Quản lý bộ nhớ trong hệ điều hành Linux

Tạ Hữu Bình, Trần Trọng Hiệp, Lê Huy Hoàng, Võ Thục Khánh Huyền

Ngày 5 tháng 6 năm 2021

Tóm tắt nội dung

Giống như các hệ điều hành khác, Linux phải có các phương pháp khai thác và sử dụng tài nguyên một cách hiệu quả, đặc biệt là tài nguyên bộ nhớ. Bài báo cáo này sẽ trình bày về hệ thống bộ nhớ cùng với các chức năng của từng thành phần và nêu ra các chiến lược quản lý bộ nhớ trong Linux. Trong đó bài báo cáo sẽ đi sâu về chiến lược bộ nhớ ảo - giải pháp thành công nhất trong các chiến lược quản lý bộ nhớ.

Mục lục

Lời giới thiệu **3**

1 Bộ nhớ ảo **3**

1.1 Demand Paging . . . 4
1.2 Swapping . . . 4
1.3 Bộ nhớ ảo dùng chung . . . 4
1.4 Chế độ địa chỉ ảo và vật lý . . . 5
1.5 Điều khiển truy cập . . . 5

2 Caches **6**

2.1 Buffer Cache . . . 6
2.2 Page cache . . . 6
2.3 Swap cache . . . 6
2.4 Hardware caches . . . 7

3 Bảng trang Linux **7**

4 Phân cấp và giải phóng trang **8**

4.1 Phân bổ trang . . . 9
4.2 Giải phóng trang . . . 9

5 Ánh xạ bộ nhớ (Memory Mapping) **10**

6 Phân trang theo yêu cầu **11**

7 Linux Page Cache **12**

8 Hoán đổi và loại bỏ trang **12**

8.1 Reducing the Size of the Page and Buffer Caches . . . 13
8.2 Swapping Out System V shared Memory Pages . . . 14
8.3 Swap out và loại bỏ trang . . . 15

9 Swap Cache **16**

10 Swap in trang **16**

11 Tổng kết **17**

Lời giới thiệu

Hệ thống quản lý bộ nhớ là một trong những phần quan trọng nhất của hệ điều hành. Từ buổi bình minh của tính toán máy tính, đã nảy sinh vấn đề "cầu nhiều hơn cung", cần nhiều bộ nhớ hơn những gì bộ nhớ vật lý máy tính đang có. Nhiều chiến lược đã được đưa ra và "bộ nhớ ảo" chính là giải pháp thành công nhất. Nhờ đó, máy tính dường như có thể mở rộng bộ nhớ bằng việc chia sẻ tài nguyên giới hạn này giữa các tiến trình.

Hệ thống quản lý bộ nhớ cung cấp những thứ sau:

Không gian địa chỉ rộng: Bộ nhớ ảo có thể lớn hơn nhiều lần bộ nhớ vật lý

Tính bảo vệ: Mỗi tiến trình trong hệ thống có không gian địa chỉ ảo riêng, vì thế một tiến trình đang thực hiện một chương trình sẽ không thể làm ảnh hưởng đến cái khác. Vùng nhớ cũng có thể ngăn cản việc bị viết đè lên.

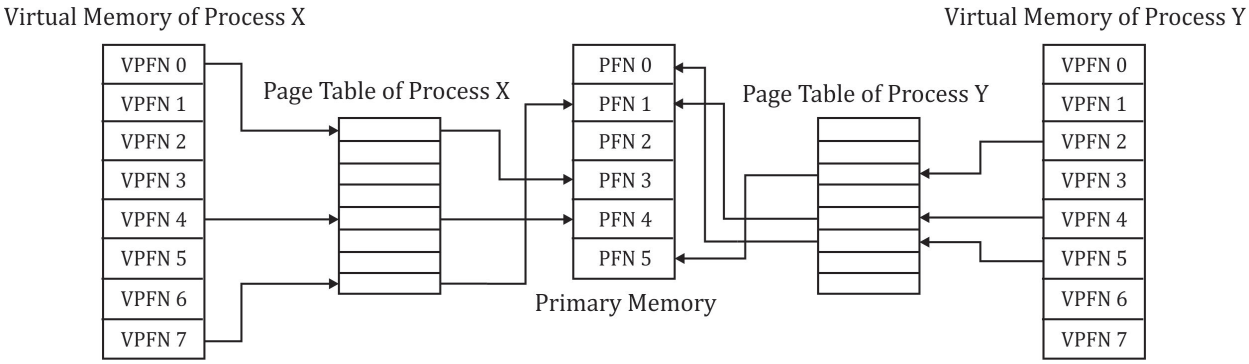
Ánh xạ bộ nhớ: Trong ánh xạ bộ nhớ, nôi dung của mọt file được dẫn trực tiếp đến không gian địa chri ảo của một tiến trình

Phân phối "công bằng" bộ nhớ vật lý: Hệ thống đảm bảo việc phân chia bộ nhớ vật lý giữa các tiến trình được "công bằng".

Chia sẻ bộ nhớ ảo: Các tiến trình có thể chia sẻ bộ nhớ với nhau, chẳng hạn cùng chạy "bash command shell". Thư viện liên kết động cũng là một ví dụ phổ biến khác khi đoạn lệnh thực thi có thể chia sẽ giữa vài tiến trình. Linux cũng hỗ trợ "Unix ™ System V" chia sẻ bộ nhớ với cơ chế truyền thông liên tiến trình.

1 Bộ nhớ ảo

Ở phần đầu tiên, chúng ta sẽ đưa ra mô hình trừu tượng của của bộ nhớ ảo.



Hình 1.1: Mô hình ánh xạ từ địa chỉ ảo đến địa chỉ vật lý

Để có thể thực thi một chương trình, vi xử lý sẽ đọc lần lượt từng câu lệnh từ bộ nhớ và giải mã chúng. Trong quá trình giải mã, vi xử lý sẽ cần tìm kiếm hoặc lưu trữ nội dung địa chỉ các câu lệnh trong bộ nhớ. Sau đó, nó sẽ thực thi câu lệnh và tiến đến câu lệnh tiếp theo của chương trình. Bằng cách này, vi xử luôn luôn cần truy cập bộ nhớ để tìm kiếm các câu lệnh hoặc tìm và lưu trữ dữ liệu.

Trong hệ thống bộ nhớ ảo, tất cả các địa chỉ cũng đều là ảo. Do đó chúng cần phải "chuyển sang" địa chỉ vật lý thông qua vi xử lý, điều này được thực hiện dựa trên thông tin từ một tập các bảng tạo ra bởi hệ điều hành. Cả bộ nhớ vật lý lẫn bộ nhớ ảo đều được chia thành các trang, và thường có kích thước giống nhau. Điều này là không bắt buộc, nhưng giả sử các trang có kết cỡ khác nhau, hệ thống sẽ khó khăn hơn trong việc quản lý. Hệ điều hành Linux trên hệ thống Alpha AXP dùng trang 8 Kbyte và trên hệ thống Intel x86 dùng trang 4 Kbyte. Mỗi trang lại có được gắn với con số riêng biệt, được gọi là "page frame number" (PFN), tạm dịch là "số hiệu trang".

Ở cách phân chia trang này, địa chỉ ảo lại được chia thành hai phần: Offset và PFN. Nếu kích thước trang là 4 Kbyte, bit 11:0 của địa chỉ ảo sẽ chứa phần offset còn bit 12 trở lên chứa PFN. Mỗi lần vi xử lý muốn tác động đến một địa chỉ ảo, nó sẽ phải đọc hai phần này. Từ đó, PFN của địa chỉ áo sẽ được "dịch sang" số hiệu trang vật lý và vi xử lý sẽ truy cập đến vị trí "chỉ ra" trên offset tương ứng đến trang vật lý. Quá trình này sẽ được dựa trên các "bảng phân trang".

Hình 1.1 miêu tả hai không gian địa chỉ ảo của tiến trình X và Y, mỗi tiến trình có bảng phân trang riêng. Ta có thể thấy trang ảo 4 của tiến trình X được nạp vào trang vật lý 4 của trang vật lý, trang ảo 2 của tiến trình Y được nạp vào trang vật lý 5,... Trang ảo không cần phải ánh xạ đến trang vật lý theo thứ tự nhất định nào. Mỗi entry (*tạm dịch*: đầu vào) của bảng phân trang chứa các thông tin sau:

- Cờ để kiểm tra đầu vào có đang trống không.
- Thông tin trang vật lý trong entry.
- Thông tinh điều khiển truy cập, chẳng hạn: Trang để ghi hay chỉ đọc,...

