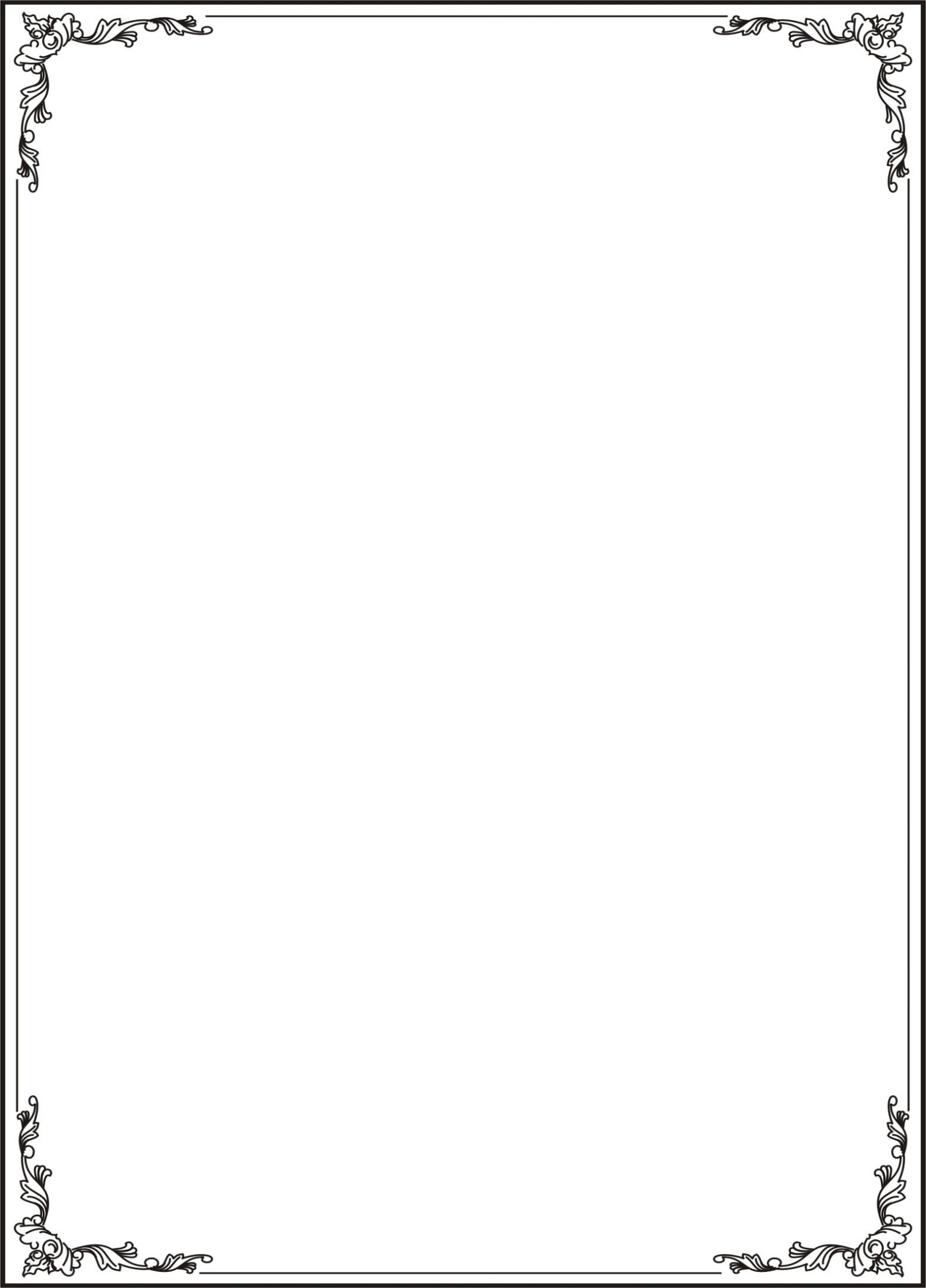
**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**



