nhỏ nào đó. Các segment của chương trình thường nằm trên đĩa cứng, khi cần thiết segment sẽ được nạp vào 1 vùng thích hợp trong RAM.
- Tại từng thời điểm, 1 vùng nhớ RAM thật chứa tốt đa 1 segment ảo, nhưng theo thời gian nó có thể chứa nhiều segment ảo khác nhau.
- Khi ứng dụng truy xuất 1 ô nhớ, nó xác định địa chỉ ô nhớ dạng phân cấp : segment + offset.

Quản lý bộ nhớ ảo phân đoạn

Nguyên lý hoạt động (tt) :

- Để quản lý quá trình ánh xạ các segement ảo của chương trình vào các vùng RAM, HĐH dùng 1 bảng đặc tả sgement cho mỗi chương trình, bảng này có số phần tử = số segement của chương trình tương ứng, mỗi phần tử của bảng là 1 record chứa các thông số quản lý segement tương ứng :



Quản lý bộ nhớ ảo phân đoạn

Qui trình đổi địa chỉ ảo sang địa chỉ thật :

1. từ địa chỉ mà chương trình truy xuất gồm 2 thành phần : segement (i) và offset.
2. Truy xuất record quản lý segement i trong bảng đặc tả segement. Nếu field inRAM=1 thì địa chỉ thật tương ứng là :

   start_addr + Offset và qui trình kết thúc.

3. Nếu inRAM =0, hệ thống sẽ tìm 1 vùng RAM thật rãnh (có địa chỉ bắt đầu là base), nếu không có phải tìm cách giải phóng 1 vùng RAM thật ít gây phiền hà nhất (có địa chỉ bắt đầu là base), dựa vào thông tin trong field "inDisk" để mở file và đọc segment vào vùng RAM thật tìm được.
4. Hiệu chỉnh lại field inRAM = 1 và field start_addr = base rồi quay lại bước 2.

Quản lý bộ nhớ ảo phân đoạn

Đặc tính	Kỹ thuật Paging	Segementation
Người lập trình có biết kỹ thuật này đang dùng ?	Không	Có
Chương trình có bao nhiêu vùng địa chỉ tuyến tính độc lập ?	1	n
Kích thước không gian ảo tổng cộng có lớn hơn kích thước RAM ?	Có	Có
Hàm và dữ liệu được tách biệt và bảo vệ riêng biệt ?	Không	Có
Các bảng dữ liệu dễ thích ứng khi kích thước của chúng bị thay đổi ?	Không	Có
Các chương trình có thể dùng chung dữ liệu và hàm ?	Không	Có
Vì sao kỹ thuật này được phát sinh	hệ thống muốn chương trình có không gian rất lớn, không bận tâm kích thước RAM	Người lập trình mong muốn có nhiều vùng độc lập để chứa thông tin độc lập.

5.6 Quản lý bộ nhớ ảo phân đoạn và phân trang

Qui trình đổi địa chỉ ảo sang địa chỉ thật ở slide trước có khuyết điểm trong trường hợp quản lý segemnt có kích thước lớn : ta khó/không tìm được vùng RAM trống chứa nó. Vì lý do này, trong thực tế, người ta phải kết hợp 2 phương pháp quản lý phân trang và phân đoạn lại, đây là phương pháp mạnh nhất hiện nay. ý tưởng là hệ thống sẽ quản lý mỗi segment phần mềm như là 1 không gian ảo gồm nhiều trang ảo, mỗi lần chương trình truy xuất ô nhớ nằm trong trang ảo nào của segment nào, hệ thống sẽ tìm cách nạp nó vào RAM.

Quản lý bộ nhớ ảo phân đoạn và phân trang

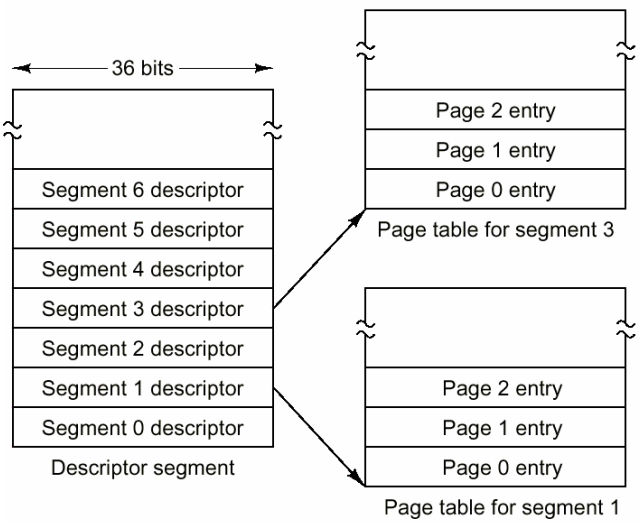
Qui trình đổi địa chỉ ảo sang địa chỉ thật :