Vi xử lý dùng số hiệu trang ảo như một chỉ số để lấy được phần entry tương ứng trong bảng phân trang. Nếu entry không thể dùng được, không còn trống, vi xử lý sẽ báo lỗi để dành quyền xử lý cho hệ điều hành. Nếu vi xử lý không tìm thấy trang trong bảng phân trang, hiện tượng "lỗi trang" sẽ xảy ra, "ngắt" được sinh ra để tiến hành nạp trang. Còn ngược lại, sẽ tìm kiếm được địa chỉ trang tương ứng.

1.1 Demand Paging

Số lượng trang vật lý trong thực tế ít hơn nhiều so với trang ảo. Một cách để tiết kiệm bộ nhớ vật lý đó là chỉ nạp các trang ảo hiện đang được sử dụng để thực thi chương trình. Chiến lược này được gọi là "demand paging".

Khi vi xử lý muốn truy cập đến một trang ảo chưa được nạp vào, lỗi trang xảy ra. Hiện tượng "ngắt" sẽ được tạo ra bởi hệ điều hành nhằm bảo vệ các tiến trình khác. Nếu địa chỉ ảo tương ứng là hợp lệ, hệ điều hành sẽ nạp nó vào bộ nhớ. Tuy nhiên quá trình này tương đối tốn kém về mặt thời gian nên tiến trình có lỗi trang sẽ phải chờ đợi trong khi hệ điều hành có thể chọn tiến trình khác để thực thi. Thời điểm trang cần nạp đã được đưa vào bộ nhớ, tiến trình sẽ lại được tiếp tục tại vị trí xảy ra "ngắt.

1.2 Swapping

Giả thiết rằng, tiến trình yêu cầu một trang ảo mà không còn trang vật lý nào còn có thể tận dụng. Hệ điều hành khi đó sẽ phải giải quyết vấn đề bằng cách lấy một trang trong bộ nhớ ra để có thêm không gian cho trang yêu cầu (Swapping).

Nếu trong các trang đang nằm trong bộ nhớ chính có trang chưa bị ghi kể từ lần lưu gần nhất, ta chỉ cần bỏ trang đó ra ngoài. Tuy nhiên, nếu trang muốn đưa ra ngoài đã bị điều chỉnh, hệ điều hành sẽ phải tìm cách lưu trữ nội dung của trang đó trước. Và trang đó (còn được gọi là "dirty page") sẽ được đưa vào một loại file đặc biệt là "swap file". Quá trình này khá tốn kém.

Nếu thuật toán thực hiện swap không hiệu quả, hiện tượng *"thrashing"* sẽ xảy ra. Trong trường hợp này, hệ điều hành phải thực hiện quá nhiều công việc do nhiều trang được sử dụng thường xuyên lại bị đưa ra ngoài. Do đó, trong hình 1.1, giả sử trang vật lý 1 mới được dùng gần đây, nó sẽ khó có khả năng là ứng cử viên "sáng giá" để thay ra ngoài. Linux vì thế sử dụng kỹ thuật "Least Recently Used (LRU) page aging". Một tập hợp lưu thời gian sử dụng gần nhất của các trang được đưa ra, và sẽ cập nhật mỗi khi một phần tử được truy cập. Các trang mà có lần sử dụng gần nhất là xa nhất sẽ là ứng cử viên tốt để bị đưa ra ngoài.

1.3 Bộ nhớ ảo dùng chung

Bộ nhớ ảo giúp cho các tiến trình dễ dàng chia sẻ tài nguyên bộ nhớ với nhau. Tất cả các sự kiện truy cập đều được thực hiện thông qua các bảng phân trang và mỗi tiến trình lại có bảng phân trang riêng.

Giả sử hai tiến trình cùng muốn dùng chung một trang vật lý, số hiệu của trang sẽ phải xuất hiện trong entry của cả hai bảng. Tiến trình X và tiến trình Y trong hình 1.1 đều sử dụng trang vật lý số 1. Ta có thể thấy trang ảo tương ứng trong hai bảng phân trang không nhất thiết phải ở vị trí giống nhau.

1.4 Chế độ địa chỉ ảo và vật lý

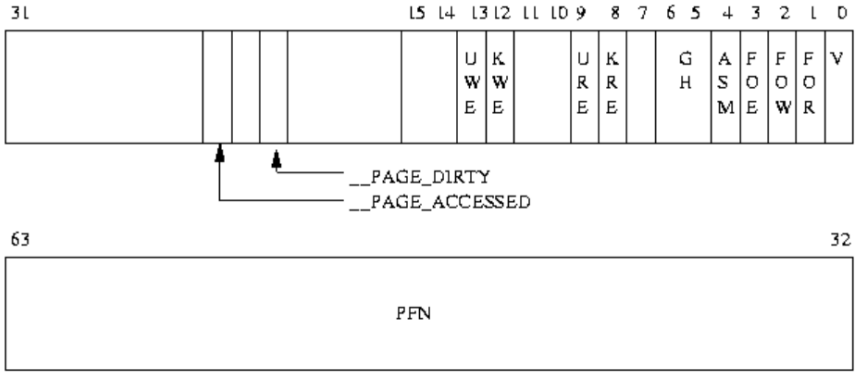
Sẽ là không hợp lý và không có cần thiết khi để hệ điều hành chạy trong bộ nhớ ảo vì khi đó nó sẽ phải quản lý bảng phân trang cho chính mình. Phần lớn các vi xử lý đa năng đều hỗ trợ cả chế độ địa chỉ ảo lẫn vật lý. Chế độ địa chỉ vật lý tất nhiên sẽ không yêu cầu bảng phân trang và vi xử lý cũng sẽ không thực hiện việc chuyển đổi địa chỉ trong chế độ này. Linux kernel được liên kết để chạy trong không gian địa chỉ vật lý.

Vi xử lý Alpha AXP không có chế độ địa chỉ vật lý đặc biệt. Thay vào đó nó chia không gian bộ nhớ ra vài vùng và chỉ định hai trong số đó là ánh xạ địa chỉ vật lý. Không gian địa chỉ kernel này được biết đến với tên gọi không gian địa chỉ KSEG và nó bao phủ tất cả các địa chỉ từ 0xfffffc0000000000 trở lên. Để thực thi từ các dòng lệnh liên kết trong KSEG hay muốn truy cập dữ liệu ở đó, đoạn lệnh phải được thực thi ở chế độ kernal.

1.5 Điều khiển truy cập

Như đã nói ở trên, các entry của bảng quản phân trang cũng chứa thông tin về phương thức truy cập. Bằng cách sử dụng các thông tin này, vi xử lý có thể biết được tiến trình được phép hay không được phép làm gì khi muốn truy cập đến bộ nhớ.

Có nhiều động cơ cho việc giới hạn quyền tác động này. Vài vùng nhớ, chẳng hạn chứa đoạn lệnh thực thi nên muốn các tiến trình chỉ có quyền đọc; hệ điều hành thường không nên để đoạn lệnh thực thi của nó bị viết đè lên. Ngược lại, trang mà chứa các dữ liệu có thể viết đè lên mà lại cố gắng dùng bộ nhớ để thực thi các lệnh chỉ dẫn thì sẽ nên bị báo lỗi. Phần lớn vi xử lý có ít nhất hai chế độ là *kernel* và *người dùng*. Đoạn lệnh của kernel sẽ không được thực thi bởi người dùng và cấu trúc dữ liệu của nó cũng chỉ có thể được truy cập khi mà vi xử lý chạy trong chế độ kernel.



Hình 1.2: Alpha AXP Page Table Entry

Thông tin điều khiển truy nhập được lưu trong PTE (Page Table Entry), hình 1.2 biểu diễn PTE cho Alpha AXP. Có vùng bit được ký hiệu bên trên có ý nghĩa như sau: **V**: Valid, thông tin cho biết PTE sẵn có hay không.

FOE: Fault on Execute, mỗi khi có sự yêu cầu thực thi lệnh trong trang, vi xử lý sẽ báo lỗi trang và chuyển quyền điều khiển cho hệ điều hành.

FOW: Fault on Write, lỗi trang xảy ra khi bị muốn ghi đè lên.

FOR: Fault on Read, lỗi trang xảy ra khi bị muốn đọc trang.

ASM: Address Space Match, phần này được dùng khi hệ điều hành muốn dọn dẹp một vài entry từ Translation Buffer.