**KHOA CÔNG NGHỆ THÔNG TIN**

----------

**BÀI TẬP LỚN MÔN HỌC: OS**

**Tên đề tài: Nghiên cứu tìm hiểu về quản lý Bộ Nhớ Trong trong HĐH Linux.**

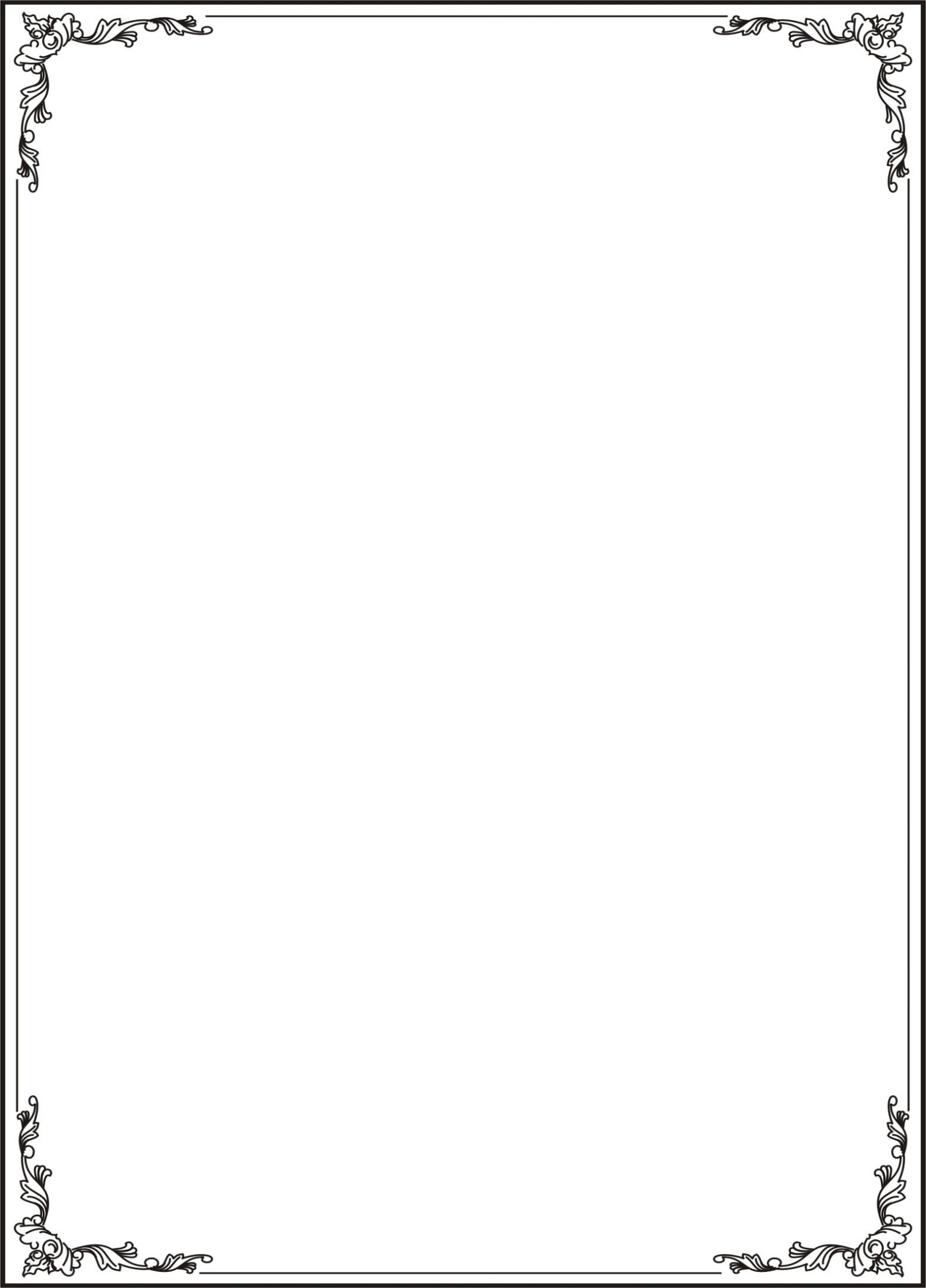
**Giáo viên:** ***Ths. Nguyễn Tuấn Tú***

**Nhóm thực hiện:** Nhóm 04

**Lớp:** IT6025.6(006)K15

Hà Nội, 2022

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**



**KHOA CÔNG NGHỆ THÔNG TIN**

----------

**BÀI TẬP LỚN MÔN HỌC: OS**

**Tên đề tài: Nghiên cứu tìm hiểu về quản lý Bộ Nhớ Trong trong HĐH Linux.**

**Giáo viên:** ***Ths. Nguyễn Tuấn Tú***

**Nhóm thực hiện:** Nhóm 04

**Sinh viên thực hiện:** 1. Phạm Tuấn Dũng

2. Nguyễn Xuân Duy

3. Nguyễn Xuân Thịnh

4. Phạm Đức Thắng

**Lớp:** IT6025.6(006)K15

Hà Nội, 2022

MỤC LỤC

[LỜI NÓI ĐẦU 5](#_Toc102678080)

[CHƯƠNG 1: GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH LINUX 6](#_Toc102678081)

[1.1. Hệ điều hành Linux 6](#_Toc102678082)

[1.2. Tổng quan về quản lí bộ nhớ trong Linux 6](#_Toc102678083)

[CHƯƠNG 2 : CƠ CHẾ QUẢN LÝ BỘ NHỚ CỦA LINUX 8](#_Toc102678084)

[2.1. Một mô hình tóm tắt về bộ nhớ ảo 9](#_Toc102678085)

[2.2. Lý do yêu cầu phải phân trang 11](#_Toc102678086)

[CHƯƠNG 3: VÙNG TRUNG GIAN (SWAPPING) 13](#_Toc102678087)

[CHƯƠNG 4: BỘ NHỚ ẢO DÙNG CHUNG (SHARED VIRTUAL MEMORY) 14](#_Toc102678088)

[CHƯƠNG 5: CACHES 16](#_Toc102678089)

[5.1. Vùng bộ đệm (Buffer Cache) 16](#_Toc102678095)

[5.2. Vùng trang nhớ (Page Cache) 16](#_Toc102678096)

[5.3. Vùng lưu trữ trung gian (Swap Cache) 17](#_Toc102678097)

[5.4. Các vùng đệm phần cứng (Hardware Caches) 17](#_Toc102678098)

[CHƯƠNG 6: CÁC BẢNG QUẢN LÝ TRANG TRONG LINUX 18](#_Toc102678099)

[6.1. Việc phân phối và thu hồi trang 18](#_Toc102678100)

[6.2. Việc phân phối trang 19](#_Toc102678101)

[6.3. Thu hồi trang đã phân phối 20](#_Toc102678102)

[CHƯƠNG 7: VẤN ĐỀ ÁNH XẠ BỘ NHỚ 22](#_Toc102678103)

[7.1. Yêu cầu đánh số trang 23](#_Toc102678104)

[7.2. Vùng trang đệm của Linux (The Linux Page Cache) 25](#_Toc102678105)

[7.3. Việc loại bỏ các trang và tráo đổi các trang ra ngoài 26](#_Toc102678106)

[7.3.1. Phương pháp giảm kích thước các trang và các vùng bộ đệm 27](#_Toc102678107)

[7.3.2. Phương pháp tráo đổi các trang nhớ dùng chung System V 29](#_Toc102678108)

[7.3.3. Phương pháp tráo đổi và loại bỏ các trang 30](#_Toc102678109)

[7.4. Việc tráo đổi các trang đã nạp 32](#_Toc102678110)

[Kết luận 33](#_Toc102678111)

# LỜI NÓI ĐẦU

**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 hệ điều hành **Linux**.

# CHƯƠNG 1: GIỚI THIỆU VỀ HỆ ĐIỀU HÀNH LINUX

## 1.1. 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 quản lí 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.

