40

Triển khai hệ thống tệp

Trong chương này, chúng tôi giới thiệu một triển khai hệ thống tệp đơn giản, được gọi là vsfs (Hệ thống tệp rất đơn giản). Hệ thống tệp này là phiên bản đơn giản hóa của hệ thống tệp UNIX điển hình và do đó dùng để giới thiệu một số cấu trúc cơ bản trên đĩa, phương pháp truy cập và các chính sách khác nhau mà bạn sẽ tìm thấy trong nhiều hệ thống tệp ngày nay.

Hệ thống tệp là phần mềm thuần túy; không giống như sự phát triển của chúng tôi về ảo hóa CPU và bộ nhớ, chúng tôi sẽ không thêm các tính năng phần cứng để làm cho một số khía cạnh của hệ thống tệp hoạt động tốt hơn (mặc dù chúng tôi sẽ muốn chú ý đến các đặc điểm của thiết bị để đảm bảo hệ thống tệp hoạt động tốt).
Do tính linh hoạt cao mà chúng tôi có trong việc xây dựng hệ thống tệp, nhiều hệ thống tệp khác nhau đã được xây dựng, từ AFS (tem Andrew File Sys) [H + 88] đến ZFS (Hệ thống tệp Zettabyte của Sun) [B07]. Tất cả các hệ thống tệp này có cấu trúc dữ liệu khác nhau và thực hiện một số thứ tốt hơn hoặc kém hơn so với các hệ thống tệp khác của chúng. Do đó, cách chúng ta sẽ tìm hiểu về hệ thống tệp là thông qua các nghiên cứu điển hình: đầu tiên, một hệ thống tệp đơn giản (vsfs) trong chương này để giới thiệu hầu hết các khái niệm, và sau đó là một loạt các nghiên cứu về hệ thống tệp thực để hiểu chúng có thể khác nhau như thế nào thực tiễn.

THE CRUX: CÁCH THỰC HIỆN HỆ THỐNG TẬP TIN ĐƠN GIẢN

Làm thế nào chúng ta có thể xây dựng một hệ thống tệp đơn giản? Những cấu trúc nào là cần thiết trên đĩa? Họ cần theo dõi những gì? Chúng được truy cập như thế nào?

40.1 Cách suy nghĩ

Để nghĩ về hệ thống tệp, chúng tôi thường khuyên bạn nên suy nghĩ về hai khía cạnh khác nhau của chúng; nếu bạn hiểu cả hai khía cạnh này, bạn có thể hiểu về cơ bản hệ thống tệp hoạt động như thế nào.

Đầu tiên là cấu trúc dữ liệu của hệ thống tệp. Nói cách khác, hệ thống tệp sử dụng những loại cấu trúc trên đĩa nào để tổ chức dữ liệu và siêu dữ liệu của nó? Hệ thống tệp đầu tiên chúng ta sẽ thấy (bao gồm vsfs bên dưới) sử dụng các cấu trúc đơn giản, như mảng khối hoặc các đối tượng khác, trong khi

ASIDE: CÁC MÔ HÌNH TÂM LÝ CỦA HỆ THỐNG FILE

Như chúng ta đã thảo luận trước đây, các mô hình tinh thần là những gì bạn đang thực sự cố gắng để phát triển khi tìm hiểu về hệ thống. Đối với hệ thống tệp, tinh thần của bạn mô hình cuối cùng nên bao gồm câu trả lời cho các câu hỏi như: cái gì trên đĩa cấu trúc lưu trữ dữ liệu và siêu dữ liệu của hệ thống tệp? Chuyện gì xảy ra khi một quá trình mở một tệp? Những cấu trúc trên đĩa nào được truy cập trong một đọc hay viết? Bằng cách nỗ lực và cải thiện mô hình tinh thần của mình, bạn phát triển sự hiểu biết trừu tượng về những gì đang diễn ra, thay vì chỉ cố gắng hiểu chi tiết cụ thể của một số mã hệ thống tệp (mặc dù điều đó tất nhiên cũng hữu ích!).

các hệ thống tệp phức tạp hơn, như XFS của SGI, sử dụng phức tạp hơn cấu trúc dựa trên cây [S + 96].

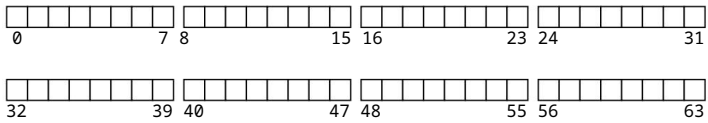
Khía cạnh thứ hai của hệ thống tệp là các phương thức truy cập của nó. Làm thế nào nó ánh xạ các cuộc gọi được thực hiện bởi một quá trình, chẳng hạn như open (), read (), write (), vv, vào cấu trúc của nó? Cấu trúc nào được đọc trong quá trình thực thi của một lệnh gọi hệ thống cụ thể? Cái nào được viết? Hiệu quả của tất cả các bước này được thực hiện?

Nếu bạn hiểu cấu trúc dữ liệu và phương pháp truy cập của hệ thống tệp, bạn đã phát triển một mô hình tinh thần tốt về cách nó thực sự hoạt động, a phần quan trọng của tư duy hệ thống. Cố gắng làm việc để phát triển trí não của bạn mô hình khi chúng tôi đi sâu vào triển khai đầu tiên của mình.

40.2 Tổ chức chung

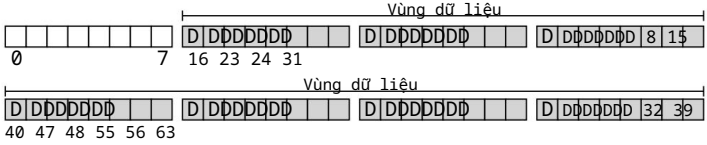
Bây giờ chúng tôi phát triển tổ chức tổng thể trên đĩa của các cấu trúc dữ liệu của hệ thống tệp vsfs. Điều đầu tiên chúng ta cần làm là chia đĩa thành khối; hệ thống tệp đơn giản chỉ sử dụng một kích thước khối và đó là chính xác những gì chúng tôi sẽ làm ở đây. Hãy chọn kích thước thường được sử dụng là 4 KB.

Do đó, chế độ xem của chúng tôi về phân vùng đĩa nơi chúng tôi đang xây dựng tem hệ thống tệp của mình rất đơn giản: một loạt các khối, mỗi khối có kích thước 4 KB. Các khối là quảng cáo được xếp từ 0 đến N - 1, trong một phân vùng có kích thước N khối 4-KB. Giả sử chúng tôi có một đĩa thực sự nhỏ, chỉ với 64 khối:



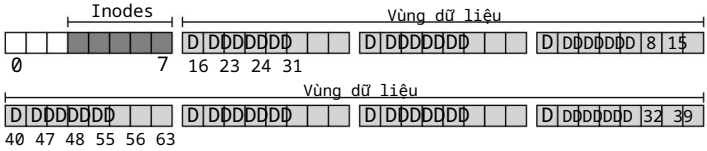
Bây giờ chúng ta hãy nghĩ về những gì chúng ta cần lưu trữ trong các khối này để xây dựng một hệ thống tệp. Tất nhiên, điều đầu tiên xuất hiện trong tâm trí là dữ liệu người dùng. Trên thực tế, hầu hết không gian trong bất kỳ hệ thống tệp nào là (và nên là) dữ liệu người dùng. Hãy gọi vùng đĩa mà chúng ta sử dụng cho dữ liệu người dùng là vùng dữ liệu và

một lần nữa để đơn giản, hãy dành một phần cố định của đĩa cho các khối này, chẳng hạn như 56 trong số 64 khối cuối cùng trên đĩa:



Như chúng ta đã tìm hiểu về (một chút) chương trước, hệ thống tệp phải theo dõi thông tin về từng tệp. Thông tin này là một phần quan trọng của siêu dữ liệu và theo dõi những thứ như khối dữ liệu nào (trong vùng dữ liệu) bao gồm một tệp, kích thước của tệp, chủ sở hữu và quyền truy cập, thời gian truy cập và sửa đổi cũng như các loại thông tin tương tự khác. Để lưu trữ thông tin này, hệ thống tệp thường có cấu trúc được gọi là inode (chúng ta sẽ đọc thêm về inode bên dưới).