KRE: Đoạn lệnh trong chế độ kernel có thể đọc trang này.

URE: Đoạn lệnh trong chế độ người dùng có thể đọc trang này.

GH: Granularity hint, được dùng nếu muốn ánh xạ cả một block (khối) với chỉ một entry của Translation Buffer.

KWE: Đoạn lệnh trong chế độ kernel có thể viết trang này.

UWE: Đoạn lệnh trong chế độ người dùng có thể viết trang này.

PFN: page frame number, vùng này chứa số hiệu trang vật lý cho cho entry.

Hai bit sau được định nghĩa và dùng bởi hệ điều hành Linux:

_PAGE_DIRTY: Cho biết trang cần viết ra lên file muốn trao đổi (swap) không.

PAGE ACCESSED: Cho biết trang có đang được truy nhập không.

2 Caches

Việc chúng ta triển khai hệ thống sử dụng những mô hình lý thuyết như trên, mặc dù có thể hoạt động, nhưng nhìn chung là không thực sự hiệu quả. Hệ quả là người người thiết kế hệ điều hành và bộ xử lý sẽ gặp khó khăn khi muốn nâng cao hiệu quả của hệ thống. Một cách tiếp cận khác đó chính là duy trì bộ nhớ caches đối với những thông tin và dữ liệu hữu ích, điều này làm cho việc vận hành hệ thống nhanh hơn. Sau đây là những bộ nhớ cache có trong hệ điều hành Linux.

2.1 Buffer Cache

Bộ nhớ buffer cache chứa những dữ liệu buffer được sử dụng bởi những block device driver.

Những dữ liệu buffer này có kích thước cố định (ví dụ là 512 bytes) và chứa những khối thông tin hoặc được đọc từ một block device, hoặc là được ghi vào đó. Một block device chỉ có thể được truy cập bằng việc đọc hoặc việc ghi một khối dữ liệu có kích thước cố định. Tất cả những đĩa cứng là block device.

Bộ nhớ buffer cache được đánh số thứ tự thông qua định danh của thiết bị và số hiệu của desired block, điều này giúp cho việc tìm một block dữ liệu nhanh hơn. Những block device chỉ có thể được truy cập thông qua bộ nhớ buffer cache. Nếu dữ liệu được tìm thấy ở bộ nhớ buffer cache thì chúng ta không cần phải đọc từ block device vật lý (ví dụ như đĩa cứng) nữa, điều này giúp việc tìm kiếm và đọc dữ liệu nhanh hơn rất nhiều.

2.2 Page cache

Bộ nhớ page cache được dùng để tăng tốc độ truy nhập ảnh và dữ liệu trên đĩa.

Bộ nhớ page cache được dùng để lưu trữ thông tin logic của một file thuộc 1 trang tại một thời điểm, được truy nhập thông qua file. Khi trang được đọc từ đĩa vào bộ nhớ, trang sẽ được lưu trữ ở bộ nhớ page cache.

2.3 Swap cache

Bộ nhớ swap cache được dùng để chứa những trang bị thay đổi (cách gọi khác là trang bẩn).

Miễn là những trang không bị thay đổi sau khi được ghi vào swap file, thì lần tiếp theo, những file này sẽ được đưa ra khỏi swap file. Thay vào đó thì các trang sẽ bị loại bỏ một cách dễ dàng. Trong một hệ thống swap mạnh thì cách tiến hành như trên sẽ giúp loại bỏ những hoạt động không cần thiết và tốn kém trên đĩa.

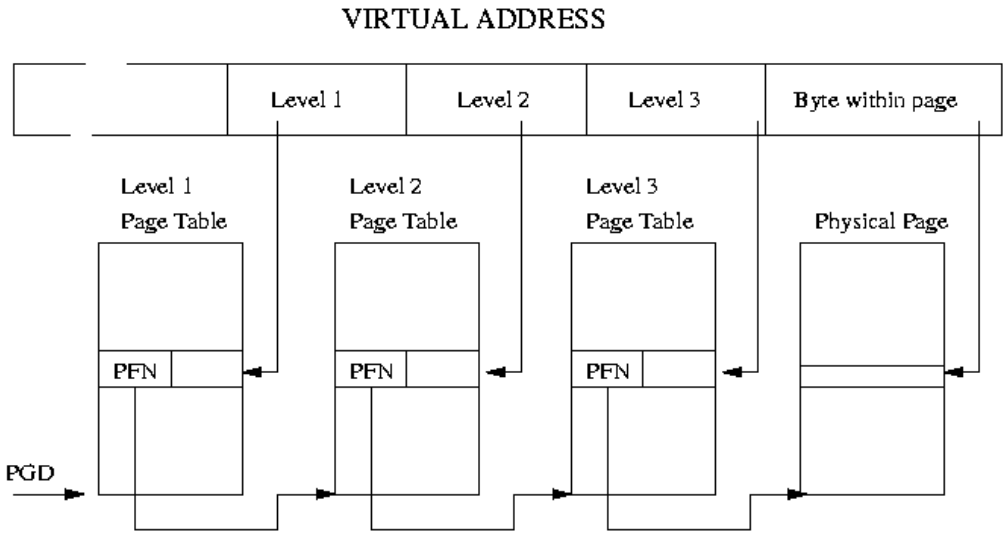
2.4 Hardware caches

Một bộ nhớ hardware cache được triển khai phổ biến là nằm ở trong bộ xử lý, 1 cache của Page Table Entries. Trong trường hợp này, bộ xử lý không luôn luôn đọc trực tiếp từ bảng trang, mà thay vào đó là đọc từ caches translations của trang khi bộ xử lý cần. Những caches translation của những trang như thế được gọi là Translation Look-aside Buffers, chúng chứa bản sao lưu trữ của những Page Table Entries từ một hoặc nhiều bộ xử lý trong hệ thống.

Khi có một tham chiếu đến địa chỉ ảo được tạo, bộ xử lý sẽ cố gắng tìm một sự tương ứng ở trong TLB. Nếu bộ xử lý tìm thấy, nó có thể trực tiếp chuyển địa chỉ aỏ thành địa chỉ vật lý. Nếu bộ xử lý không thể tìm thấy sự tương ứng ở trong TLB, bộ xử lý sẽ yêu cầu hệ điều hành giúp đỡ. Hệ điều hành sẽ tạo ra một TLB mới cho địa chỉ ánh xạ.

Hạn chế của việc sử dụng bộ nhớ cache là khi Linux muốn tiết kiệm công sức, nó phải sử dụng nhiều thời gian và không gian để duy trì những bộ nhớ cache này, và khi bộ nhớ cache bị hỏng, hệ thống sẽ sập theo.

3 Bảng trang Linux



Hệ điều hành Linux định nghĩa ra ba mức cho các bảng trang. Mỗi bảng trang được truy nhập sẽ chứa số hiệu trang của mức tiếp theo của bảng trang. Hình ảnh ở trên cho ta thấy cách bộ nhớ ảo được phân tách ra thành các trường như thế nào, mỗi trường sẽ chứa một offset để truy cập đến một bảng trang cụ thể. Để có thể chuyển một địa chỉ ảo thành một địa chỉ vật lý tương ứng, bộ xử lý cần phải lấy thông tin từ các trường mức độ, chuyển nó thành một offset vào trong trang vật lý chứa bảng trang và đọc số hiệu trang của mức tiếp theo của bảng trang. Điều này được thực hiện ba lần cho đến khi số hiệu trang của trang vật lý chứa địa chỉ chỉ ảo được tìm thấy. Bây giờ, trường cuối cùng trong bộ nhớ ảo, byte offset, được dùng để tìm dữ liệu bên trong trang.

Mỗi nền tảng mà hệ điều hành Linux chạy trên phải cung cấp những translation macro mà cho phép kernel duyệt qua bảng trang cho một tiến trình cụ thể. Bằng cách này, kernel không cần phải biết về định dạng của bảng trang và cách bảng trang được sắp xếp như nào.

Hệ điều hành Linux sử dụng cùng code điều chỉnh bảng trang cho bộ xử lý Alpha (có ba mức cho các bảng trang), và cho bộ xử lý Intel x86 (có hai mức cho các bảng trang).