# CHƯƠNG 2 : 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.

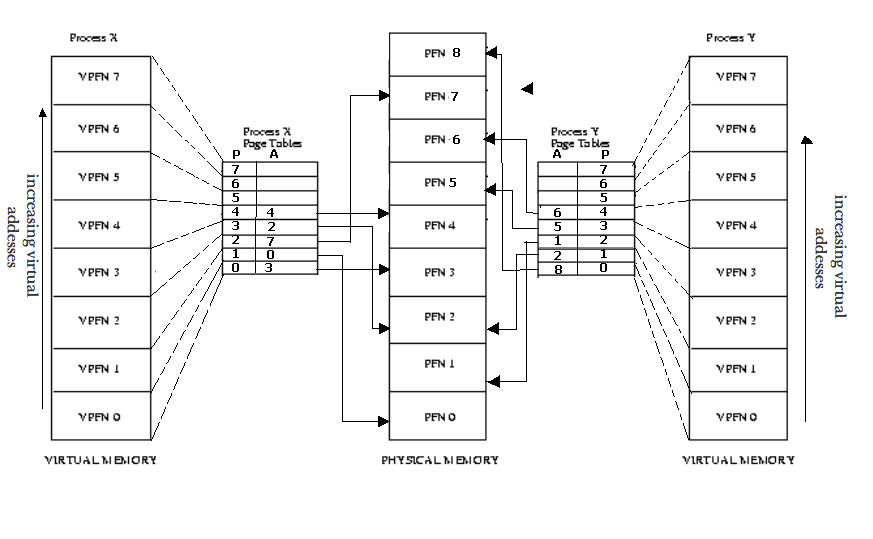
**Phân phối bộ nhớ vật lý hợp lý** (Fair Physical Memory Allocation)

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.

## 2.1. Một mô hình tóm tắt về bộ nhớ ảo



***Hình 2.1****: 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, theo 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.

## 2.2. Lý do yêu cầu phải phân trang

Khi có ít không gian nhớ vật lý hơn không gian nhớ ảo, hệ điều hành phải cân nhắc để sử dụng hiệu quả bộ nhớ. Một phương pháp để tiết kiệm bộ nhớ vật lý là **yêu cầu trang.** Theo đó hệ thốngchỉ nạp các trang bộ nhớ ảo được truy cập mà đang được sử dụng bởi chương trình hiện thi hành.

Khi tiến trình cố gắng truy cập tới địa chỉ ảo không hiện thời trong bộ nhớ, hay bộ vi xử lý không thể tìm thấy phần tử trong bảng phân trang ứng với trang ảo đã tham chiếu. VD trong hình 1 không có phần tử nào trong bảng phân trang ứng với số hiệu khung trang ảo 2 nên khi tiến trình X cố gắng đọc một địa chỉ từ số hiệu khung trang ảo này, bộ vi xử lý không thể chuyển đổi địa chỉ này thành một địa chỉ vật lý. Lúc này bộ vi xử lý thông báo với hệ điều hành xuất hiện một lỗi trang.

\*) Nếu địa chỉ ảo lỗi là địa chỉ ***không hợp lệ*** (hay không tồn tại), hệ điều hành sẽ kết thúc nó để bảo vệ các tiến trình khác trong hệ thống không bị ảnh hưởng bởi tiến trình không minh bạch này.

\*) Nếu địa chỉ ảo lỗi là địa chỉ hợp lệ nhưng trang mà nó đưa ra không có trong bộ nhớ, hệ điều hành phải nạp trang phù hợp vào bộ nhớ từ trên đĩa. Việc truy cập đĩa mất nhiều thời gian vì vậy tiến trình phải đợi cho đến khi trang được nạp. Sau khi trang ảo đã nạp được ghi vào khung trang vật lý còn trống và một phần tử trong bảng phân trang có chỉ số giống số hiệu khung trang ảo đó sẽ nhận số hiệu trang vật lý vừa được phân phối cho nó. Sau đó tiến trình được chuyển trở lại trạng thái hoạt động để thực hiện tiếp lệnh bị tạm dừng khi lỗi bộ nhớ xuất hiện. Việc truy cập bộ nhớ ảo được tiếp tục thực hiện, bộ vi xử lý thực hiện việc chuyển đổi địa chỉ ảo thành địa chỉ vật lý, tiến trình tiếp tục được thi hành.

*\* Trong thời gian chờ nạp trang nếu có một tiến trình khác có thể chạy thì hệ điều hành sẽ lựa chọn một trong số chúng để cho thi hành.*

Linux sử dụng kỹ thuật yêu cầu trang để nạp các trang có thể thi hành vào trong bộ nhớ ảo. Bất cứ khi nào một lệnh được thi hành, file đang chứa nó được mở và nội dung của nó được ánh xạ vào bộ nhớ ảo. Điều này được thực hiện bởi sơ đồ bộ nhớ (*memory mapping*). Tuy nhiên, chỉ có phần đầu tiên của trang được đưa vào bộ nhớ vật lý. Khi thi hành lệnh, hệ thống sinh ra lỗi trang và Linux sẽ sử dụng sơ đồ bộ để quyết định những phần nào của nó được nạp để thi hành.