Để chứa các inodes, chúng tôi cũng cần phải dành một số dung lượng trên đĩa cho chúng. Hãy gọi phần này của đĩa là bảng inode, chỉ đơn giản là chứa một mảng các inode trên đĩa. Do đó, hình ảnh trên đĩa của chúng tôi bây giờ trông giống như hình này, giả sử rằng chúng tôi sử dụng 5 trong số 64 khối của chúng tôi cho các inodes (được biểu thị bằng I trong sơ đồ):

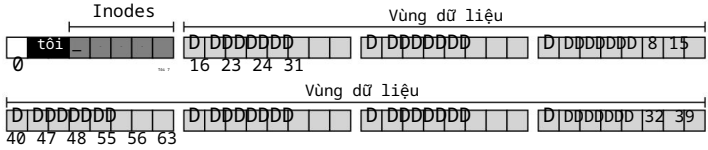


Chúng ta nên lưu ý ở đây rằng các inodes thường không lớn như vậy, ví dụ 128 hoặc 256 byte. Giả sử 256 byte cho mỗi inode, khối 4 KB có thể chứa 16 inode và hệ thống tệp của chúng tôi ở trên chứa tổng số 80 inode. Trong hệ thống tệp đơn giản của chúng tôi, được xây dựng trên một phân vùng 64 khối nhỏ, con số này đại diện cho số tệp tối đa mà chúng tôi có thể có trong hệ thống tệp của mình; tuy nhiên, hãy lưu ý rằng cùng một hệ thống tệp, được xây dựng trên đĩa lớn hơn, có thể chỉ cần phân bổ một bảng inode lớn hơn và do đó chứa được nhiều tệp hơn.

Hệ thống tệp của chúng tôi cho đến nay có các khối dữ liệu (D) và inodes (I), nhưng một số thứ vẫn còn thiếu. Một thành phần chính vẫn cần thiết, như bạn có thể đoán, là một số cách để theo dõi xem các inodes hoặc khối dữ liệu là miễn phí hay được cấp phát. Do đó, cấu trúc phân bổ như vậy là một yếu tố cần thiết trong bất kỳ hệ thống tệp nào.

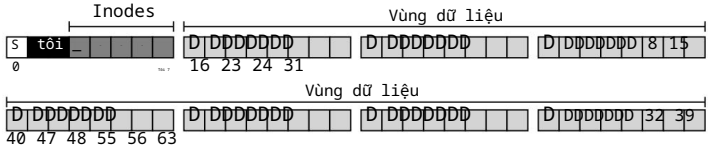
Tất nhiên là có thể có nhiều phương pháp theo dõi phân bổ. Đối với đề thi, chúng ta có thể sử dụng danh sách miễn phí trỏ đến khối miễn phí đầu tiên, sau đó trỏ đến khối miễn phí tiếp theo, v.v. Thay vào đó, chúng tôi chọn một cấu trúc đơn giản và phổ biến được gọi là bitmap, một cấu trúc cho vùng dữ liệu (dữ liệu bitmap) và một cấu trúc cho bảng inode (bitmap inode). Một bitmap là một

cấu trúc đơn giản: mỗi bit được sử dụng để cho biết đối tượng / khối tương ứng là miễn phí (0) hay đang được sử dụng (1). Và do đó, bố cục trên đĩa mới của chúng tôi, với một bitmap inode (i) và một bitmap dữ liệu (d):



Bạn có thể nhận thấy rằng hơi quá mức cần thiết khi sử dụng toàn bộ khối 4 KB cho các ảnh bitmap này; một bitmap như vậy có thể theo dõi xem 32K đối tượng có được cấp phát hay không, tuy nhiên chúng ta chỉ có 80 inodes và 56 khối dữ liệu. Tuy nhiên, chúng tôi chỉ sử dụng toàn bộ khối 4 KB cho mỗi bitmap này để đơn giản hóa.

Người đọc cẩn thận (tức là người đọc vẫn còn tỉnh táo) có thể không bận tâm rằng vẫn còn một khối trong thiết kế cấu trúc trên đĩa của hệ thống tệp rất đơn giản của chúng ta. Chúng tôi dành điều này cho siêu khối, được ký hiệu bằng chữ S trong sơ đồ bên dưới. Siêu khối chứa thông tin về hệ thống tệp cụ thể này, bao gồm, ví dụ: có bao nhiêu inode và khối dữ liệu trong hệ thống tệp (tương ứng là 80 và 56 trong trường hợp này), nơi bắt đầu bảng inode (khối 3), v.v. . Nó có thể cũng sẽ bao gồm một số phép thuật nào đó để xác định loại hệ thống tệp (trong trường hợp này là vsfs).



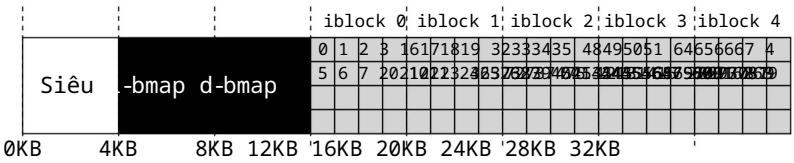
Do đó, khi gắn hệ thống tệp, hệ điều hành sẽ đọc siêu khối trước tiên, để khởi tạo các tham số khác nhau, sau đó đính kèm ổ đĩa vào cây hệ thống tệp. Khi các tệp trong ổ đĩa được truy cập, hệ thống sẽ biết chính xác nơi cần tìm các cấu trúc trên đĩa cần thiết.

40.3 Tổ chức tệp: Inode

Một trong những cấu trúc trên đĩa quan trọng nhất của hệ thống tệp là inode; hầu như tất cả các hệ thống tệp đều có cấu trúc tương tự như thế này. Tên inode là viết tắt của nút chỉ mục, tên lịch sử được đặt cho nó trong UNIX [RT74] và có thể là các hệ thống trước đó, được sử dụng vì các nút này được sắp xếp ban đầu trong một mảng và mảng được lập chỉ mục khi truy cập vào một inode cụ thể.

ASIDE : CẤU TRÚC DỮ LIỆU - INODE inode là
tên chung được sử dụng trong nhiều hệ thống tệp để xác định cấu trúc lưu giữ siêu dữ liệu cho một tệp nhất định, chẳng hạn như độ dài, quyền và vị trí của các khối cấu thành của nó . Tên quay trở lại ít nhất là từ UNIX (và có thể trở lại với Multics nếu không phải là các hệ thống trước đó); nó là viết tắt của nút chỉ mục, vì số inode được sử dụng để lập chỉ mục vào một mảng các inode trên đĩa để tìm inode của số đó. Như chúng ta sẽ thấy, thiết kế của inode là một phần quan trọng của thiết kế hệ thống tệp. Hầu hết các hệ thống hiện đại đều có một số loại cấu trúc như thế này cho mọi tệp mà chúng theo dõi, nhưng có thể gọi chúng là những thứ khác nhau (chẳng hạn như dnodes, fnodes, v.v.).