4 Phân cấp và giải phóng trang

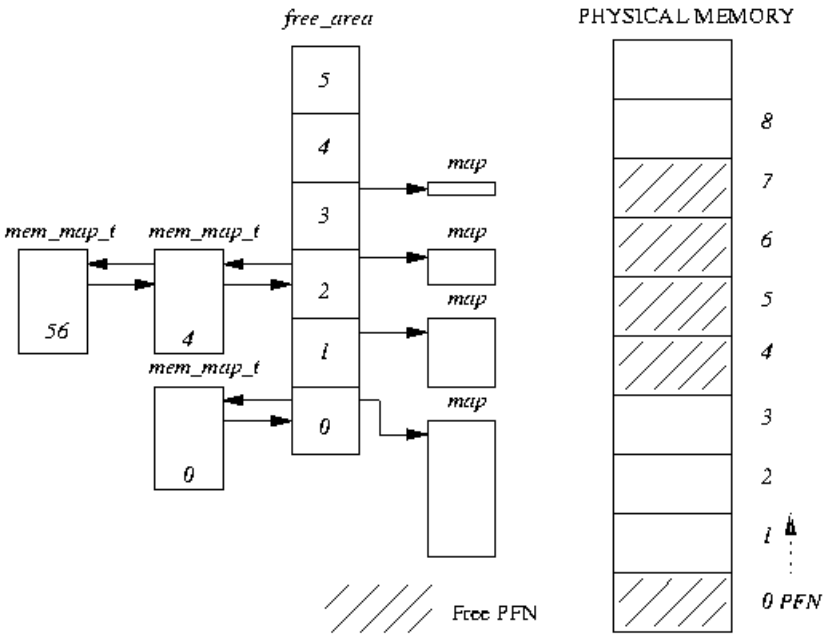
Có nhiều nhu cầu đối với các trang vật lý trong hệ thống. Ví dụ, khi một hình ảnh được tải vào bộ nhớ, hệ điều hành cần phân bổ các trang. Chúng sẽ được giải phóng khi hình ảnh hoàn tất quá trình thực thi và được tải xuống. Một cách sử dụng khác cho các trang vật lý là giữ các cấu trúc dữ liệu cụ thể của **kernel**, chẳng hạn như chính các bảng trang. Các cơ chế và cấu trúc dữ liệu được sử dụng để phân bổ trang và định vị giao dịch có lẽ là quan trọng nhất trong việc duy trì hiệu quả của hệ thống con bộ nhớ ảo. Tất cả các trang vật lý trong hệ thống đều được mô tả bằng cấu trúc dữ liệu **mem_map**, mỗi cấu trúc là danh sách các **mem_map_t**. Mỗi **mem_map_t** mô tả một trang vật lý duy nhất trong hệ thống. Các trường quan trọng (liên quan đến quản lý bộ nhớ) là:

1. **count**
   Số lượng người dùng của trang này. Biến count lớn hơn một khi trang được chia sẻ giữa nhiều quá trình.
2. **age**
   Trường này mô tả tuổi của trang và được sử dụng để quyết định xem trang có phù hợp để loại bỏ hoặc hoán đổi hay không.
3. **map_nr**
   Số khung trang vật lý mà mem_map_t này mô tả.

Vector **free_area** được sử dụng bởi mã phân bổ trang để tìm và giải phóng trang. Mỗi phần tử của free_area chứa thông tin về các khối trang. Phần tử đầu tiên trong mảng mô tả các trang đơn, các khối tiếp theo gồm 2 trang, các khối tiếp theo gồm 4 trang và tiếp tục tăng theo lũy thừa của 2. Phần tử danh sách được sử dụng làm đầu hàng đợi và có các con trỏ đến cấu trúc dữ liệu trang trong mảng mem_map. Các khối trang tự do được xếp hàng đợi ở đây. **map** là một con trỏ tới một bitmap theo dõi các nhóm trang được phân bổ có kích thước này. Bit N của bitmap được đặt nếu khối trang thứ N là tự do.

Hình "free-area" cho thấy cấu trúc free_area. Phần tử 0 có một trang tự do (số khung trang 0) và phần tử 2 có 2 khối tự do gồm 4 trang, khối thứ nhất bắt đầu từ khung trang số 4 và khối thứ hai ở khung trang số 56.

4.1 Phân bổ trang



Hình 4.1: The free-area data structure

Linux sử dụng thuật toán Buddy để phân bổ và giải phóng các khối trang một cách hiệu quả. Mã phân bổ trang cố gắng phân bổ một khối gồm một hoặc nhiều trang vật lý. Các trang được phân bổ trong các khối có kích thước là lũy thừa 2. Điều đó có nghĩa là nó có thể phân bổ khối 1 trang, 2 trang, 4 trang, v.v. Miễn là có đủ các trang tự do trong hệ thống để cấp yêu cầu này (nr_free_pages> min_free_pages), mã phân bổ sẽ tìm kiếm free_area cho một khối các trang có kích thước được yêu cầu. Mỗi phần tử của free_area có một bản đồ của các khối trang được phân bổ và miễn phí cho khối có kích thước đó. Ví dụ, phần tử 2 của mảng có một bản đồ bộ nhớ mô tả các khối tự do và được cấp phát, mỗi khối dài 4 trang. Thuật toán phân bổ trước tiên sẽ tìm kiếm các khối trang có kích thước được yêu cầu. Nó theo sau chuỗi các trang miễn phí được xếp hàng trên phần tử danh sách của cấu trúc dữ liệu free_area. Nếu không có khối trang nào có kích thước được yêu cầu là miễn phí, thì các khối có kích thước tiếp theo (gấp đôi kích thước được yêu cầu) sẽ được tìm kiếm. Quá trình này tiếp tục cho đến khi tất cả free_area đã được tìm kiếm hoặc cho đến khi tìm thấy một khối các trang. Nếu khối trang được tìm thấy lớn hơn khối lượng yêu cầu, nó phải được chia nhỏ cho đến khi có khối có kích thước phù hợp. Bởi vì mỗi khối có kích thước lớn hơn 2 trang nên quá trình chia nhỏ này rất dễ dàng vì bạn chỉ cần bẻ đôi các khối. Các khối miễn phí được xếp vào hàng đợi thích hợp và khối trang được phân bổ được trả lại cho người gọi.

4.2 Giải phóng trang

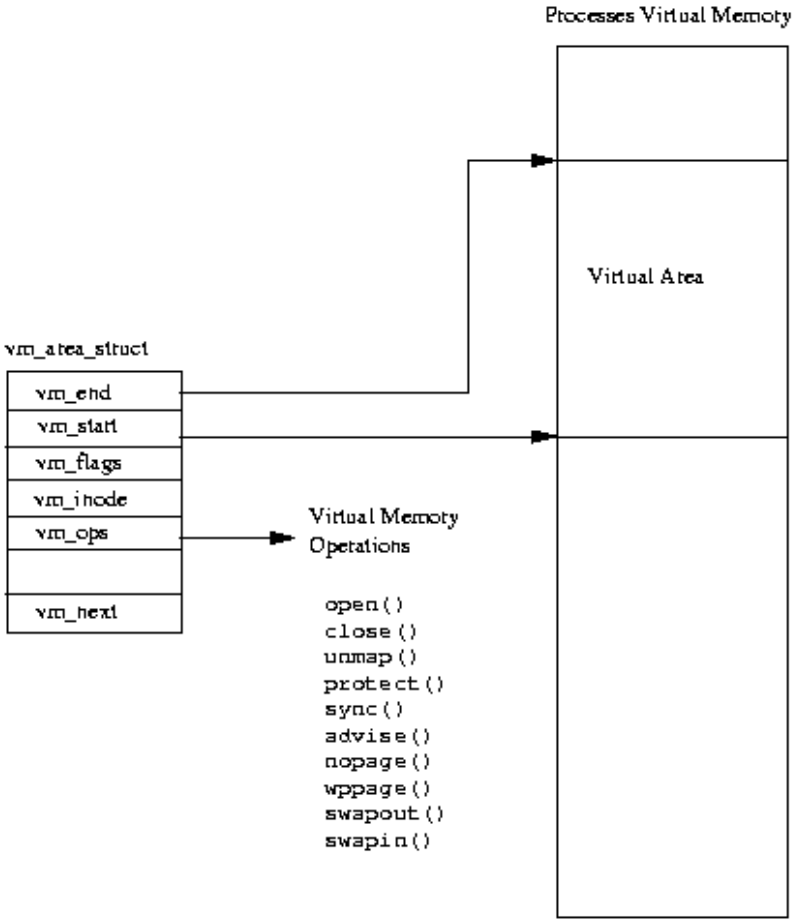
Việc phân bổ các khối trang có xu hướng phân mảnh bộ nhớ với các khối trang miễn phí lớn hơn được chia thành các khối nhỏ hơn. Mã phân bổ trang kết hợp các trang thành các khối trang miễn phí lớn hơn bất cứ khi nào có thể. Trên thực tế, kích thước khối trang rất quan trọng vì nó cho phép dễ dàng kết hợp các khối thành các khối lớn hơn.