# CHƯƠNG 3: 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.

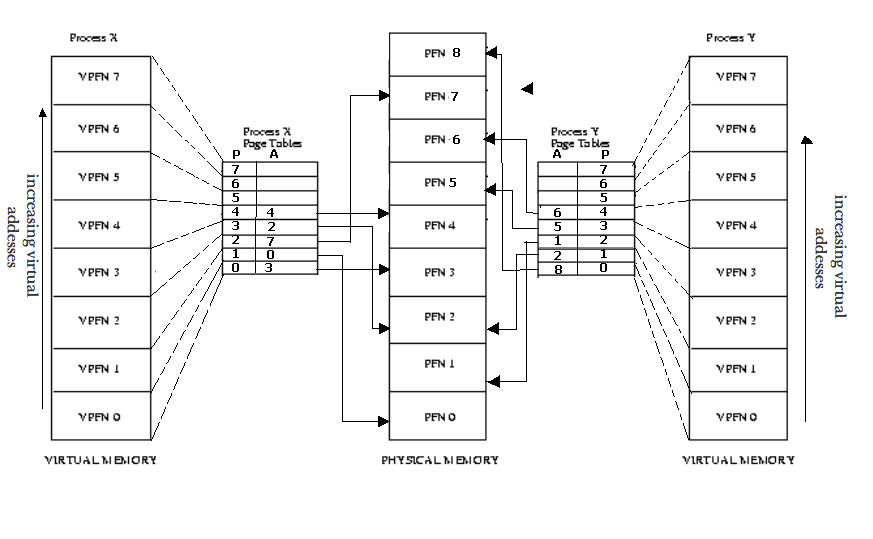
Tuy nhiên, nếu trang đã bị thay đổi, hệ điều hành phải giữ nội dung của trang đó để truy cập về sau. Kiểu của trang này gọi là trang ***dirty***, khi nó bị di chuyển khỏi bộ nhớ sẽ được ghi vào một loại file đặc biệt là swap file (file trao đổi). Việc truy cập tới swap file liên quan rất nhiều tới tốc 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.

# CHƯƠNG 4: 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.

**\*) Các kiểu địa chỉ vật lý và địa chỉ ảo**

Hệ điều hành luôn hạn chế sử dụng bộ nhớ ảo, vì việc duy trì các bảng phân trang trong thời gian dài tốn lượng tài nguyên máy rất lớn dù đa số các vi xử lý hiện nay đều hỗ trợ xử lí các kiểu địa chỉ vật lý và kiểu địa chỉ ảo.

Lấy ví dụ bộ vi xử lý Alpha AXP không có kiểu địa chỉ vật lý riêng biệt. Thay vào đó, nó chia không gian nhớ thành một số vùng và chỉ định 2 trong số chúng để lưu trữ các địa chỉ ảo đã được ánh xạ vật lý. Không gian địa chỉ này được gọi là không gian địa chỉ KSEG và nó bao gồm tất cả các địa chỉ tính từ *0xfffffc0000000000* đến cao hơn. Để thi hành mã đã liên kết trong KSEG hoặc truy cập dữ liệu ở đó, mã phải được thi hành ở chế độ hạt nhân (*Trong tài liệu sẽ không đi sâu vào tìm hiểu mục này*). Hạt nhân Linux trên Alpha sẽ thi hành từ địa chỉ *0xfffffc0000310000*.

# CHƯƠNG 5: 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)

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

Các bộ đệm có dung lượng đã ấn định trước (ví dụ 512 bytes) và chứa các khối thông tin được đọc từ một thiết bị khối hoặc sẽ được ghi tới thiết bị khối. Một thiết bị khối là thiết bị chỉ có thể được truy cập bởi việc đọc hoặc ghi các khối dữ liệu có kích thước đã quy ước từ trước. Ví dụ như tất cả các ổ đĩa đều là các thiết bị khối.

Vùng bộ đệm được đánh số thông qua việc định danh thiết bị và số hiệu khối được sử dụng để tìm kiếm một khối dữ liệu. Nếu dữ liệu có thể được tìm trong vùng bộ đệm thì nó không cần phải đọc từ thiết bị khối vật lý, ví dụ một đĩa cứng sẽ truy cập dữ liệu nhanh hơn do việc truy cập ổ đĩa vật lý tốn rất nhiều thời gian.

## Vùng trang nhớ (Page Cache)

Vùng trang nhớ được sử dụng để tăng tốc độ truy cập các trang và dữ liệu trên đĩa.

Một trang nhớ được sử dụng để lưu trữ các nội dung logic của một file và trang được truy cập thông qua file và địa chỉ offset trong file. Khi các trang được đọc từ đĩa để ghi vào bộ nhớ, chúng sẽ được ghi vào vùng trang nhớ.

## 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).

Nếu các trang không bị thay đổi sau khi từng được ghi tới một file trung gian trước đó thì trong lần truy cập tiếp theo không cần phải ghi lại trang này vào file trung gian. Thay vào đó nó có thể bị loại bỏ, điều này tiết kiệm cho ta nhiều thao tác không cần thiết với ổ đĩa.

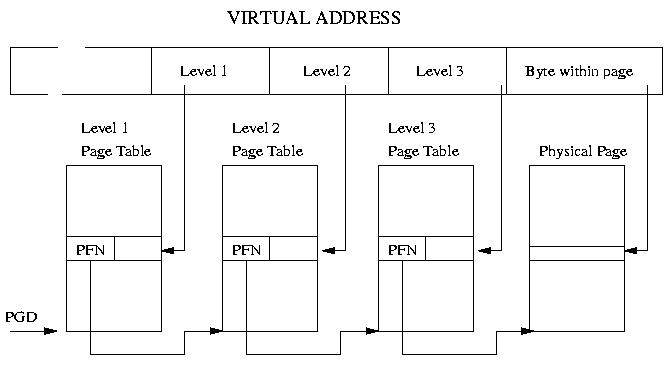
## Các vùng đệm phần cứng (Hardware Caches)

Thông thường một vùng đệm phần cứng được đặt trong bộ vi xử lý, tại nơi lưu giữ các phần tử bảng quản lý trang. Trường hợp này bộ vi xử lý không đọc các trang trực tiếp từ bảng quản lý trang mà sẽ chuyển đổi giá trị trong bảng quản lý trang này để có được địa chỉ các trang mà nó cần. Hệ thống sẽ sử dụng các phần tử gọi là Translation Look-aside Buffers (TLB) để chứa các bản sao của các phần tử bảng quản lý trang từ một hoặc nhiều tiến trình trong hệ thống.

