**Bộ nhớ ảo (Virtual Memory)**

Trong bài học [Khái quát về tiến trình](https://vimentor.com/vi/lesson/khai-quat-ve-tien-trinh-1), chúng ta đã lướt qua các thành phần bộ nhớ (được gọi là các segment) cấu thành bộ nhớ của một tiến trình. Trong bài này, chúng ta cần hiểu rằng các thành phần và bộ nhớ của tiến trình được đề cập đó là bộ nhớ ảo (virtual memory). Vậy bộ nhớ ảo là gì, tại sao lại cần dùng đến nó và nó hoạt động như thế nào?

Giống như hầu hết các kernel hiện đại, Linux sử dụng 1 kỹ thuật được gọi là quản lý bộ nhớ ảo (virtual memory management) nhằm mục đích sử dụng hiệu quả cả CPU và RAM (bộ nhớ vật lý hay, cũng có thể được gọi là bộ nhớ thật). Kỹ thuật này khai thác 1 đặc điểm chung về truy cập bộ nhớ của hầu hết các chương trình là locality of reference (dịch nôm na là tham chiếu vùng), được biểu hiện qua 2 đặc tính sau:

* **Spatial locality: chương trình có xu hướng tham chiếu đến địa chỉ bộ nhớ gần với phân vùng bộ nhớ mà nó đang truy cập. Vì chương trình C xử lý các chỉ lệnh một cách tuần tự, hoặc chương trình xử lý các struct (vùng nhớ của các thành phần của struct được sắp xếp liền kề nhau)**
* **Temporal locality: chương trình có xu hướng truy cập cùng 1 vùng nhớ trong các thời điểm rất gần nhau. Ví dụ trường hợp xử lý vòng lặp, các vùng nhớ có thể được truy cập nhiều lần các thời điểm gần nhau mỗi khi vòng lặp quay lại.**

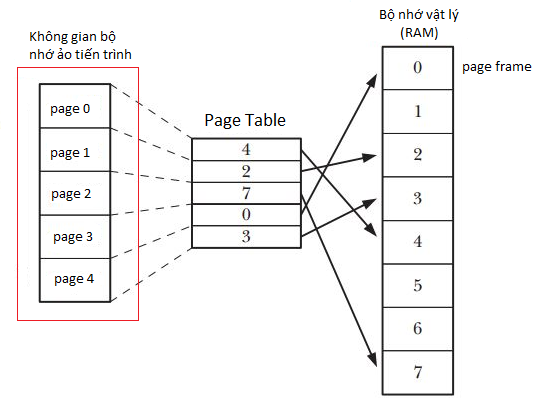
Nhìn vào đăc tính locality of reference này, chúng ta có thể rút ra là 1 chương trình có thể chạy được khi chỉ cần 1 phần không gian địa chỉ của nó trên RAM, thay vì tải toàn bộ không gian bộ nhớ của tiến trình.

Một bộ nhớ ảo chia không gian bộ nhớ của 1 tiến trình ra làm nhiều đoạn nhỏ có kích thước cố định được gọi là các trang (page). Tương tự, RAM cũng được chia làm nhiều đoạn nhỏ cùng kích thước được gọi là page frame. Trong 1 thời điểm, chỉ 1 vài trang của tiến trình cần có mặt trong các frame của RAM để chạy. Các trang chưa được sử dụng của chương trình sẽ được để trong phân vùng swap (là phân vùng dự trữ của ổ cứng hỗ trợ lưu trũ bổ sung cho RAM) và sẽ được tải vào RAM khi cần thiết.

**Page table**

Để ánh xạ giữa các trang của không gian bộ nhớ ảo đến các frame của bộ nhớ vật lý, kernel tạo ra 1 bảng trang (page table) cho mỗi tiến trình. Mỗi entry của page table ứng với 1 trang của bộ nhớ ảo cho phép chỉ ra vị trí của trang đó trong RAM hoặc chỉ ra nó đang nằm ở phân vùng swap của ổ cứng.

Để hình dung rõ hơn về page table, chúng ta xem hình dưới đây:



**Hình 1: Page Table của một tiến trình**

Trong thực tế, không cần phải xây dựng bảng ánh xạ cho tất cả không gian địa chỉ bộ nhớ ảo của tiến trình. Trái lại, thường chỉ có 1 phần số lượng các trang trong số đó được sử dụng và cần phải có bảng ánh xạ cho các trang đó. Vì vậy, về mặt lý thuyết không cần phải xây dựng entry cho toàn bộ không gian bộ nhớ ảo của tiến trình.