Bất cứ khi nào một khối trang được giải phóng, khối liền kề hoặc khối bạn thân có cùng kích thước sẽ được kiểm tra xem nó có còn trống hay không. Nếu đúng như vậy, thì nó được kết hợp với khối trang mới được giải phóng để tạo thành một khối trang miễn phí mới cho khối trang có kích thước tiếp theo. Mỗi khi hai khối trang được kết hợp lại thành một khối trang miễn phí lớn hơn, mã phân bổ trang sẽ cố

gắng kết hợp lại khối đó thành một khối lớn hơn. Bằng cách này, các khối trang miễn phí có dung lượng lớn như mức sử dụng bộ nhớ sẽ cho phép.

5 Ánh xạ bộ nhớ (Memory Mapping)

Khi một file được thực thi, nội dung của file thực thi phải được đưa vào không gian địa chỉ ảo của tiến trình. File thực thi không thực sự được đưa vào bộ nhớ vật lý, thay vào đó nó chỉ được liên kết vào bộ nhớ ảo của các tiến trình. Sau đó, khi các phần của chương trình được tham chiếu bởi ứng dụng đang chạy, file được đưa vào bộ nhớ từ thực thi. Liên kết của một file vào một không gian địa chỉ ảo của tiến trình được gọi là ánh xạ bộ nhớ.



Mọi tiến trình bộ nhớ ảo được biểu diễn bằng cấu trúc dữ liệu mm_struct. Điều này chứa thông tin về hình ảnh mà nó hiện đang thực thi (ví dụ: bash) và cũng có các con trỏ đến một số cấu trúc dữ liệu vm_area_struct. Mỗi cấu trúc dữ liệu vm_area_struct mô tả điểm bắt đầu và kết thúc của vùng bộ nhớ ảo, các tiến trình có quyền truy cập vào bộ nhớ đó và một tập hợp các hoạt động cho bộ nhớ đó. Các hoạt động này là một tập hợp các quy trình mà Linux phải sử dụng khi thao tác với vùng này của bộ nhớ ảo. Ví dụ: một trong các hoạt động bộ nhớ ảo thực hiện các hành động chính xác khi quá trình đã cố gắng truy cập bộ nhớ ảo này nhưng phát hiện (thông qua lỗi trang) rằng bộ nhớ thực sự không nằm trong bộ nhớ vật lý. Thao tác này là thao tác nopage. Thao tác nopage được sử dụng khi Linux yêu cầu trang các trang của hình ảnh thực thi vào bộ nhớ.

Khi một hình ảnh thực thi được ánh xạ vào một địa chỉ ảo xử lý, một tập hợp các cấu trúc dữ liệu vm_area_struct được tạo ra. Mỗi cấu trúc dữ liệu vm_area_struct đại diện cho một phần của hình ảnh thực thi; mã thực thi, dữ liệu khởi tạo (biến), dữ liệu đơn nguyên, v.v. Linux hỗ trợ một số hoạt động bộ nhớ ảo tiêu chuẩn và khi cấu trúc dữ liệu vm_area_struct được tạo, tập hợp các hoạt động bộ nhớ ảo chính xác được liên kết với chúng.

6 Phân trang theo yêu cầu

Khi một hình ảnh thực thi đã được ánh xạ trong bộ nhớ vào một bộ nhớ ảo xử lý, nó có thể bắt đầu thực thi. Vì chỉ phần đầu của hình ảnh được kéo vào bộ nhớ, nó sẽ sớm truy cập vào một vùng bộ nhớ ảo chưa có trong bộ nhớ vật lý. Khi một quá trình truy cập vào một địa chỉ ảo không có mục nhập bảng trang hợp lệ, bộ xử lý sẽ báo lỗi trang cho Linux. Lỗi trang mô tả địa chỉ ảo nơi xảy ra lỗi trang và kiểu truy cập bộ nhớ đã gây ra.

Linux phải tìm vm_area_struct đại diện cho vùng bộ nhớ mà lỗi trang xảy ra. Vì việc tìm kiếm thông qua cấu trúc dữ liệu vm_area_struct là rất quan trọng để xử lý hiệu quả các lỗi trang, chúng được liên kết với nhau trong một cây AVL (Adelson-Velskii và Landis) kết cấu. Nếu không có cấu trúc dữ liệu vm_area_struct cho địa chỉ ảo bị lỗi này, quá trình này đã truy cập vào một địa chỉ ảo bất hợp pháp. Linux sẽ báo hiệu quá trình, gửi một tín hiệu SIGSEGV, và nếu quá trình không có trình xử lý cho tín hiệu đó, nó sẽ bị chấm dứt.

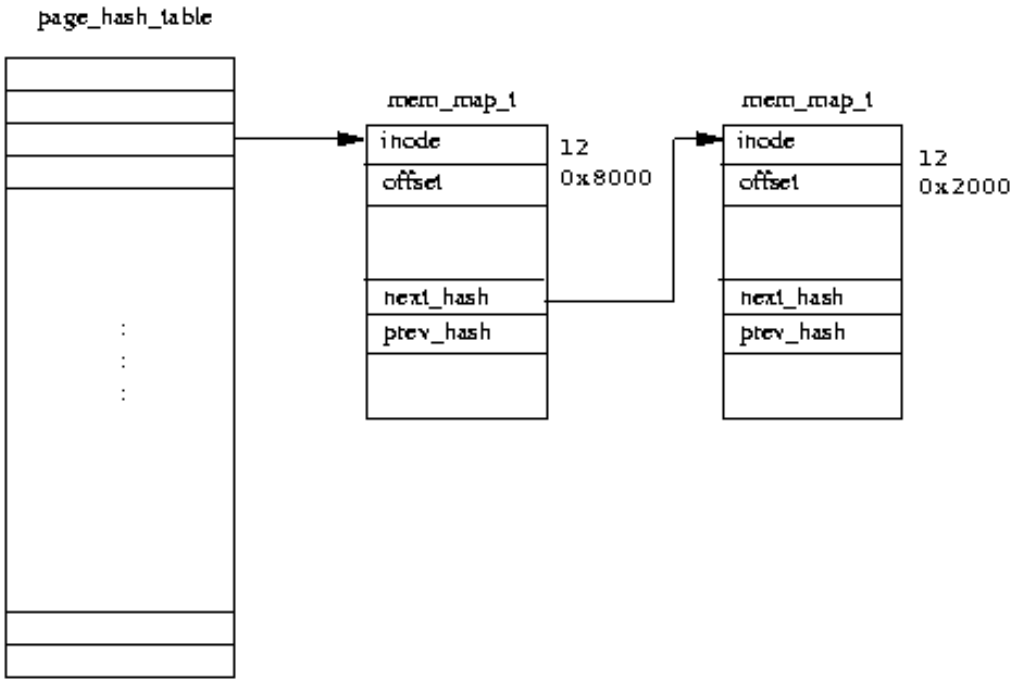
Tiếp theo, Linux sẽ kiểm tra loại lỗi trang xảy ra đối với các loại truy cập được phép đối với vùng bộ nhớ ảo này. Nếu quá trình đang truy cập bộ nhớ theo cách bất hợp pháp, chẳng hạn như ghi vào một khu vực mà nó chỉ được phép đọc từ đó, nó cũng được báo hiệu với lỗi bộ nhớ.

Bây giờ Linux đã xác định rằng lỗi trang là hợp pháp, nó phải xử lý nó. Linux phải phân biệt giữa các trang nằm trong tệp hoán đổi và những trang là một phần của hình ảnh thực thi trên đĩa ở đâu đó. Nó thực hiện điều này bằng cách sử dụng mục nhập bảng trang cho địa chỉ ảo bị lỗi này. Nếu mục nhập bảng trang của trang không hợp lệ nhưng không trống, lỗi của trang là do một trang hiện đang được giữ trong tệp hoán đổi. Đối với các mục nhập bảng trang Alpha AXP, đây là các mục nhập không có bộ bit hợp lệ nhưng có giá trị khác 0 trong trường PFN của chúng. Trong trường hợp này, trường PFN giữ thông tin về vị trí trong tệp hoán đổi (và tệp hoán đổi nào) mà trang đang được lưu giữ. Cách xử lý các trang trong tệp hoán đổi được mô tả ở phần sau của chương này.