Khi địa chỉ ảo được xử lý, bộ vi xử lý sẽ cố gắng tìm một phần tử TLB phù hợp cho nó. Nếu nó tìm được phần tử phù hợp, nó có thể chuyển đổi trực tiếp địa chỉ ảo thành một địa chỉ vật lý và thực hiện thao tác dữ liệu. Nếu không thì nó phải gửi một tín hiệu thông báo một lỗi LTB tới hệ điều hành. Lỗi này sẽ được chuyển tới hệ điều hành để sửa chữa: hệ điều hành sẽ đưa ra một phần tử TLB mới để ánh xạ địa chỉ, phần tử lỗi bị xóa, bộ vi xử lý sẽ chuyển đổi địa chỉ ảo sang địa chỉ vật lý. Lúc này nó sẽ thực hiện thao tác đọc/ghi dữ liệu với bộ nhớ.

Hạn chế của việc sử dụng các vùng đệm là Linux phải sử dụng nhiều lần và nhiều không gian nhớ cho việc duy trì các vùng đệm này và nếu các vùng đệm này bị lỗi, hệ thống sẽ bị treo.

# CHƯƠNG 6: CÁC BẢNG QUẢN LÝ TRANG TRONG LINUX



***Hình 6.1:*** *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.

## 6.1. 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.

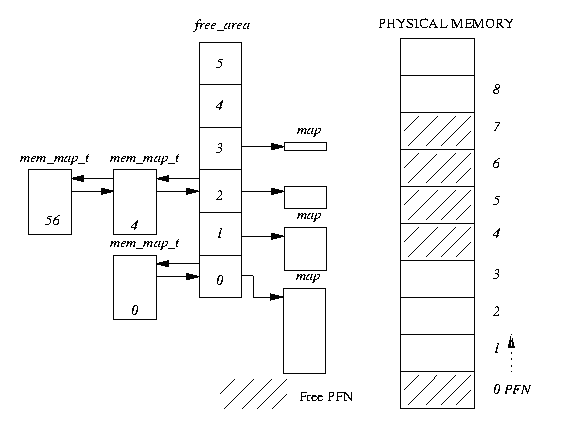
## 6.2. Việc phân phối trang

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 6.2****: 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*.

## 6.3. Thu hồi trang đã phân phối

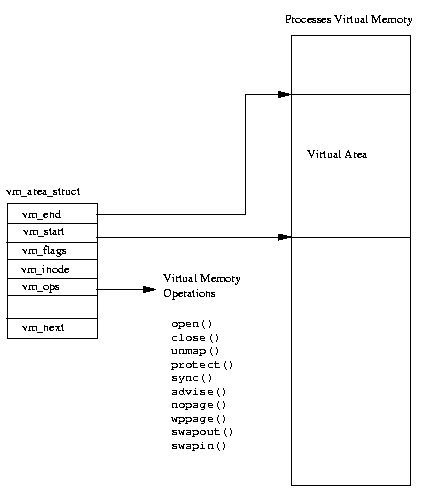
Việc phân phối các khối nhớ gồm nhiều trang dẫn đến sự phân mảnh bộ nhớ do các khối nhớ tự do có số trang lớn hơn yêu cầu phải tách thành các khối nhỏ hơn. Mã lệnh thu hồi trang sẽ gộp các khối có số trang nhỏ thành các khối có số trang tự do lớn hơn bất cứ khi nào có thể.

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.

# CHƯƠNG 7: 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 7.1****: 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.

## 7.1. Yêu cầu đánh số trang

Một khi file có khả năng thi hành đã được ánh xạ vào một bộ nhớ ảo của tiến trình, nó có thể được thi hành. Ngay khi file vừa được nạp vào bộ nhớ, nó sẽ truy cập một vùng bộ nhớ ảo không thuộc bộ nhớ vật lý. Khi tiến trình truy cập một địa chỉ ảo mà không có một phần tử trong bảng quản lý trang hợp lệ, bộ vi xử lý sẽ báo với Linux rằng có một lỗi trang.

Linux phải tìm cấu trúc dữ liệu *vm\_area\_struct* diễn tả vùng nhớ xuất hiện lỗi trang. Nếu không có cấu trúc dữ liệu *vm\_area\_struct* chứa địa chỉ ảo đó, tiến trình này được xác nhận đã truy cập một địa chỉ không hợp lệ. Linux chuyển một tín hiệu tới hệ điều hành, nếu tiến trình không có phản hồi hợp lệ, nó sẽ bị kết thúc.

Tiếp theo Linux kiểm tra kiểu lỗi trang đã xảy ra có đúng với các kiểu truy cập được phép đối với vùng nhớ ảo không. Nếu tiến trình đang truy cập theo một phương pháp không hợp lệ, như là ghi tới một vùng mà nó chỉ được phép đọc, nó cũng gây ra một lỗi bộ nhớ.

Nếu Linux đã xác định lỗi trang là lỗi hợp lệ, nó sẽ bắt đầu xử lý lỗi.

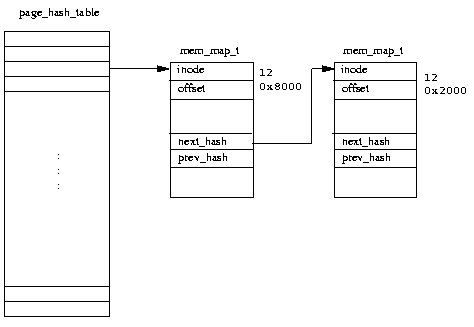
Nếu phần tử trong bảng phân trang không hợp lệ nhưng không rỗng, do hiện thời một trang do phần tử này quản lý đang được lưu trữ trong swap file, đó là các phần tử có bit **valid** bằng 1 nhưng trường PFN có giá trị khác 0. Trong trường hợp này trường PFN sẽ chứa thông tin về vị trí lưu trữ trang trong swap file.