Mỗi inode được tham chiếu ngầm bởi một số (được gọi là số i), mà chúng ta đã gọi trước đó là tên cấp thấp của tệp. Trong vsfs (và các hệ thống tệp đơn giản khác), với một số i, bạn sẽ có thể trực tiếp tính toán vị trí của inode tương ứng trên đĩa. Để có nhiều thông tin, hãy lấy bảng inode của vsfs như trên: kích thước 20KB (năm khối 4KB) và do đó bao gồm 80 inode (giả sử mỗi inode là 256 byte); fur ther giả định rằng vùng inode bắt đầu ở 12KB (tức là siêu khối bắt đầu ở 0KB, bitmap inode ở địa chỉ 4KB, bitmap dữ liệu ở 8KB và do đó bảng inode xuất hiện ngay sau đó). Trong vsfs, do đó, chúng tôi có bố cục sau cho phần đầu của phân vùng hệ thống tệp (trong chế độ xem cận cảnh): Bảng Inode (Closeup)



Để đọc số inode 32, trước tiên hệ thống tệp sẽ tính toán bộ tắt vào vùng inode (32 · sizeof (inode) hoặc 8192), thêm nó vào địa chỉ bắt đầu của bảng inode trên đĩa (inodeStartAddr = 12KB), và do đó đến đúng địa chỉ byte của khối inodes mong muốn: 20KB.
Nhớ lại rằng các đĩa không thể định địa chỉ theo byte, mà bao gồm một số lượng lớn các khu vực có thể định địa chỉ, thường là 512 byte. Do đó, để tìm nạp khối inode có chứa inode 32, hệ thống tệp sẽ đưa ra giá trị đọc đến $\frac{20 \times 1024}{512}$, giây hoặc 40, để tìm nạp khối inode mong muốn. Nói chung, khu vực địa chỉ sector của khối inode có thể được tính như sau:

```
blk       = (inumber * sizeof (inode_t)) / blockSize;
sector = ((blk * blockSize) + inodeStartAddr) / sectorSize; Bên trong mỗi inode
```

hầu như là tất cả thông tin bạn cần về một tệp: loại của nó (ví dụ: tệp thông thường, thư mục, v.v.), kích thước của nó, số khối được phân bổ cho nó, thông tin bảo vệ (chẳng hạn như ai sở hữu tệp, cũng

Kích thước	Tên	Trường inode này dùng để làm gì?
2	chế độ	tập tin này có thể được đọc / ghi / thực thi?
2	uid	ai sở hữu tập tin này?
4	kích thước	có bao nhiêu byte trong tệp này?
4	time	tệp này được truy cập lần cuối vào lúc nào?
4	ctime	tệp này được tạo lúc mấy giờ?
4	giờ	tập tin này được sửa đổi lần cuối vào lúc mấy giờ?
4	dtime	inode này bị xóa lúc mấy giờ?
2	gid	tập tin này thuộc nhóm nào?
2	liên kết	đếm xem có bao nhiêu liên kết cứng đến tệp này?
4	khối	có bao nhiêu khối đã được phân bổ cho tệp này?
4	cờ	làm thế nào để ext2 sử dụng inode này?
4	osd1	một trường phụ thuộc vào hệ điều hành
60	khối	một bộ con trỏ đĩa (tổng số 15)
		Phiên bản tệp 4 thế hệ (được sử dụng bởi NFS)
4	tập tin_acl	một mô hình quyền mới ngoài bit chế độ
4	dir acl_	được gọi là danh sách kiểm soát truy cập

Hình 40.1: Inode Ext2 đơn giản hóa

như ai có thể truy cập nó), một số thông tin về thời gian , bao gồm cả thời điểm tệp được tạo, sửa đổi hoặc truy cập lần cuối cũng như thông tin về nơi các khối dữ liệu nằm trên đĩa (ví dụ: con trỏ của một số loại). Chúng tôi tham khảo tất cả thông tin đó về tệp dưới dạng siêu dữ liệu; trên thực tế, bất kỳ thông tin nào bên trong hệ thống tệp không phải là dữ liệu người dùng thuần túy thường được gọi như vậy. Một ví dụ inode từ ext2 [P09] được hiển thị trong Hình 40.11 .

Một trong những quyết định quan trọng nhất trong việc thiết kế inode là làm thế nào nó đề cập đến vị trí của các khối dữ liệu. Một cách tiếp cận đơn giản là có một hoặc nhiều con trỏ trực tiếp (địa chỉ đĩa) bên trong inode; mỗi con trỏ đề cập đến một khối đĩa thuộc tệp. Tiếp câ n như vâ y bị giới hạn: ví dụ: nếu bạn muốn có một tệp thực sự lớn (ví dụ: lớn hơn kích thước khối nhân với số lượng con trỏ trực tiếp trong inode), bạn không may mắn.

Chỉ số đa cấp

Để hỗ trợ các tệp lớn hơn, các nhà thiết kế hệ thống tệp đã phải giới thiệu các cấu trúc khác nhau trong inodes. Một ý tưởng phổ biến là có một con trỏ được biết đến như một con trỏ gián tiếp. Thay vì trỏ đến một khối chứa dữ liệu người dùng, nó trỏ đến một khối chứa nhiều con trỏ hơn, mỗi trong đó trỏ tới dữ liệu người dùng. Do đó, một inode có thể có một số số cố định con trỏ trực tiếp (ví dụ: 12) và một con trỏ gián tiếp. Nếu một tập tin phát triển đủ lớn, một khối gián tiếp được phân bổ (từ khu vực khối dữ liệu của đĩa) và khe cắm của inode cho một con trỏ gián tiếp được đặt để trỏ tới nó. Giả sử các khối 4 KB và địa chỉ đĩa 4 byte, điều đó sẽ thêm 1024 khác con trỏ; tệp có thể phát triển thành (12 + 1024) · 4K hoặc 4144KB.

1Type thông tin được giữ trong mục nhập thư mục và do đó không được tìm thấy trong chính inode.

MẸO: XÉT DUYỆT CÁC PHƯƠNG PHÁP TIẾP CẬN BÊN NGOÀI

Một cách tiếp cận khác là sử dụng các khoảng mở rộng thay vì các con trỏ. Một mức độ chỉ đơn giản là một con trỏ đĩa cộng với chiều dài (tính bằng khối); do đó, thay vì yêu cầu một con trỏ cho mọi khối tệp, tất cả những gì người ta cần là một con trỏ và độ dài để chỉ định vị trí trên đĩa của tệp. Chỉ một mức độ duy nhất là hạn chế, vì người ta có thể gặp khó khăn khi tìm một phần liền kề của không gian trống trên đĩa khi phân bổ tệp. Do đó, các hệ thống tệp dựa trên mức độ thường cho phép nhiều hơn một mức độ, do đó mang lại nhiều tự do hơn cho hệ thống tệp trong quá trình cấp phát tệp.

Khi so sánh hai cách tiếp cận, cách tiếp cận dựa trên con trỏ là linh hoạt nhất nhưng sử dụng một lượng lớn siêu dữ liệu trên mỗi tệp (đặc biệt đối với các tệp lớn). Các phương pháp tiếp cận dựa trên phạm vi ít linh hoạt hơn nhưng nhỏ gọn hơn; nói chung, chúng hoạt động tốt khi có đủ dung lượng trống trên đĩa và các tệp có thể được sắp xếp liền nhau (đó là mục tiêu cho hầu như bất kỳ chính sách phân bổ tệp nào).