1. từ địa chỉ mà chương trình truy xuất gồm 2 thành phần : segement (s) và offset, hệ thống sẽ tách offset ra thành 2 thành phần page (p) + offset1.
2. Truy xuất record quản lý segement s trong bảng đặc tả segement. Nếu field inRAM=1 thì bản đặc tả trang cho segement s đã có trong RAM. Nếu không thì tìm cách nạp nó vào RAM.
3. Truy xuất record quản lý trang ảo p trong bảng đặc tả trang. Nếu field inRAM=1 thì địa chỉ thật tương ứng là :

page frame	Offset

   và qui trình kết thúc.
4. Nếu inRAM =0, hệ thống sẽ tìm 1 trang thật rãnh (k), nếu không có phải tìm cách giải phóng 1 trang thật ít gây phiền hà nhất (k), dựa vào thông tin trong field "inDisk" để mở file và đọc trang ảo vào trang thật k.
5. Hiệu chỉnh lại field inRAM = 1 và field page frame = k rồi quay lại bước 3.

Quản lý bộ nhớ ảo phân đoạn và phân trang



5.7 Quản lý bộ nhớ của CPU Intel 80x86

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.

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 2^{16} segment, mỗi segment có 2^{16} byte $\rightarrow$ 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,

Quản lý bộ nhớ Real mode

Nguyên lý hoạt động (tt) :

- 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.

Quản lý bộ nhớ Protected mode

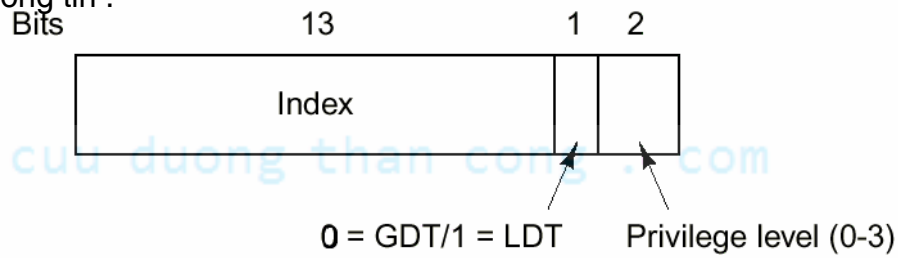
Nguyên lý hoạt động :

- 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 2^{16} segment, mỗi segment có 2^{32} byte $\rightarrow$ mỗi chương trình dài maximum 2^{48} = 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.

Quản lý bộ nhớ Protected mode

Nguyên lý hoạt động (tt) :

- 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 32-bit cho offset) $\rightarrow$ bộ nhớ chương trình tổng cộng là 64 TB.

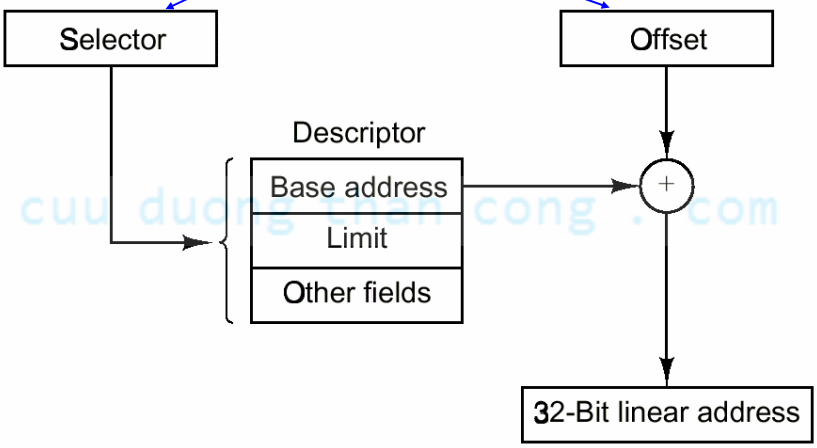
Cấu trúc record quản lý segment của Intel

- 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

<table>
<tr><td colspan="4">Limit 0-7</td></tr>
<tr><td colspan="4">Limit 8-15</td></tr>
<tr><td colspan="4">Base 0-7</td></tr>
<tr><td colspan="4">Base 8-15</td></tr>
<tr><td colspan="4">Base 16-23</td></tr>
<tr><td>P</td><td>DPL</td><td>S</td><td>Type</td></tr>
<tr><td>G</td><td>D</td><td></td><td>Limit 16-19</td></tr>
<tr><td colspan="4">Base 24-31</td></tr>
</table>

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 :