Không phải tất cả cấu trúc dữ liệu *vm\_area\_struct* đều có một tập các thao tác bộ nhớ ảo, thậm chí các cấu trúc dữ liệu đó có thể không có thao tác *nopage*… Nếu có một thao tác *nopage* cho vùng bộ nhớ ảo này, Linux sẽ sử dụng nó. Nếu không, một trang nhớ vật lý mới và một phần tử bảng phân trang hợp lệ sẽ được tạo ra cho nó

Thao tác *nopage* của Linux được sử dụng cho các file có khả năng thi hành đã ánh xạ bộ nhớ và nó sử dụng các vùng trang đệm để nạp trang yêu cầu vào bộ nhớ vật lý.

Bất cứ lúc nào trang được yêu cầu đã đưa vào bộ nhớ vật lý, các bảng quản lý trang cần được cập nhật, đặc biệt khi bộ vi xử lý sử dụng việc chuyển đổi địa chỉ để duyệt các bộ đệm dự trữ. Lúc này đã xác định được nguyên nhân lỗi trang và xử lý lỗi, tiến trình được kích hoạt lại và thực hiện tiếp lệnh đang chạy lúc bộ nhớ ảo bị lỗi.

## 7.2. Vùng trang đệm của Linux (The Linux Page Cache)



***Hình 7.2****: 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.

## 7.3. Việc loại bỏ các trang và tráo đổi các trang ra ngoài

Khi bộ nhớ vật lý trở nên ít đi, hệ thống con quản lý bộ nhớ của Linux phải cố gắng giải phóng các trang nhớ vật lý đang lưu trữ dữ liệu. Công việc này thực hiện bởi trình tiện ích tráo đổi của nhân Linux (*kswapd*)

Trình tiện ích tráo đổi của hạt nhân là một kiểu đặc biệt của tiến trình, một đoạn mã trong nhân HĐH. Các đoạn mã hạt nhân là các tiến trình không có bộ nhớ ảo, chúng chạy ở chế độ hạt nhân trong không gian địa chỉ vật lý. Trình tiện ích này chỉ đưa các trang cần tráo đổi ra swap file của hệ thống. Vai trò của nó là đảm bảo có đủ các trang tự do trong hệ thống có đủ để giúp cho hệ điều hành thực hiện quản lý và sử dụng bộ nhớ hiệu quả.

*Kswap* được khởi tạo tại thời điểm khởi động hệ thống và thực hiện việc tráo đổi trang theo định kỳ nhờ bộ swap timer của hạt nhân.

Tại mỗi thời điểm số đếm thời gian chạy đến không, *kswapd* xem xét xem số các trang nhớ tự do trong hệ thống để quyết định giải phóng một số trang về trạng thái tự do hay không. Chừng nào mà số trang nhớ tự do trong hệ thống duy trì vẫn trong phạm vi cho phép, *kswapd* sẽkhông làm gì cả. Nó trở về trạng thái sẵn sàng cho đến thời điểm số đếm thời gian về không tiếp theo.

Để kiểm tra thì *kswapd* sẽ nhận được giá trị tổng số trang hiện thời đang được ghi ra swap file. Nó lưu trữ giá trị tổng này lại, giá trị này được tăng tại mỗi thời điểm một trang được xếp hàng đợi để ghi ra swap file, và giảm tại mỗi thời điểm việc ghi tới thiết bị swap (ví dụ ổ cứng chứa swap file) hoàn thành. Nếu số trang tự do trong hệ thống thấp hơn mức cho phép, *kswapd* sẽ cố gắng giảm số trang vật lý đang được sử dụng bởi hệ thống theo 3 cách sau:

* Giảm kích thước của bộ đệm và các vùng trang đệm.
* Sử dụng các trang nhớ dùng chung
* Loại bỏ và tráo đổi các trang nhớ.

Nếu số các trang nhớ tự do trong hệ thống thấp HĐH sẽ cố gắng giải phóng 6 trang nhớ trước khi nó chạy lần tiếp theo. Nếu không thì nó sẽ cố gắng giải phóng 3 trang. Mỗi một cách ở trên được lần lượt cố gắng thực hiện cho đến khi có đủ số trang nhớ tự do cần thiết. *Kswap* sẽnhớ cách mà nó đã sử dụng sau cùng, mỗi lần chạy nó sẽ cố gắng thử giải phóng các trang nhớ bằng việc sử dụng phương pháp sau cùng này.

Sau khi nó có đủ các trang nhớ tự do, *kswapd* một lần nữa quay về trạng thái sẵn sàng cho đến thời điểm mà số đếm thời gian về không. Nếu vì một nguyên nhân nào đó, số trang tự do trong hệ thống vẫn quá thấp thì nó chỉ ở trạng thái sẵn sàng trong 1 khoảng thời gian bằng một nửa khoảng thời gian trạng thái sẵn sàng thông thường của nó. Một khi số các trang tự do lớn hơn nó sẽ trở lại giá trị bình thường.

### 7.3.1. Phương pháp giảm kích thước các trang và các vùng bộ đệm

Các trang được lưu giữ trong các vùng trang đệm (page caches) và các vùng bộ đệm (buffer caches) là đối tượng phù hợp để trở thành các trang tự do trong *free\_area vector.* Vùng trang đệm chứa các trang của các file đã ánh xạ bộ nhớ có thể chứa cả các trang không còn cần thiết nữa và các trang này sẽ làm đầy bộ nhớ của hệ thống. Tương tự như vậy, các vùng bộ đệm cho việc đọc hoặc ghi dữ liệu tới thiết bị vật lý, cũng có thể chứa các bộ đệm dữ liệu không còn cần thiết.

Khi bộ nhớ vật lý trong hệ thống bắt đầu chạy trong trạng thái đầy, việc loại bỏ các trang ra khỏi các vùng đệm này tương đối dễ do nó không yêu cầu ghi tới các thiết bị vật lý (Không giống như các trang được lưu trữ tạm ra ngoài ổ đĩa) Việc loại bỏ này có ưu điểm là không phải truy cập tới thiết bị vật lý và các file đã ánh xạ bộ nhớ (2 việc này làm cho hệ thống chạy chậm hơn). Tuy việc loại bỏ các trang khỏi các vùng đệm này cũng làm cho tất cả các tiến trình trở nên kém hiệu quả.