Không có gì đáng ngạc nhiên, trong cách tiếp cận như vậy, bạn có thể muốn hỗ trợ các tệp lớn hơn. Để làm như vậy, chỉ cần thêm một con trỏ khác vào inode: con trỏ gián tiếp dou ble. Con trỏ này đề cập đến một khối chứa các con trỏ đến các khối gián tiếp, mỗi khối chứa các con trỏ đến các khối dữ liệu. Do đó, một khối gián tiếp dou ble tăng thêm khả năng phát triển tệp với thêm 1024 · 1024 hoặc 1 triệu khối 4KB, nói cách khác là hỗ trợ các tệp có kích thước trên 4GB. Tuy nhiên, bạn có thể muốn nhiều hơn nữa, và chúng tôi cá là bạn biết điều này đang đi đến đâu: con trỏ gián tiếp ba.

Nhìn chung, cây không cân bằng này được gọi là ap chỉ mục đa cấp để trỏ tới các khối tệp. Hãy xem xét một ví dụ với mười hai con trỏ trực tiếp, cũng như cả một khối gián tiếp đơn và một khối gián tiếp kép. Vì kích thước khối 4 KB và con trỏ 4 byte, cấu trúc này có thể chứa một tệp có kích thước chỉ hơn 4 GB (tức là (12 + 1024 + 10242) × 4 KB).
Bạn có thể tìm ra mức độ lớn của một tệp có thể được xử lý với việc bổ sung một khối gián tiếp ba lần không? (gợi ý: khá lớn)

Nhiều hệ thống tệp sử dụng chỉ mục nhiều cấp, bao gồm các hệ thống tệp thường được sử dụng như Linux ext2 [P09] và ext3, WAFL của NetApp, cũng như hệ thống tệp UNIX gốc. Các hệ thống tệp khác, bao gồm SGI XFS và Linux ext4, sử dụng các phần mở rộng thay vì các con trỏ đơn giản; xem phần trước sang một bên để biết chi tiết về cách thức hoạt động của các lược đồ dựa trên mức độ (chúng giống với các phân đoạn trong cuộc thảo luận về bộ nhớ ảo).

Bạn có thể tự hỏi: tại sao lại sử dụng một cái cây không cân đối như thế này? Tại sao không phải là một cách tiếp cận khác? Hóa ra, nhiều nhà nghiên cứu đã nghiên cứu các hệ thống tệp và cách chúng được sử dụng, và hầu như mỗi khi họ tìm thấy một số "chân lý" nhất định tồn tại trong nhiều thập kỷ. Một phát hiện như vậy là hầu hết các tệp đều nhỏ. Thiết kế không cân đối này phản ánh một thực tế như vậy; nếu hầu hết các tệp thực sự nhỏ, thì nên tối ưu hóa cho trường hợp này. Do đó, với một số lượng nhỏ các con trỏ trực tiếp (12 là một số điển hình), một inode

Hầu hết các tệp đều nhỏ ~ 2K là kích thước phổ biến nhất	
Kích thước tệp trung bình đang tăng lên	Gần 200K là mức trung bình
Hầu hết các byte được lưu trữ trong các tệp lớn Một vài tệp lớn sử dụng hầu hết dung lượng	
Hệ thống tệp chứa rất nhiều tệp Trung bình gần 100K	
Hệ thống tệp gần đầy một nửa Ngay cả khi đĩa phát triển, hệ thống tệp vẫn đầy ~ 50%	
Thư mục thường nhỏ	Nhiều người có ít mục nhập; phần lớn có 20 hoặc ít hơn

Hình 40.2: Tóm tắt đo lường hệ thống tệp

có thể trỏ trực tiếp đến 48 KB dữ liệu, cần một (hoặc nhiều) khối gián tiếp cho các tệp lớn hơn. Xem Agrawal et. al [A + 07] cho một nghiên cứu gần đây; Nhân vâ t 40.2 tóm tắt các kết quả đó.

Tất nhiên, trong không gian của thiết kế inode, nhiều khả năng khác ex ist; xét cho cùng, inode chỉ là một cấu trúc dữ liệu và bất kỳ cấu trúc dữ liệu nào lưu trữ thông tin liên quan và có thể truy vấn nó một cách hiệu quả là đủ. Vì phần mềm hệ thống tệp dễ dàng thay đổi, bạn nên sẵn sàng tìm hiểu các thiết kế khác nhau nếu khối lượng công việc hoặc công nghệ thay đổi.

40.4 Tổ chức thư mục

Trong vsfs (như trong nhiều hệ thống tệp), các thư mục có một tổ chức đơn giản; một thư mục về cơ bản chỉ chứa một danh sách các cặp (tên mục nhập, inode num ber). Đối với mỗi tệp hoặc thư mục trong một thư mục nhất định, có một chuỗi và một số trong (các) khối dữ liệu của thư mục. Đối với mỗi chuỗi, có cũng có thể là một độ dài (giả sử các tên có kích thước thay đổi).

Ví dụ: giả sử một thư mục dir (inode số 5) có ba tệp trong đó (foo, bar và foobar là một tên khá dài), với số inode lần lượt là 12, 13 và 24. Dữ liệu trên đĩa cho dir có thể giống như sau:

inum	reclen	strlen	Tên
	12	2	.
5 2	12		..
12	12	3 4	foo
13	12	4	quán ba
24	36	28	foobar_is_a_pretty_longname

Trong ví dụ này, mỗi mục nhập có một số inode, độ dài bản ghi (tổng số byte cho tên cộng với bất kỳ khoảng trống còn lại nào), độ dài chuỗi (thực tế độ dài của tên), và cuối cùng là tên của mục nhập. Lưu ý rằng mỗi nhà lãnh đạo có hai mục bổ sung ,. "Dot" và .. "dot-dot"; thư mục dấu chấm chỉ là thư mục hiện tại (trong ví dụ này là dir), trong khi dấu chấm là thư mục mẹ (trong trường hợp này là thư mục gốc).

Xóa một tệp (ví dụ: gọi unlink ()) có thể để lại một khoảng trống trong giữa thư mục và do đó cần có một số cách để đánh dấu điều đó cũng vậy (ví dụ: với một số inode dành riêng chẳng hạn như số không). Như một xóa là một lý do khiến độ dài bản ghi được sử dụng: một mục nhập mới có thể sử dụng lại một mục cũ, lớn hơn và do đó có thêm không gian bên trong.

BÊN NGOÀI: CÁC CÁCH TIẾP CẬN ĐƯỢC LIÊN KẾT

Một cách tiếp cận đơn giản hơn trong việc thiết kế inodes là sử dụng danh sách liên kết. Do đó, bên trong một inode, thay vì có nhiều con trỏ, bạn chỉ cần một con trỏ để trỏ đến khối đầu tiên của tệp. Để xử lý các tệp lớn hơn, hãy thêm một con trỏ khác vào cuối khối dữ liệu đó, v.v., do đó bạn có thể hỗ trợ các tệp lớn.

Như bạn có thể đoán, phân bổ tệp được liên kết hoạt động kém đối với một số khối lượng công việc; Ví dụ: hãy nghĩ đến việc đọc khối cuối cùng của tệp hoặc chỉ thực hiện truy cập ngẫu nhiên. Do đó, để phân bổ liên kết hoạt động tốt hơn, một số hệ thống sẽ giữ một bảng thông tin liên kết trong bộ nhớ, thay vì lưu trữ các con trỏ tiếp theo với chính các khối dữ liệu. Bảng được lập chỉ mục theo địa chỉ của khối dữ liệu D; nội dung của một mục nhập chỉ đơn giản là con trỏ tiếp theo của D. rằng một khối cụ thể là miễn phí. Việc có một bảng các con trỏ tiếp theo như vậy giúp cho một lược đồ phân bổ được liên kết có thể thực hiện truy cập tệp ngẫu nhiên một cách hiệu quả, đơn giản bằng cách quét qua bảng (trong bộ nhớ) trước tiên để tìm khối mong muốn, sau đó truy cập trực tiếp (trên đĩa) vào nó.