**Demand Paging**

Demand paging nghĩa là xây dựng bảng paging table và tải các page của tiến trình theo nhu cầu. Như đã nói ở phần trên, trong 1 thời điểm chỉ cần 1 phần không gian bộ nhớ của tiến trình được xây dựng bảng paging table và được tải vào RAM, các phần còn lại được đặt ở phân vùng swap. Vậy nếu tiến trình truy cập vào phân vùng hiện tại chưa có trong RAM hoặc thậm chí chưa được xây dựng bảng paging table thì sao?

Trong trường hợp này, CPU sẽ sinh ra 1 page fault, và kernel sau đó sẽ tải bộ nhớ mà tiến trình đang cần đó từ phân vùng swap vào RAM. Thông thường, page fault không phải là 1 lỗi nghiêm trọng như tên của nó, mà chỉ đơn giản là CPU biết rằng tiến trình sẽ bị tạm dừng thực thi cho đến khi kernel hoàn tất tải bộ nhớ nó đang cần vào RAM. Kỹ thuật này được gọi là demand paging.

Tuy nhiên, chắc chắn là bạn không muốn CPU phải ngồi không chờ kernel tải bộ nhớ từ swap vào RAM. Để tối ưu hóa xử lý, CPU sẽ chuyển sang thực thi tiến trình khác đang chờ và đã có bộ nhớ cần thiết trong RAM và sẽ quay lại thực thi tiến trình trước từ chính câu lệnh đã sinh ra page fault sau khi kernel tải xong bộ nhớ cần thiết.

Kỹ thuật quản lý bộ nhớ ảo đi cùng với demand paging đã tách biệt không gian địa chỉ ảo của tiến trình với không gian địa chỉ vật lý trên RAM. Kỹ thuật này mang lại một số ưu điểm sau:

* **Các tiến trình trong hệ thống chạy tách biệt nhau, và tách biệt khỏi kernel. Một tiến trình không thể truy cập hoặc thay đổi bộ nhớ của tiến trình khác hoặc của kenrel. Điều này thực hiện được bởi mỗi địa chỉ trong tiến trình được bảng page table trỏ đến 1 địa chỉ riêng biệt trên RAM (hoặc phân vùng swap)**
* **Các tiến trình có thể chia sẻ bộ nhớ nếu muốn (khi các tiến trình dùng chung text segment hoặc dùng shared memory). Bảng page table có thể ánh xạ địa chỉ bộ nhớ ảo của các tiến trình đến cùng 1 địa chỉ trên RAM.**
* **Cho phép triển khai cơ chế bảo vệ bộ nhớ. Mỗi entry của bảng page table có thể được đánh dấu để chỉ ra nội dung của page này cho phép chỉ đọc (readble), hay chỉ ghi (writeable) hoặc cả hai. Điều này cũng có nghĩa là nếu các page của các tiến trình đều ánh xạ đến cùng 1 địa chỉ vật lý, kernel có thể thiết lập quyền truy cập khác nhau trên một địa chỉ giữa các tiến trình khác nhau, tiến trình này chỉ có thể đọc, trong khi tiến trình khác có thể cả đọc và ghi.**
* **Lập trình viên và các tool như compiler hoặc linker không cần phải quan tâm đến việc tổ chức địa chỉ vật lý của tiến trình trong RAM.**
* **Vì chỉ cần 1 phần không gian địa chỉ của tiến trình cần có mặt trong RAM, chương trình sẽ được tải và chạy nhanh hơn. Điều này cũng cho phép hệ thống sử dụng dung lượng RAM nhỏ hơn so với không gian địa chỉ ảo của tiến trình. Ví dụ không gian địa chỉ ảo của 1 tiến trình trong hệ thống 32 bit là 4 GB, nhưng có thể chạy được trên hệ thống với chỉ 1GB RAM.**
* **Vì mỗi tiến trình sử dụng ít RAM hơn, nhiều tiến trình có thể cùng tồn tại trong RAM. Việc này làm tăng hiệu quả rất nhiều cho CPU, vì không phải chờ cho tiến trình được tải vào RAM để thực thi.**

**Kết luận**

Khái niệm về bộ nhớ ảo là một trong những kỹ thuật mạnh mẽ và không thể thiếu trong các hệ thống hiện đại về quản lý bộ nhớ. Vì vậy, khi nói đến bộ nhớ hoặc địa chỉ của của tiến trình, chúng ta cần nhớ là đang bàn về bộ nhớ ảo của nó. Trong bài học sau, chúng ta sẽ tìm hiểu các kỹ thuật cấp phát bộ nhớ khi lập trình trên hệ thống Linux.