Không phải tất cả các cấu trúc dữ liệu vm_area_struct đều có một tập hợp các hoạt động bộ nhớ ảo và thậm chí những cấu trúc này có thể không có hoạt động nopage. Điều này là do theo mặc định, Linux sẽ sửa chữa quyền truy cập bằng cách phân bổ một trang vật lý mới và tạo một mục nhập bảng trang hợp lệ cho nó. Nếu có một thao tác nopage cho vùng bộ nhớ ảo này, Linux sẽ sử dụng nó.

Hoạt động nopage Linux chung được sử dụng cho các hình ảnh thực thi được ánh xạ trong bộ nhớ và nó sử dụng bộ đệm trang để đưa trang hình ảnh cần thiết vào bộ nhớ vật lý. Tuy nhiên, trang yêu cầu được đưa vào bộ nhớ vật lý, các bảng trang quy trình được cập nhật. Có thể cần thiết cho các hành động cụ thể của phần cứng để cập nhật các mục nhập đó, đặc biệt nếu bộ xử lý sử dụng bộ đệm dịch chuyển sang một bên. Bây giờ lỗi trang đã được xử lý, nó có thể bị loại bỏ và quá trình được khởi động lại theo hướng dẫn đã thực hiện truy cập bộ nhớ ảo bị lỗi.

7 Linux Page Cache



Mỗi tệp trong Linux được xác định bởi cấu trúc dữ liệu inode VFS và mỗi inode VFS là duy nhất và mô tả đầy đủ một và chỉ một tệp. Chỉ mục trong bảng trang được lấy từ inode VFS của tệp và phần bù vào tệp.

Bất cứ khi nào một trang được đọc từ tệp ánh xạ bộ nhớ, chẳng hạn như khi nó cần được đưa trở lại bộ nhớ trong quá trình phân trang theo yêu cầu, trang sẽ được đọc qua bộ đệm ẩn của trang. Nếu trang có trong bộ đệm, một con trỏ đến cấu trúc dữ liệu mem_map_t đại diện cho nó sẽ được trả về mã xử lý lỗi trang. Nếu không, trang phải được đưa vào bộ nhớ từ hệ thống tệp chứa hình ảnh. Linux phân bổ một trang vật lý và đọc trang đó từ tệp trên đĩa.

Nếu có thể, Linux sẽ bắt đầu đọc trang tiếp theo trong tệp. Trang này được đọc trước có nghĩa là nếu tiến trình đang truy cập các trang trong tệp theo thứ tự, thì trang tiếp theo sẽ được chờ đợi trong bộ nhớ cho tiến trình.

Theo thời gian, bộ nhớ cache của trang phát triển khi hình ảnh được đọc và thực thi. Các trang sẽ bị xóa khỏi bộ nhớ cache vì chúng không còn cần thiết, chẳng hạn như một hình ảnh không còn được sử dụng bởi bất kỳ quá trình nào. Vì Linux sử dụng bộ nhớ, nó có thể bắt đầu chạy chậm trên các trang vật lý. Trong trường hợp này, Linux sẽ giảm kích thước của bộ đệm trang.

8 Hoán đổi và loại bỏ trang

Khi bộ nhớ vật lý không đủ cho nhu cầu, hệ thống quản lý bộ nhớ của Linux phải cố gắng giải phóng các trang vật lý. Nhiệm vụ này thuộc về kernel swap daemon (kswapd).

Kernel swap daemon là một loại chương trình đặc biệ và là 1 luồng kernel. Những luồng Kernel là những chương trình không có bộ nhớ ảo, thay vào đó chúng chạy ở Kernel Mode trong không gian địa

chỉ vật lý. Kswapd không chỉ đơn thuần là hoán đổi các trang thành các tệp "swap file" của hệ thống mà còn đảm bảo rằng có đủ các trang tự do trong hệ thống để giữ cho hệ thống quản lý bộ nhớ hoạt động một cách hiệu quả

Kswapd được khởi động bởi đơn vị xứ lý kernel tại thời điểm khởi động và chờ cho đến khi thời gian hoán đổi của kernel trong một định kỳ kết thúc

Mỗi lần vào lúc thời gian của một định kỳ kết thúc, kswapd sẽ để ý xem số lượng trang tự do trong hệ thống có quá thấp hay không. Để biết được điều đó, nó sử dụng 2 biến là *free_pages_high* và *free_pages_low* để kiểm tra và dựa vào đó để nó quyết định liệu có nên giải phóng một số trang. Miễn là số trang tự do trong một hệ thống luôn duy trì ở trên mức trên *free_pages_high*, kswapd sẽ không làm gì cả, nó lại sleep cho đến khi thời gian của một định kỳ kế tiếp kết thúc .Dựa vào việc kiểm tra này, kswapd đếm số lượng các trang hiện tại đang được ghi vào tệp "swap file". Nó lưu lại số lượng này vào biến đếm *nr_async_page*, *nr_async_page* sẽ tăng lên mỗi khi có một trang được xếp vào hằng đợi để ghi ra tệp "swap file" và giảm khi việc viết vào đó hoàn tất. *free_pages_low* và *free_pages_high* được thiết lập lúc khởi động và nó liên quan đớn số lượng những trang vật lý trong hệ thống. Nếu số lượng những trang tự do trong hệ thống ở dước mức *free_pages_high* và vẫn tên mức *free_pages_low*, kswapd sẽ thử 3 cách để làm giảm số lượng những trang vật lý được sử dụng trong hệ thống

- Giảm kích thước của bufer và page caches
- Hoán đổi các trang của vùng nhớ dùng chung System V
- Hoán đổi và loại bỏ các trang

Nếu số lượng các trang tự do trong hệ thống ở dưới mức *free_pages_low*, kswapd sẽ cố gắng giải phóng 6 trang trước khi bắt đầu lần chạy tiếp theo. Nếu không được nó sẽ cố gắng giải phóng 3 trang. Mỗi cách ở trên sẽ được thử cho đến khi đúng số lượng trang được giải phóng. Kswapd ghi nhớ cái cách nó sử dụng lần cuối cùng khi nó cố gắng giải phóng các trang vật lý thành công. Mỗi lần nó chạy nó sẽ bắt đầu thử giải phóng những trang bằng các sử dụng các cách mà nó đã giải phong thành công gần đây nhất.

Sau khi có đủ số trnag tự do, kswapd sẽ lại sleep cho đến khi thời gian của định kỳ kết thúc. Nếu lý do kswapd giải phóng cá trang là do số lượng các trang tự do trong hệ thống ở dưới mức *free_pages_low* thì nó chỉ sleep 1 nửa thời gian bình thường. Một khi số lượng những trang tự do nhiều hơn mức *free_pages_low*, kswapd sẽ quay lại trạng thới sleep lâu hơn vào giữa những lần kiểm tra.

8.1 Reducing the Size of the Page and Buffer Caches

Các trang được lưu giữ trong các bộ nhớ Page Cache và Buffer Cache là những ứng cử viên sáng giá để được giải phóng trong vector *free_area* . Bộ nhớ Page Cache chứa các trang của các file bộ nhớ được ánh xạ, nó có thể chứa những trang không cần thiết làm đầy bộ nhớ hệ thống. Tương tự với bộ nhớ Bufer Cache, bộ nhớ Buffer Cache chứa các bộ đệm được đọc hoặc ghi vào thiết bị vật lý, nó cũng có thể chứa những thứ không cần thiết. Khi các trang vật lý trong hệ thống bắt đầu hết, việc loại bỏ các trang khỏi các bộ nhớ Cache này tương đối dễ dàng vì nó không yêu cầu ghi vào các thiết bị vật lý (không giống như việc hoán đổi các trang trong bộ nhớ). Việc loại bỏ các trang này không gây ra sự ảnh hưởng bất lợi nhiều ngoài việc làm cho việc truy cập vào các thiết bị vật lý và các file được bộ nhớ ánh xạ chậm hơn. Tuy nhiên, nếu việc loại bỏ các trang khỏi các bộ nhớ Cache này được thực hiện một cách công bằng, tất cả các quá trình sẽ bị ảnh hưởng như nhau.