Một cái bàn như vậy nghe có quen không? Những gì chúng tôi đã mô tả là cấu trúc cơ bản của những gì được gọi là bảng phân bổ tệp, hoặc hệ thống tệp FAT .
Có, hệ thống tệp Windows cũ cổ điển này, trước NTFS [C94], dựa trên một sơ đồ phân bổ dựa trên liên kết đơn giản. Cũng có những điểm khác biệt so với hệ thống tệp UNIX tiêu chuẩn ; ví dụ: không có các inodes mà là các mục nhập thư mục lưu trữ siêu dữ liệu về một tệp và tham chiếu trực tiếp đến khối đầu tiên của tệp đã nói, điều này khiến cho việc tạo liên kết cứng không thể thực hiện được. Xem Brouwer [B02] để biết thêm các chi tiết không phù hợp.

Bạn có thể tự hỏi nơi chính xác các thư mục được lưu trữ. Thông thường, hệ thống tệp coi thư mục như một loại tệp đặc biệt. Do đó, một thư mục có một inode, ở đâu đó trong bảng inode (với trường loại của inode được đánh dấu là "thư mục" thay vì "tệp thông thường"). Thư mục có các khối dữ liệu được trỏ tới bởi inode (và có lẽ, các khối gián tiếp); các khối dữ liệu này nằm trong vùng khối dữ liệu của hệ thống tệp đơn giản của chúng tôi. Do đó, cấu trúc trên đĩa của chúng tôi vẫn không thay đổi.

Chúng ta cũng cần lưu ý lại rằng danh sách thư mục tuyến tính đơn giản này không phải là cách duy nhất để lưu trữ thông tin như vậy. Như trước đây, bất kỳ cấu trúc dữ liệu nào cũng có thể thực hiện được. Ví dụ: XFS [S + 96] lưu trữ các thư mục ở dạng B-tree, thực hiện các hoạt động tạo tệp (phải đảm bảo rằng tên tệp chưa được sử dụng trước khi tạo) nhanh hơn so với các hệ thống có danh sách đơn giản phải được quét trong toàn bộ.

BÊN TRONG: QUẢN LÝ KHÔNG GIAN MIỄN PHÍ Có nhiều cách để quản lý không gian trống; bitmap chỉ là một cách.
Một số hệ thống tệp đầu tiên sử dụng danh sách miễn phí, trong đó một con trỏ duy nhất trong siêu khối được giữ để trỏ đến khối miễn phí đầu tiên; bên trong khối đó, con trỏ miễn phí tiếp theo được giữ lại, do đó tạo thành một danh sách thông qua các khối miễn phí của hệ thống. Khi cần một khối, khối head đã được sử dụng và danh sách được cập nhật tương ứng.

Các hệ thống tệp hiện đại sử dụng cấu trúc dữ liệu phức tạp hơn. Ví dụ: XFS [S + 96] của SGI sử dụng một số dạng của cây B để thể hiện một cách gọn nhẹ phần nào của đĩa là trống. Như với bất kỳ cấu trúc dữ liệu nào, có thể có sự cân bằng không gian thời gian khác nhau.

40.5 Quản lý không gian trống

Hệ thống tệp phải theo dõi inodes và khối dữ liệu nào là miễn phí và không có, để khi một tệp hoặc thư mục mới được cấp phát, nó có thể tìm thấy không gian cho nó. Vì vậy, quản lý không gian trống là quan trọng đối với tất cả các hệ thống tệp.
Trong vsfs, chúng tôi có hai bitmap đơn giản cho tác vụ này.

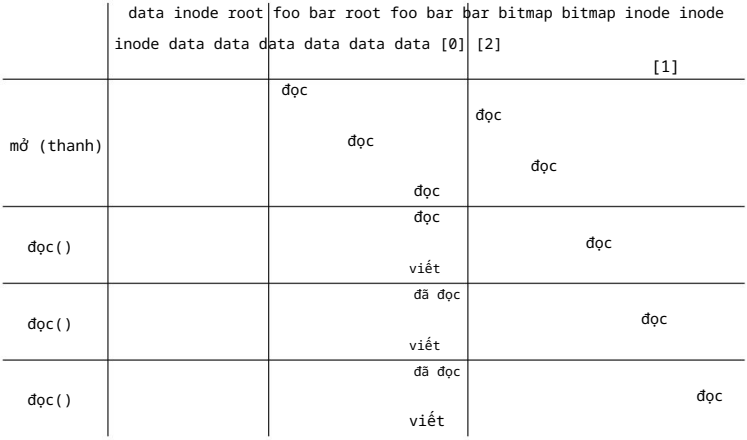
Ví dụ, khi chúng ta tạo một tệp, chúng ta sẽ phải cấp phát một inode cho tệp đó. Do đó, hệ thống tệp sẽ tìm kiếm thông qua bitmap cho một trong ode miễn phí và phân bổ nó cho tệp; hệ thống tệp sẽ phải đánh dấu inode là được sử dụng (với 1) và cuối cùng cập nhật bitmap trên đĩa với thông tin chính xác. Một tập hợp các hoạt động tương tự diễn ra khi một khối dữ liệu được cấp phát.

Một số cân nhắc khác cũng có thể được áp dụng khi phân bổ các khối dữ liệu cho một tệp mới. Ví dụ, một số hệ thống tệp Linux, chẳng hạn như ext2 và ext3, sẽ tìm kiếm một chuỗi các khối (ví dụ 8) miễn phí khi một tệp mới được tạo và cần các khối dữ liệu; bằng cách tìm một dãy các khối miễn phí như vậy, rồi phân bổ chúng vào tệp mới tạo, hệ thống tệp đảm bảo rằng một phần của tệp sẽ nằm liền kề trên đĩa, do đó cải thiện hiệu suất. Chính sách phân bổ trước như vậy là một phương pháp heuristic thường được sử dụng khi phân bổ không gian cho các khối dữ liệu.

40.6 Đường dẫn truy cập: Đọc và Viết

Bây giờ chúng ta đã có một số ý tưởng về cách tệp và thư mục được lưu trữ trên đĩa, chúng ta sẽ có thể theo dõi luồng hoạt động trong quá trình đọc hoặc ghi tệp. Do đó, hiểu những gì xảy ra trên đường dẫn truy cập này là chìa khóa thứ hai trong việc phát triển sự hiểu biết về cách thức hoạt động của hệ thống tệp; chú ý!

Đối với các ví dụ sau, chúng ta hãy giả sử rằng hệ thống tệp đã được gắn kết và do đó, siêu khối đã có trong bộ nhớ. Mọi thứ khác (tức là inodes, thư mục) vẫn còn trên đĩa.



Hình 40.3: Dòng thời gian đọc tệp (Thời gian tăng dần xuống)

Đọc tệp từ đĩa

Trong ví dụ đơn giản này, trước tiên chúng ta hãy giả sử rằng bạn chỉ muốn mở một tệp (ví dụ: / foo / bar), đọc nó rồi đóng nó lại. Đối với ví dụ đơn giản này, giả sử tệp chỉ có kích thước 12KB (tức là 3 khối).

Khi bạn thực hiện một lệnh gọi mở ("/ foo / bar", O RDONLY), hệ thống tệp trước tiên cần phải tìm inode cho thanh tệp, để có được một số hình thức cơ bản về tệp (thông tin quyền, kích thước tệp, v.v. .). Để làm như vậy, hệ thống tệp phải có thể tìm thấy inode, nhưng tất cả những gì nó có ngay bây giờ là tên đường dẫn đầy đủ. Hệ thống tệp phải duyệt qua tên đường dẫn và do đó xác định vị trí inode mong muốn.