Quản lý bộ nhớ 368 enchanced mode

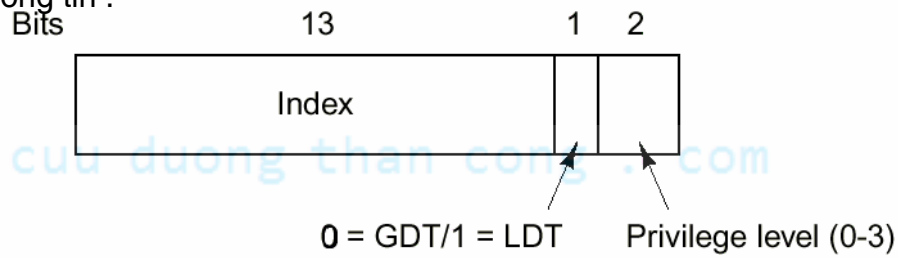
Nguyên lý hoạt động :

- 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 2^{16} segment, mỗi segment có 2^{32} byte $\rightarrow$ mỗi chương trình dài maximum 2^{48} = 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.

Quản lý bộ nhớ Protected mode

Nguyên lý hoạt động (tt) :

- 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 32-bit cho offset) $\rightarrow$ bộ nhớ chương trình tổng cộng là 64 TB.

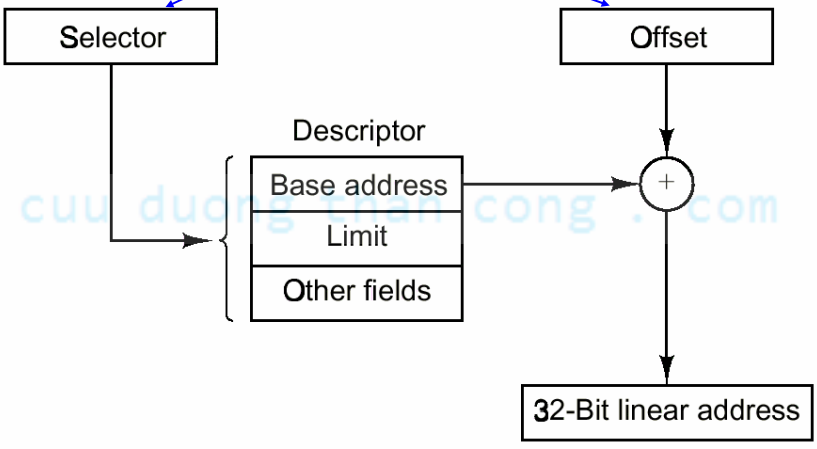
Cấu trúc record quản lý segment của Intel

- 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

<table>
<tr><td colspan="4">Limit 0-7</td></tr>
<tr><td colspan="4">Limit 8-15</td></tr>
<tr><td colspan="4">Base 0-7</td></tr>
<tr><td colspan="4">Base 8-15</td></tr>
<tr><td colspan="4">Base 16-23</td></tr>
<tr><td>P</td><td>DPL</td><td>S</td><td>Type</td></tr>
<tr><td>G</td><td>D</td><td></td><td>Limit 16-19</td></tr>
<tr><td colspan="4">Base 24-31</td></tr>
</table>

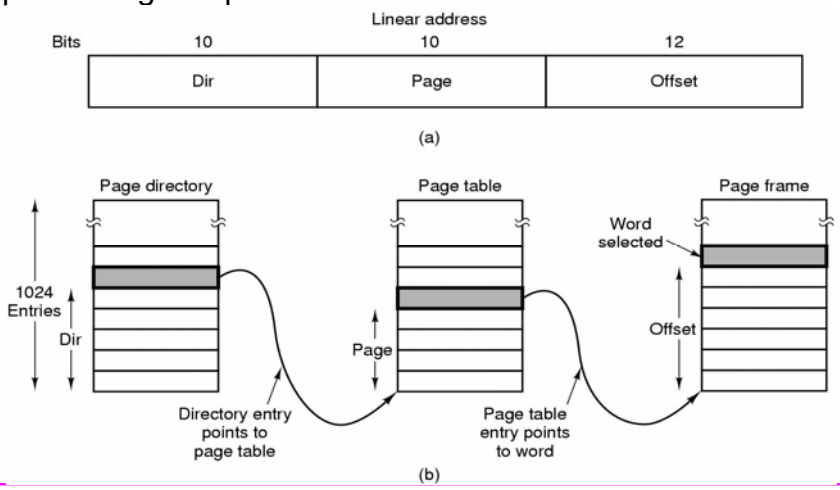
Qui trình đổi địa chỉ ảo sang địa chỉ thật

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



Qui trình đổi địa chỉ ảo sang địa chỉ thật

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



Các mức phân quyền bảo vệ segment

CPU 80x86 cung cấp 4 mức phân quyền từ 0 (cao nhất) tới 3 (thấp nhất). Mỗi segment có field DPL = mức phân quyền của mình.

Mỗi process có mức phân quyền : hệ thống có mức 0, còn application có mức 3. Hệ thống có thể truy xuất bất kỳ segment nào, còn application chỉ có thể truy xuất các segment có mức phân quyền bằng hay cao hơn mình.