Mỗi lần như vậy kswapd cố gắng thu nhỏ những cache này.Nó kiểm tra một khối các trang trong vector trang mem_map để xem liệu có thể bỏ bất kì trang nào từ bộ nhớ không. Kích thức của khối trang được kiểm tra sẽ cao hơn nếu kswapd được hoán đổi mạnh khi mà các trang trong tự do trong hệ thống rớt xuống mức thấp đáng quan ngại. Những khối trang được kiểm tra theo một chu kì, những khối trang khác nhau được kiểm tra mỗi khi cố gánh làm thu nhỏ vùng nhớ ánh xạ. Đây được gọi là thuộc toán *clock*, giống như kim phút của đồng hồ, toàn bộ vector trang mem_map kiểm tra một vài trang tại một thời điểm.

Mỗi trang được kiểm tra sẽ kiểm tra xem nó được lưu ở bộ nhớ Page Cache hay Bufer Cache. Các trang được chia sẻ không được xem xét để hủy trong thời gian này và một trang không thể nằm trong cả 2 bộ nhớ cache cùng một thời điểm. Nếu trang không ở trong bất cứ bộ nhớ cache nào thì trang tiếp theo đó của vector trang mem_map sẽ được kiểm tra.

Các trang được lưu trong bộ nhớ Bufer Cache giúp phân bổ bộ đệm và xử lý hiệu quả hơn. Code thu nhỏ vùng nhớ ánh xạ cố gắng giải phóng vùng nhớ đệm được chứa trong trang được kiểm tra.

Nếu tất các các vùng đệm đệm giải phóng, sau đó các trang chứa chúng cũng sẽ được giải phóng. Nếu trang được kiểm tra là nằm trong bộ nhớ Page Cache, nó sẽ xóa khỏi bộ nhớ Page Cache và được giải phóng. Khi có đủ số trang được giải phóng trong lần này thì kswapd sẽ sleep đến cho đến lần đánh thức của định kỳ tiếp theo. Vì không có trang được giải phóng nào là một phần của bộ nhớ ảo của bất kỳ quá trình nào (chúng là các trang được lưu trong bộ nhớ cache) nên không cần cập nhật bảng trang. Nếu không loại bỏ đủ số trang đã lưu trong bộ nhớ cache thì kswapd sẽ cố gắng hoán đổi một số trang được chia sẻ.

8.2 Swapping Out System V shared Memory Pages

Vùng nhớ dùng chung System V là một cơ chế giao tiếp giữa các quá trình cho phép hai hoặc nhiều quá trình chia sẻ bộ nhớ ảo để truyền thông tin giữa chúng với nhau. Mỗi vùng của vùng nhớ dùng chung System V được mô tả bằng cấu trúc dữ liệu shmid_ds. Nó chứ con trỏ trỏ tới day sách cảu cấu trúc dữ liệu vm_area_struct, một cấu trúc cho mỗi quá trình chia sẻ vùng bộ nhớ ảo này.. Mỗi cấu trúc dự vm_area_struct của vùng nhớ dùng chung System V này liên kết với nhau bằng cách sử dụng con trỏ vm_next_shared và vm_prev_shared. Mỗi cầu trúc dữ liệu shmid_ds cũng chứa danh sách các entry của bảng quản phân trang, mỗi entry trong bảng đó mô tả trang vật lý mà một trang ảo được chia sẻ được ánh xạ tới.

Kswapd cũng sử dụng thuột toán *clock* khi hoán đổi các trang của vùng nhớ dùng chung System V. Mỗi lần chạy, nó sẽ nhớ trang nào trong vùng bộ nhớ ảo được chia sẻ mà nó đã hoán đổi lần cuối. Nó thực hiện điều này bằng cách giữ hai entry, entry đầu tiên là entry trong tập hợp cấu trúc dữ liệu shmid_ds, entry thứ hai là danh sách các entry của bảng quản phân trang cho khu vực này của vùng nhớ dùng chung System V.

Vì số khung trang vật lý cho một trang ảo nhất đỉnh của vùng nhớ dùng chung System V được chứa trong bản phân trang của tất cả các tiến trình dùng chung vùng nhớ ảo này, kswapd phải điều chỉnh tất các những bảng trang để thể hiện rằng trang không ở trong bộ nhớ nhưng hiện ở trong tệp swap file. Đối với mỗi trang mà nó hoán đổi, kswapd tìm kiểm entry của trang trong mỗi bản quản phân trang(dựa theo con trỏ cấu trúc dữ liệu vm_area_struct). Nếu entry trong bảng quản phân trang của trang nằm trong vùng nhớ dùng chung là valid, nó sẽ chuyển đổi thành invalid nhưng được hoán đổi

và giảm số lượng các trang của users đi 1. Định dạng của một entry của bảng quản phân trang trong vùng nhớ dùng chung System V được hoán đổi chứa một index của tập hợp cấu trúc dữ liệu shmid_ds và một index vào các entry của bảng quản phân trang cho khu vực này của vùng nhớ dùng chung System V.

Nếu số lượng các trang là 0 sau thi khi tất cả các bảng phân trang của tiến trình được sửa đổi, các trang được chia sẻ có thể viết vào tệp swap file. Entry của bảng quản phân trang trong danh sách của cấu trúc dữ liệu shmi_ds trỏ tới khu vực vùng nhớ dùng chung System V thì được thay thế bởi entry của bảng quản phân trang đã được hoán đổi. Một entry của bảng quản phân trang đã được hoán đổi là invalid nhưng chứa index trong các tệp swap file đang mở và offset trong file đó là nơi các trang được hoán đổi có thể tìm thấy. Thông tin này sẽ được sử dụng khi trang phải được đưa lại bộ nhớ vật lý.

8.3 Swap out và loại bỏ trang

Kswapd lần lượt xem xét từng tiến trình trong hệ thống để xem liệu nó có phải là một ứng cử viên tốt cho việc hoán đổi hay không.

Các ứng cử viên tốt là các tiến trình có thể được hoán đổi (một số không thể) và có một hoặc nhiều trang có thể được hoán đổi hoặc loại bỏ khỏi bộ nhớ. Các trang chỉ được hoán đổi từ bộ nhớ vật lý vào các tệp swap file của hệ thống nếu dữ liệu trong đó không thể được truy xuất theo cách khác.

Rất nhiều nội dung của một hình ảnh đang chạy từ fie ảnh có thể dễ dàng mở lại từ file đó.Ví dụ, lệnh thực thi của một hình ảnh sẽ không bao giờ được điều chỉnh bởi hình ảnh và sẽ không bao giờ được viết vào tệp swap file. Nó chỉ có thể bị xóa đi, khi những nội dung này được gọi lại bởi tiến trình, chúng sẽ được đưa lại bộ nhớ.

Khi quá trình hoán đổi đã được xác định,kswapd sẽ xem xét tất cả các vùng bộ nhớ ảo của nó để tìm kiếm các vùng không được chia sẻ hoặc bị khóa.

Linux không hoán đổi tất cả các trang có thể hoán đổi của tiến trình mà nó đã chọn; thay vào đó nó chỉ loại bỏ một số trang nhỏ.

Không thể hoán đổi hoặc hủy các trang nếu chúng bị khóa trong bộ nhớ.

huật toán hoán đổi Linux sử dụng là tính năng lão hóa trang. Mỗi trang có một bộ đếm (được giữ trong cấu trúc dữ liệu mem_map_t) cung cấp cho kswapd một số ý tưởng về việc một trang có đáng để hoán đổi hay không. Các trang cũ là các trang đã dùng và sau đó không được sử dụng và dùng lại khi truy cập; kswapd chỉ hoán đổi các trang cũ. Hành động mặc định khi một trang được cấp phát lần đầu tiên, là đặt tuổi ban đầu cho nó là 3. Mỗi lần nó được chạm vào, độ tuổi của nó được tăng lên 3 đến tối đa là 20. Mỗi khi kswapd chạy nó sẽ làm lão háo các trang, giảm dần tuổi của chúng bằng 1. Những hành động mặc định này có thể được thay đổi và vì lý do này, chúng (và các thông tin liên quan đến hoán đổi khác) được lưu trữ trong cấu trúc dữ liệu swap_control.

Nếu trang cũ (age = 0), kswapd sẽ xử lý nó thêm. Các dirty page là các trang có thể được hoán đổi. Tuy nhiên không phải tất cả các dirty page đều nhất thiết phải ghi chép vào tệp swap file. Mỗi vùng bộ nhớ ảo của một tiến trình có thể có thao tác hoán đổi riêng của nó (được trỏ bởi con trỏ vm_ops trong cấu trúc dữ liệu vm_area_struct) và phương thức đó được sử dụng. Nếu không, kswapd sẽ phân bổ một trang trong tệp swap file và ghi trang đó ra thiết bị đó.