Tất cả các đường truyền bắt đầu từ thư mục gốc của hệ thống tệp, trong thư mục gốc được gọi đơn giản là /. Vì vậy, điều đầu tiên FS sẽ đọc từ đĩa là inode của thư mục gốc. Nhưng inode này ở đâu? Để tìm một inode, chúng ta phải biết số i của nó. Thông thường, chúng ta tìm số thứ i của một tệp hoặc thư mục trong thư mục mẹ của nó; gốc không có cha (theo định nghĩa).
Do đó, số inode gốc phải được "biết rõ"; FS phải biết nó là gì khi hệ thống tệp được gắn kết. Trong hầu hết các hệ thống tệp UNIX , số inode gốc là 2. Do đó, để bắt đầu quá trình, FS đọc trong khối có chứa inode số 2 (khối inode đầu tiên).

Khi inode được đọc vào, FS có thể nhìn vào bên trong nó để tìm các con trỏ tới các khối dữ liệu, chứa nội dung của thư mục gốc. Do đó, FS sẽ sử dụng các con trỏ trên đĩa này để đọc qua thư mục, trong trường hợp này là tìm kiếm một mục nhập cho foo. Bằng cách đọc trong một hoặc nhiều khối dữ liệu thư mục, nó sẽ tìm thấy mục nhập cho foo; sau khi được tìm thấy, FS cũng sẽ tìm thấy số inode của foo (giả sử là 44) mà nó sẽ cần tiếp theo.

Bước tiếp theo là duyệt đệ quy tên đường dẫn cho đến khi tìm thấy inode mong muốn. Trong ví dụ này, FS đọc khối chứa

BÊN NGOÀI : CÁC BÀI ĐỌC KHÔNG TRUY CẬP CẤU TRÚC PHÂN PHỐI Chúng tôi đã thấy nhiều sinh viên bối rối bởi các cấu trúc phân bổ như bitmap. Đặc biệt, nhiều người thường nghĩ rằng khi bạn chỉ đọc một tệp và không phân bổ bất kỳ khối mới nào, thì bitmap vẫn sẽ được tham khảo. Đây không phải là sự thật! Cấu trúc phân bổ, chẳng hạn như bitmap, chỉ được truy cập khi cần phân bổ. Các inodes, thư mục và khối gián tiếp có tất cả thông tin chúng cần để hoàn thành nhiệm vụ đọc lại; không cần đảm bảo một khối được cấp phát khi inode đã trỏ đến nó.

inode của foo và sau đó là dữ liệu thư mục của nó, cuối cùng tìm thấy số inode của thanh. Bước cuối cùng của open () là đọc inode của thanh vào bộ nhớ; FS sau đó thực hiện kiểm tra quyền cuối cùng, cấp phát bộ mô tả tệp cho quá trình này trong bảng tệp mở cho mỗi quá trình và trả lại cho người dùng.

Sau khi mở, chương trình có thể thực hiện lệnh gọi hệ thống read () để đọc từ tệp. Lần đọc đầu tiên (ở độ lệch 0 trừ khi lseek () đã được gọi) do đó sẽ đọc trong khối đầu tiên của tệp, tham khảo ý kiến của inode để tìm vị trí của khối đó; nó cũng có thể cập nhật inode với thời gian mới được truy cập lần cuối. Quá trình đọc sẽ cập nhật thêm bảng tệp mở trong bộ nhớ cho bộ mô tả tệp này, cập nhật phần bù tệp sao cho lần đọc tiếp theo sẽ đọc khối tệp thứ hai, v.v.

Tại một thời điểm nào đó, tệp sẽ bị đóng. Có ít công việc hơn phải làm ở đây; rõ ràng, bộ mô tả tệp nên được phân bổ, nhưng hiện tại, đó là tất cả những gì FS thực sự cần làm. Không có đĩa I / Os nào diễn ra.

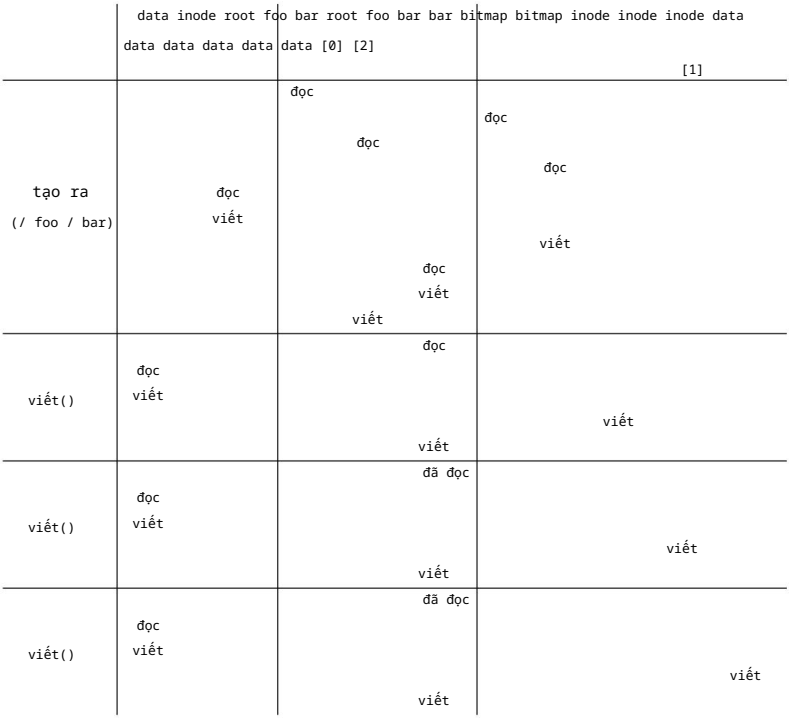
Mô tả toàn bộ quá trình này được tìm thấy trong Hình 40.3 (trang 11); thời gian tăng xuống trong hình. Trong hình, việc mở diễn ra nhiều lần đọc để cuối cùng xác định vị trí inode của tệp.
Sau đó, việc đọc từng khối yêu cầu hệ thống tệp trước tiên phải tham khảo inode, sau đó đọc khối và sau đó cập nhật trường thời gian truy cập cuối cùng của inode với một lần ghi. Dành một chút thời gian và hiểu những gì đang xảy ra.

Cũng lưu ý rằng số lượng I / O được tạo bởi open tương ứng với độ dài của tên đường dẫn. Đối với mỗi thư mục bổ sung trong đường dẫn, chúng ta phải đọc inode cũng như dữ liệu của nó. Làm cho điều này tồi tệ hơn sẽ là sự hiện diện của các thư mục lớn; ở đây, chúng ta chỉ phải đọc một khối để lấy nội dung của một thư mục, trong khi với một thư mục lớn, chúng ta có thể phải đọc nhiều khối dữ liệu để tìm được mục nhập mong muốn. Vâng, cuộc sống có thể trở nên khá tồi tệ khi đọc một tập tin; khi bạn sắp tìm ra, việc viết ra một tệp (và đặc biệt là tạo một tệp mới) thậm chí còn tệ hơn.

Ghi tệp vào đĩa Ghi tệp vào

một tệp là một quá trình tương tự. Đầu tiên, tệp phải được mở (như trên). Sau đó, ứng dụng có thể thực hiện lệnh gọi write () để cập nhật tệp với nội dung mới. Cuối cùng, tệp được đóng.