*kswapd* luôn cố gắng làm giảm kích thước các vùng đệm đã sử dụng này. Nó kiểm tra các trang của một khối nhớ trong *mem\_map page vecto.* Để chọn các trang có thể giải phóng khỏi bộ nhớ vật lý. Kích thước của khối tính theo trang khá lớn nếu *kswapd* thực hiện việc tráo đổi mạnh (trường hợp số lượng các trang tự do trong hệ thống đã thấp đến mức nguy hiểm).

Các khối gồm nhiều trang được kiểm tra theo kiểu xoay vòng, mỗi lần các trang trong một khối khác nhau được kiểm tra và tối thiểu hóa việc ánh xạ bộ nhớ. Thuật toán sử dụng gọi là thuật toán *clock*. Tại cùng một thời điểm, một vài trang trong toàn bộ *mem\_map page vector* sẽ cùng được kiểm tra.

Nếu trang đã từng được kiểm tra sẽ được kiểm tra lại nếu nó được lưu trữ trong vùng trang đệm hoặc vùng bộ đệm. Nếu trang không thuộc cả 2 vùng đệm thì trang tiếp theo trong *mem\_map page vector* sẽ được kiểm tra.

Nếu tất cả các bộ đệm đã được giải phóng thì các trang chứa chúng cũng được giải phóng. Các trang đã được kiểm tra nếu thuộc vùng trang đệm của Linux sẽ được loại bỏ khỏi vùng trang đệm và trở thành các trang tự do.

Khi có đủ các trang tự do thì *kswapd* sẽ đợi cho đến lần chạy tiếp theo để kích hoạt trở lại. Khi không có trang nào được giải phóng và là một phần của bộ nhớ ảo (chúng là các trang đã được cất giữ) thì sẽ không cập nhật bảng quản lý trang. Nếu không có đủ các trang đã lưu trữ dữ liệu để loại bỏ thì *nó* sẽ cố gắng tráo đổi một vài trang dùng chung.

### 7.3.2. Phương pháp tráo đổi các trang nhớ dùng chung System V

Bộ nhớ dùng chung ***System V*** là một kỹ thuật liên lạc giữa các tiến trình bên trong cho phép 2 hoặc nhiều tiến trình dùng chung bộ nhớ ảo để chuyển giao thông tin giữa chúng với nhau.

Mỗi vùng của bộ nhớ dùng chung ***System V*** được diễn tả bởi một cấu trúc dữ liệu *shmid\_ds*. Cấu trúc này chứa một con trỏ trỏ tới danh sách của các cấu trúc dữ liệu *vm\_area\_struct* ứng với tiến trình đang chia sẻ vùng nhớ này trong bộ nhớ ảo. Các cấu trúc dữ liệu *vm\_area\_struct* chứa vị trí vùng bộ nhớ chia sẻ system V của bộ nhớ ảo tiến trình.

Mỗi một cấu trúc dữ liệu *vm\_area\_struct* cho bộ nhớ chia sẻ system V này được liên kết với một cấu trúc dữ liệu *vm\_area\_struct* khác bằng việc sử dụng các con trỏ *vm\_next\_shared* và *vm\_prev\_shared*. Cấu trúc dữ liệu *shmid\_ds* cũng chứa một danh sách các phần tử trong bảng quản lý trang diễn tả trang nhớ vật lý được ánh xạ bởi trang nhớ ảo dùng.

*Kswap* sử dụng thuật toán *clock* khi tráo đổi các trang nhớ dùng chung system V. Nó ghi nhớ các trang thuộc vùng bộ nhớ ảo dùng chung mà nó đã tráo đổi gần đây nhất bằng cách giữ 2 chỉ số, chỉ số đầu tiên là một chỉ số so sánh trong tập các cấu trúc dữ liệu *shmid\_ds*, chỉ số thứ 2 là một chỉ số trong danh sách các phần tử trong bảng quản lý trang cho vùng bộ nhớ chia sẻ system V này.

Khi số hiệu khung trang vật lý tương ứng với trang nhớ ảo của bộ nhớ chia sẻ (số hiệu này có trong các bảng phân trang của tất cả các tiến trình đang chia sẻ vùng nhớ ảo này), *kswapd* phải thay đổi tất cả các bảng phân trang đó để chỉ ra trang nhớ này không còn trong bộ nhớ nữa và lưu lại trong swap file. Đối với mỗi trang dùng chung đang được tráo đổi, *kswapd* tìm một phần tử trong bảng quản lý trang (chứa trong tập các bảng quản lý trang của các tiến trình đang chia sẻ). Nếu phần tử hợp lệ, nó chuyển thành không hợp lệ và giảm tổng số người sử dụng trang này (giá trị count) đi 1.

Nếu giá trị count của trang bằng 0 sau khi tất cả các bảng quản lý trang của các tiến trình đang chia sẻ bị thay đổi, trang dùng chung đó có thể được ghi ra swap file. Phần tử trong bảng quản lý trang này ứng với cấu trúc dữ liệu *shmid\_ds* cho vùng này của bộ nhớ chia sẻ system V sẽ được thay bằng phần tử quản lý trang đã tráo đổi ra ngoài. Thông tin phần tử trong bảng quản lý trang đã tráo đổi ra ngoài sẽ được sử dụng lại khi trang được đưa trở lại vào trong bộ nhớ vật lý.

### 7.3.3. Phương pháp tráo đổi và loại bỏ các trang

Trình tiện ích tráo đổi sẽ xem xét từng tiến trình trong hệ thống một cách lần lượt để tìm đối tượng phù hợp để tráo đổi ra ngoài.