Entry của bảng quản phân trang của trang được đánh dấu là invalid nhưng có chứa thông tin về các trang trong tệp swap file. Đó là 1 offset để có thể tìm kiếm trong tệp swap file vị trí các trang được chứa trong đó và chỉ báo cho biết tệp swap file nào được sử dụng. Với bất cứ cách hoán đổi nào được sử dụng, trang vật lý ban đầu vẫn được giải phóng bằng cách đưa nó vào vector free_area. Làm sạch các trang có thể bị loại bọ và đưa chúng vào free_area cho lần sử dụng tiếp theo.

Nếu các trang đã được hoán đổi hoặc bị hủy, kswapd sẽ lại sleep. Lần đánh thức tiếp theo nó sẽ xem xé tiến trình kế tiếp trong hệ thống. Bằng cách này, kswapd sẽ thực hiện dần dần việc xử lý các trang vật lý cho tới khi hệ thống dược cân bằng. Điều này đẹp hơn nhiều so với việc hoán đổi toàn bộ tiến trình.

9 Swap Cache

Khi hoán đổi các trang vào tệp swap file, Linux tránh ghi các trang nếu không cần thiết. Đôi khi một trang nằm trong cả tệp swap file lẫn trong bộ nhớ vật lý. Điều này xảy ra khi một trang đã được hoán đổi khỏi bộ nhớ sau đó được đưa trở lại bộ nhớ khi nó được truy cập lại bởi một tiến trình. Miễn là trang trong bộ nhớ không được chỉnh sửa, bản sao trong tệp swap file vẫn valid.

Linux sử dụng bộ nhớ swap cache để theo dõi các trang này. Bộ nhớ swap cache chứa danh sách các entry của bảng quản phân trang, mỗi entry chứa thông tin về mỗi trang vật lý trong hệ thống. Entry của bảng quản phân trang cho một trang đã được hoán đổi sẽ mô tả tệp swap file đang giữ trang đó cùng với vị trí của nó trong tệp swap file. Nếu entry của bộ nhớ swap cache khác 0, nó thể hiện rằng một trang đang được giữ trong tệp swap file chưa được sửa đổi. Nếu trang được sửa đổi, entry của nó sẽ bị xóa bộ nhớ swap cache.

Khi Linux cần hoán đổi một trang vật lý sang tệp swap file, nó sẽ tham khảo bộ nhớ swap cache và nếu có entry là valid cho trang này, nó không cần ghi trang ra tệp swap file. Điều này là do trang trong bộ nhớ chưa được sửa đổi kể từ lần đọc cuối cùng từ tệp swap file.

Các entry trong bộ nhớ swap cache là các entry của bảng quản phân trang cho các trang được hoán đổi. Chúng được đánh dấu là invalid và chứa thông tin cho phép Linux tìm đúng tệp swap file và đúng trang trong tệp swap file đó.

10 Swap in trang

Các dirty page được lưu trong tệp swap file có thể được cần lại, chẳng hạn như khi một ứng dụng ghi vào vùng bộ nhớ ảo có nội dung được giữ trong một trang vật lý đã hoán đổi. Việc truy cập một trang của bộ nhớ ảo không được giữ trong bộ nhớ vật lý gây ra lỗi trang. Lỗi trang là bộ xử lý báo hiệu hệ điều hành rằng nó không thể dịch một địa chỉ ảo thành một địa chỉ vật lý. Trong trường hợp này là do entry của bảng quản phân trang mô tả trang bộ nhớ ảo này đã bị đánh dấu là invalid khi trang bị hoán đổi. Bộ xử lý không thể xử lý việc dịch địa chỉ ảo sang vật lý và do đó địa chỉ ảo bị lỗi và lý do gây ra lỗi.

Code xử lý lỗi trang cụ thể của bộ xử lý phải xác định nơi cấu trúc dữ liệu vm_area_struct cái mà mô tả vùng bộ nhớ ảo có chứa địa chỉ ảo bị lỗi. Nó thực hiện điều này bằng cách tìm kiếm cấu trúc dữ liệu vm_area_struct cho quá trình này cho đến khi nó tìm thấy cấu trúc chứa địa chỉ ảo bị lỗi. Đây là code rất quan trọng về thời gian và cấu trúc dữ liệu vm_area_struct của tiến trình được sắp xếp để làm

cho việc tìm kiếm này mất ít thời gian nhất có thể.

Sau khi thực hiện các hành động cụ thể của bộ xử lý thích hợp và nhận thấy rằng địa chỉ ảo bị lỗi là dành cho một vùng hợp lệ của bộ nhớ ảo, việc xử lý lỗi trang trở nên chung và có thể áp dụng cho tất cả các bộ xử lý Linux chạy trên đó.

Mã xử lý lỗi trang tìm kiếm entry của bảng quản phân trang trang cho địa chỉ ảo bị lỗi. Nếu entry của bảng quản phân trang mà nó tìm thấy là một trang đã được hoán đổi, Linux phải hoán đổi trang đó trở lại bộ nhớ vật lý. Định dạng của entry của bảng quản phân trang cho một trang được hoán đổi là dành riêng cho bộ xử lý nhưng tất cả bộ xử lý đánh dấu các trang này là invalid và đưa thông tin cần thiết để xác định trang trong tệp swap file vào entry của bảng quản phân trang. Linux cần thông tin này để đưa trang trở lại bộ nhớ vật lý.

Tại thời điểm này, Linux biết địa chỉ ảo bị lỗi và có một entry của bảng quản phân trang chứa thông tin về nơi trang này đã được hoán đổi. Các cấu trúc dữ liệu vm_area_struct có thể chứa một con trỏ đến một nơi cố định mà sẽ hoán đổi bất kỳ trang nào của khu vực bộ nhớ ảo mà nó mô tả lại vào bộ nhớ vật lý. Đây là hoạt động hoán đổi của nó . Nếu có một hoạt động hoán đổi cho vùng bộ nhớ ảo này thì Linux sẽ sử dụng nó. Trên thực tế, đây là cách xử lý các trang của vùng nhớ dùng chung System V được hoán đổi vì nó yêu cầu xử lý đặc biệt vì định dạng của trang được hoán đổi trong vùng nhớ dùng chung System V hơi khác so với định dạng của trang được hoán đổi thông thường. Có thể hoán đổi không hoạt động, trong trường hợp đó Linux sẽ cho rằng đây là một trang đã hết hạn sử dụng không cần được xử lý đặc biệt.

Linux phân bổ một trang vật lý tự do và đọc lại trang được hoán đổi từ tệp swap file. Thông tin cho nó biết vị trí trong tệp swap file (và tệp swap file nào) được lấy từ entry của bảng quản phân trang.

Nếu quyền truy cập gây ra lỗi trang không phải là quyền truy cập ghi thì trang đó được để lại trong bộ nhớ swap cache và entry của bảng quản phân trang của nó không được đánh dấu là có thể ghi. Nếu sau đó trang được ghi vào, một lỗi trang khác sẽ xảy ra và tại thời điểm đó, trang bị đánh dấu là dirty và entry của nó sẽ bị xóa khỏi bộ nhớ swap cache. Nếu trang không được ghi vào và nó cần được hoán đổi lại, Linux có thể tránh việc ghi trang vào tệp swap file của nó vì trang đã nằm trong tệp swap file.

Nếu quyền truy cập khiến trang được đưa vào từ tệp swap file là hoạt động ghi, trang này sẽ bị xóa khỏi bộ nhớ swap cache và entry của bảng quản trang của nó được đánh dấu là dirty và có thể ghi.

11 Tổng kết

Quá trình tìm hiểu về cách quản lý bộ nhớ trong hệ điều hành Linux đã giúp chúng em mở mang kiến thức rất nhiều, dồng thời đó là củng cố lại những nội dung về phương thức quản lý bộ nhớ được giới thiệu ở môn Nguyên lý hệ điều hành. Toàn bộ nội dung của bài báo cáo được chúng em tìm hiểu tại [1].

Để hoàn thành được báo cáo này, nhóm chúng em xin tỏ lòng biết ơn sâu sắc đến thầy Phạm Đăng Hải, cảm ơn thầy vì những tiết dạy tuyệt vời và đáng quý của thầy ở môn Nguyên lý hệ điều hành.

Tài liệu

[1] Duke University. *Linux Memory Management.* https://tldp.org/LDP/tlk/mm/memory.html.