Không giống như đọc, ghi vào tệp cũng có thể phân bổ một khối (ví dụ: trừ khi khối đang bị ghi đè). Khi ghi ra một tệp mới, mỗi lần ghi không chỉ phải ghi dữ liệu vào đĩa mà trước tiên phải quyết định



Hình 40.4: Dòng thời gian tạo tệp (Thời gian tăng dần xuống)

chặn phân bổ cho tệp và do đó cập nhật các cấu trúc khác của đĩa cho phù hợp (ví dụ: bitmap dữ liệu và inode). Do đó, mỗi lần ghi vào tệp một cách hợp lý sẽ tạo ra năm I / Os: một để đọc bitmap dữ liệu (sau đó được cập nhật để đánh dấu khối mới được cấp phát là được sử dụng), một để ghi bitmap (để phản ánh trạng thái mới của nó vào đĩa) , thêm hai để đọc và sau đó ghi inode (được cập nhật với vị trí của khối mới), và cuối cùng là một để ghi chính khối thực.

Lưu lượng ghi thậm chí còn tồi tệ hơn khi người ta xem xét một sim

ple và hoạt động phổ biến như tạo tệp. Để tạo tệp, hệ thống tệp không chỉ phải cấp phát inode mà còn phải cấp phát không gian trong thư mục chứa tệp mới. Tổng lưu lượng I / O để làm như vậy là khá cao: một người đọc vào bitmap inode (để tìm một inode miễn phí), một người ghi vào bitmap inode (để đánh dấu nó được cấp phát), một người ghi vào chính inode mới (để khởi tạo nó), một vào dữ liệu của thư mục (để liên kết tên cấp cao của tệp với số inode của nó), và một đọc và ghi vào inode thư mục để cập nhật nó. Nếu thư mục cần phát triển để tiêu thụ mục nhập mới, thì I / Os bổ sung (tức là, cho bitmap dữ liệu và khối thư mục mới) cũng sẽ cần thiết. Tất cả những điều đó chỉ để tạo một tệp!

Hãy xem xét một ví dụ cụ thể, nơi tệp / foo / bar được tạo và ba khối được ghi vào nó. Hình 40.4 (trang 13) cho thấy những gì xảy ra trong quá trình mở () (tạo tệp) và trong mỗi ba lần ghi 4KB.

Trong hình, các lần đọc và ghi vào đĩa được nhóm lại theo đó lệnh gọi hệ thống khiến chúng xảy ra và thứ tự sơ bộ chúng có thể diễn ra theo thứ tự từ trên xuống dưới của hình. Bạn có thể thấy công việc tạo tệp: 10 I / O trong trường hợp này là đặt tên đường dẫn và cuối cùng là tạo tệp. Bạn cũng có thể thấy rằng mỗi lần ghi phân bổ tốn 5 I / O: một cặp để đọc và cập nhật inode, một cặp khác để đọc và cập nhật bitmap dữ liệu, và cuối cùng là ghi dữ liệu.

Làm thế nào một hệ thống tệp có thể thực hiện bất kỳ điều này với hiệu quả hợp lý?

THE CRUX: CÁCH GIẢM CHI PHÍ I / O HỆ THỐNG FILE Ngay cả những thao tác đơn giản nhất như mở, đọc hoặc ghi tệp cũng phải chịu một số lượng lớn các hoạt động I / O, nằm rải rác trên đĩa. Hệ thống tệp có thể làm gì để giảm chi phí cao khi thực hiện nhiều I / O như vậy?

40.7 Bộ nhớ đệm và đệm

Như các ví dụ trên cho thấy, việc đọc và ghi các tệp có thể rất nhanh, dẫn đến nhiều I / O cho đĩa (chậm). Để khắc phục những gì rõ ràng sẽ là một vấn đề hiệu suất lớn, hầu hết các hệ thống tệp tích cực sử dụng bộ nhớ hệ thống (DRAM) để lưu vào bộ đệm các khối quan trọng.

Hãy tưởng tượng ví dụ mở ở trên: không có bộ nhớ đệm, mọi tệp được mở sẽ yêu cầu ít nhất hai lần đọc cho mọi cấp trong hệ thống phân cấp thư mục (một để đọc inode của thư mục được đề cập và ít nhất một để đọc dữ liệu của nó). Với một tên đường dẫn dài (ví dụ: / 1/2/3 / ... /100/file.txt), hệ thống tệp thực sự sẽ thực hiện hàng trăm lần đọc chỉ để mở tệp!

Do đó, các hệ thống tệp ban đầu đã giới thiệu một bộ đệm có kích thước cố định để chứa các khối phổ biến. Như trong phần thảo luận của chúng tôi về bộ nhớ ảo, các chiến lược như LRU và các biến thể khác nhau sẽ quyết định khối nào cần lưu trong bộ nhớ cache. Bộ nhớ đệm có kích thước cố định này thường sẽ được cấp phát vào thời điểm khởi động bằng khoảng 10% tổng bộ nhớ.

Tuy nhiên, phân vùng tĩnh này của bộ nhớ có thể lãng phí; Điều gì sẽ xảy ra nếu hệ thống tệp không cần 10% bộ nhớ tại một thời điểm nhất định?

Với cách tiếp cận kích thước cố định được mô tả ở trên, các trang không sử dụng trong bộ đệm tệp không thể được sử dụng lại cho một số mục đích sử dụng khác, và do đó sẽ trở nên lãng phí.

Ngược lại, các hệ thống hiện đại sử dụng cách tiếp cận phân vùng động . Cụ thể, nhiều hệ điều hành hiện đại tích hợp các trang bộ nhớ ảo và các trang hệ thống tệp thành một bộ đệm trang thống nhất [S00]. Bằng cách này, bộ nhớ có thể được phân bổ linh hoạt hơn trên bộ nhớ ảo và hệ thống tệp, tùy thuộc vào bộ nhớ nào cần thêm bộ nhớ tại một thời điểm nhất định.

Bây giờ hãy tưởng tượng ví dụ mở tệp với bộ nhớ đệm. Lần mở đầu tiên có thể tạo ra nhiều lưu lượng I / O để đọc trong inode thư mục và dữ liệu, nhưng phụ

MẸO: HIỂU THỐNG KÊ VS. PHÂN BỐ NĂNG ĐỘNG Khi phân chia tài nguyên giữa các máy khách / người dùng khác nhau, bạn có thể sử dụng phân vùng tĩnh hoặc phân vùng động. Cách tiếp cận tĩnh chỉ đơn giản là chia tài nguyên thành các tỷ lệ cố định một lần; ví dụ: nếu có thể có hai người dùng bộ nhớ, bạn có thể cấp một phần cố định bộ nhớ cho một người dùng và phần còn lại cho người kia. Phương pháp tiếp cận năng động linh hoạt hơn, cung cấp các lượng tài nguyên khác nhau theo thời gian; ví dụ: một người dùng có thể nhận được phần trăm băng thông đĩa cao hơn trong một khoảng thời gian, nhưng sau đó, hệ thống có thể chuyển đổi và quyết định cung cấp cho người dùng khác một phần lớn hơn băng thông đĩa khả dụng.

Mỗi cách tiếp cận đều có lợi thế của nó. Phân vùng tĩnh đảm bảo mỗi người dùng nhận được một số chia sẻ tài nguyên, thường mang lại hiệu suất dễ dự đoán hơn và thường dễ thực hiện hơn. Phân vùng động có thể đạt được hiệu quả sử dụng tốt hơn (bằng cách cho phép người dùng đói tài nguyên sử dụng tài nguyên nhàn rỗi), nhưng có thể phức tạp hơn để triển khai và có thể dẫn đến hiệu suất kém hơn cho người dùng có tài nguyên nhàn rỗi bị người khác sử dụng và sau đó mất nhiều thời gian để đòi lại khi cần thiết. Như trong số mười trường hợp, không có phương pháp tốt nhất; thay vào đó, bạn nên suy nghĩ về vấn đề đang gặp phải và quyết định cách tiếp cận nào là phù hợp nhất. Thật vậy, không phải lúc nào bạn cũng nên làm như vậy?

các lần mở tệp tuần tự của cùng một tệp đó (hoặc các tệp trong cùng một thư mục) chủ yếu sẽ được truy cập trong bộ nhớ cache và do đó không cần I / O.