Các đối tượng tốt là các tiến trình có thể được tráo đổi (có một số tiến trình không thể) và đối tượng đó có một hoặc nhiều trang có thể được tráo đổi hoặc loại bỏ khỏi bộ nhớ. Khi dữ liệu trong chúng không thể lấy lại, các trang được tráo đổi ra ngoài bộ nhớ vật lý sẽ được ghi vào các swap file của hệ thống chỉ.

Nhiều nội dung của một file có khả năng thi hành được nạp và dễ dàng được đọc lại từ file đó. Ví dụ, các lệnh có khả năng thi hành sẽ không bao giờ bị thay đổi và cũng sẽ không bao giờ được ghi tới swap file. Khi chúng được tham chiếu lại bởi tiến trình, chúng sẽ được đưa trở lại bộ nhớ từ file.

\* Các trang sẽ không thể bị tráo đổi hoặc loại bỏ nếu chúng bị khóa trong bộ nhớ.

Thuật toán tráo đổi trong Linux sử dụng các trang “già”. Mỗi một trang có 1 giá trị truyền vào *kswapd* để *kswapd* biết trang này có thuộc diện bị tráo đổi hay không. Độ già của trang tuỳ thuộc vào việc truy cập và sử dụng trang, *kswapd* sẽchỉ tráo đổi các trang già nhất.

Các trang kiểu *Dirty* là các trang có thể được tráo đổi ra ngoài. Linux sử dụng một bit xác định cấu trúc trong PTE để diễn tả các trang có kiểu *dirty*. Tuy không phải tất cả các trang *dirty* đều được ghi tới swap file do mỗi vùng bộ nhớ ảo của một tiến trình có thể có thao tác tráo đổi và phương thức sử dụng riêng. Mặt khác, trình tiện ích tráo đổi sẽ chỉ định một trang trong swap file để ghi trang tới thiết bị tráo đổi.

Phần tử trong bảng quản lý trang được thay thế bởi một phần tử mà nó được đánh dấu là invalid chứa thông tin về vị trí của trang trong swap file. Đó là địa chỉ offset của swap file chỉ ra nơi cất trang và một dấu hiệu thông báo swap file đang sử dụng.

Nếu đã đủ số trang của các tiến trình có thể tráo đổi, các trang được tráo đổi ra ngoài hay trang bị loại bỏ, trình tiện ích tráo đổi quay lại trạng thái sẵn sàng. Nó sẽ xem xét tiến trình tiếp theo trong hệ thống. Trình tiện ích tráo đổi luôn thực hiện tráo đổi trang vật lý tiến trình cho đến khi hệ thống trở lại trạng thái cân bằng. Phương pháp này tối ưu hơn nhiều với việc tráo đổi ra ngoài toàn bộ các tiến trình.

Khi các trang đang tráo đổi ra ngoài các swap file, Linux tránh việc cố gắng ghi các trang nếu nó không thể, do việc này có thể gây ra hiện tượng một trang vừa ở trong swap file và vừa ở trong bộ nhớ vật lý. Điều này xảy ra khi một trang đã được tráo đổi ra ngoài khỏi bộ nhớ lại được nạp lại vào bộ nhớ khi nó được truy cập bởi một tiến trình khác.

Linux sử dụng các vùng đệm tráo đổi để kiểm soát các file này. Vùng đệm tráo đổi là một danh sách các phần tử bảng quản lý trang, mỗi phần tử tương ứng với một trang vật lý trong hệ thống. Nếu một phần tử vùng đệm tráo đổi khác 0, nó sẽ chỉ ra một trang đang được lưu giữ trong một swap file chưa bị thay đổi. Nếu sau đó trang này bị thay đổi (bị ghi), phần tử quản lý nó sẽ bị đẩy khỏi vùng đệm tráo đổi.

Các phần tử trong vùng đệm tráo đổi là các phần tử thuộc bảng quản lý trang cho các trang đã bị tráo đổi ra ngoài. Chúng chứa thông tin cho phép Linux tìm ra swap file và trang trong swap file một cách chính xác.

## 7.4. Việc tráo đổi các trang đã nạp

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.

# Kết luận

Qua phần bài tập được trình bày trên tài liệu này, ta đã nắm được các nguyên lí trong cơ chế quản lý bộ nhớ của hệ điều hành Linux ở một mức độ căn bản. Từ đó giúp hiểu được một phần cách vận hành hoạt động hệ thống của hệ điều hành Linux. Tài liệu đã mang đến 1 phần kiến thức tuy nhỏ nhưng khá đầy đủ và căn bản có thể giúp em tự tin khi nghiên cứu sâu hơn trong vấn đề quản lý bộ nhớ không chỉ của hệ điều hành Linux mà còn cả các hệ điều hành khác.

Tài liệu này chủ yếu nghiên cứu về hệ thống bộ nhớ ảo, một kỹ thuật quản lý bộ nhớ rất hiệu quả và thông dụng không chỉ áp dụng riêng với Linux. Phần bài tập này chỉ mang tính chất nghiên cứu cơ bản nên không trình bày sâu nhưng em đã cố gắng tóm lược lại các vấn đề chính để hiểu được cách hoạt động của bộ nhớ trong hệ thống sử dụng hệ điều hành Linux như: địa chỉ ảo, địa chỉ vật lý, các loại bảng phân trang, bộ đệm, vùng đệm cache, ... Qua đó thấy được tầm quan trọng của việc quản lý bộ nhớ và sử dụng hiệu quả tài nguyên hệ thống một cách hợp lý.

Em xin chân thành cảm ơn!

**Tài liệu tham khảo trong bài:**

-Homepage:

-[https://www.kernel.org](https://www.kernel.org/)

-[Understanding The Linux Virtual Memory Manager](https://www.kernel.org/doc/gorman/pdf/understand.pdf)

-<http://www.tldp.org/LDP/tlk/mm/memory.html>

-Giáo trình hệ điều hành Linux và Unix