Chúng ta cũng hãy xem xét ảnh hưởng của bộ nhớ đệm đối với việc ghi. Trong khi I / O đọc có thể tránh được hoàn toàn với một bộ nhớ đệm đủ lớn, lưu lượng ghi phải vào đĩa để trở nên liên tục. Do đó, bộ nhớ cache không đóng vai trò như một loại bộ lọc trên lưu lượng ghi mà nó thực hiện đối với các lần đọc. Điều đó nói rằng, bộ đệm ghi (như đôi khi nó được gọi) chắc chắn có một số lợi ích cho mỗi công thức. Đầu tiên, bằng cách trì hoãn việc ghi, hệ thống tệp có thể đưa một số cập nhật vào một tập hợp I / Os nhỏ hơn ; ví dụ: nếu một bitmap inode được cập nhật khi một tệp được tạo và sau đó cập nhật ngay sau đó khi tệp khác được tạo, hệ thống tệp sẽ lưu I / O bằng cách trì hoãn việc ghi sau lần cập nhật đầu tiên. Thứ hai, bằng cách đệm một số lần ghi trong bộ nhớ, hệ thống sau đó có thể lên lịch cho I / Os tiếp theo và do đó tăng theo từng hình thức. Cuối cùng, một số bài viết được tránh hoàn toàn bằng cách trì hoãn chúng; ví dụ: nếu một ứng dụng tạo một tệp và sau đó xóa nó, thì việc trì hoãn việc ghi để phản ánh việc tạo tệp vào đĩa sẽ tránh hoàn toàn chúng. Trong trường hợp này, lười biếng (trong việc ghi các khối vào đĩa) là một đức tính tốt.

Vì những lý do trên, hầu hết bộ đệm của hệ thống tệp hiện đại sẽ ghi vào mem trong khoảng thời gian từ năm đến ba mươi giây, đại diện cho một sự đánh đổi khác: nếu hệ thống gặp sự cố trước khi các bản cập nhật được đưa vào đĩa, thì các bản cập nhật sẽ bị mất; tuy nhiên, bằng cách giữ các ghi trong mem lâu hơn, hiệu suất có thể được cải thiện bằng cách phân lô, lên lịch và thậm chí tránh ghi.

MẸO: HIỂU VỀ THƯƠNG MẠI ĐỘ DÀI / HIỆU SUẤT Hệ thống lưu trữ thường đánh đổi độ bền / hiệu suất đối với người dùng. Nếu người dùng muốn dữ liệu được ghi ngay lập tức lâu bền, hệ thống phải thực hiện toàn bộ nỗ lực cam kết dữ liệu mới được ghi vào đĩa, và do đó việc ghi chậm (nhưng an toàn). Tuy nhiên, nếu người dùng có thể chịu đựng được việc mất một ít dữ liệu, hệ thống có thể ghi vào bộ nhớ đệm một thời gian và ghi chúng sau đó vào đĩa (ở mặt đất sau). Làm như vậy làm cho các bài viết dường như hoàn thành nhanh chóng, do đó tôi chứng minh hiệu suất được nhận thức; tuy nhiên, nếu sự cố xảy ra, các ghi chưa được cam kết vào đĩa sẽ bị mất và do đó phải đánh đổi. Để hiểu cách thực hiện sự đánh đổi này một cách hợp lý, tốt nhất là hiểu những gì ứng dụng sử dụng hệ thống lưu trữ yêu cầu; ví dụ: mặc dù có thể chấp nhận được việc mất một vài hình ảnh cuối cùng được tải xuống bởi trình duyệt web của bạn, nhưng việc mất một phần của giao dịch cơ sở dữ liệu đang nạp thêm tiền vào tài khoản ngân hàng của bạn có thể ít được chấp nhận hơn. Tất nhiên, trừ khi bạn giàu có; trong trường hợp đó, tại sao bạn lại quan tâm nhiều đến việc tích trữ từng xu cuối cùng?

Một số ứng dụng (chẳng hạn như cơ sở dữ liệu) không được hưởng sự đánh đổi này. Do đó, để tránh mất dữ liệu không mong muốn do bộ đệm ghi, họ chỉ cần buộc ghi vào đĩa, bằng cách gọi fsync (), bằng cách sử dụng các giao diện I / O trực tiếp hoạt động xung quanh bộ đệm hoặc bằng cách sử dụng giao diện đĩa thô và tránh hệ thống tệp hoàn toàn 2 . Mặc dù hầu hết các ứng dụng sống với sự đánh đổi do hệ thống tệp thực hiện, nhưng có đủ các biện pháp kiểm soát để hệ thống thực hiện những gì bạn muốn, nếu mặc định không được đáp ứng.

40.8 Tóm tắt

Chúng tôi đã thấy những máy móc cơ bản cần thiết trong việc xây dựng một hệ thống tệp. Cần có một số thông tin về mỗi tệp (siêu dữ liệu), thường được lưu trữ trong cấu trúc gọi là inode. Thư mục chỉ là một loại tệp cụ thể lưu trữ tên ánh xạ số inode. Và các cấu trúc khác cũng cần thiết; ví dụ, hệ thống tệp thường sử dụng cấu trúc như bitmap để theo dõi inodes hoặc khối dữ liệu nào là miễn phí hoặc được cấp phát.

Khía cạnh tuyệt vời của thiết kế hệ thống tệp là sự tự do của nó; các hệ thống tệp mà chúng ta khám phá trong các chương tới, mỗi hệ thống sẽ tận dụng sự tự do này để tối ưu hóa một số khía cạnh của hệ thống tệp. Rõ ràng cũng có nhiều quyết định chính sách mà chúng tôi chưa khám phá. Ví dụ, khi một tệp mới được tạo, nó nên được đặt ở đâu trên đĩa? Chính sách này và các chính sách khác cũng sẽ là chủ đề của các chương trong tương lai. Hay họ sẽ? 3

2Tham gia một lớp học cơ sở dữ liệu để tìm hiểu thêm về các cơ sở dữ liệu cũ và ý thức trước đây của chúng trong việc tránh hệ điều hành và tự kiểm soát mọi thứ. Nhưng hãy coi chừng! Những kiểu cơ sở dữ liệu đó luôn cố gắng làm xấu hệ điều hành. Xấu hổ cho bạn, những người làm cơ sở dữ liệu. Xấu hổ.

Âm nhạc bí ẩn 3Cue khiến bạn bị hấp dẫn hơn nữa về chủ đề hệ thống tệp.

Người giới thiệu

[A + 07] “Nghiên cứu 5 năm về siêu dữ liệu hệ thống tệp” của Nitin Agrawal, William J. Bolosky, John R. Douceur, Jacob R. Lorch. FAST '07, San Jose, California, tháng 2 năm 2007. Một phân tích tuyệt vời gần đây về cách các hệ thống tệp thực sự được sử dụng. Sử dụng thư mục bên trong để theo dõi dấu vết của các bài báo phân tích hệ thống tệp từ đầu những năm 1980.

[B07] “ZFS: Lời cuối cùng trong hệ thống tệp” của Jeff Bonwick và Bill Moore. Có tại: http://www.ostep.org/Cences/zfs_last.pdf. Một trong những hệ thống tệp quan trọng gần đây nhất, có đầy đủ các tính năng và sự tuyệt vời. Chúng ta nên có một chương về nó, và có lẽ sẽ sớm thôi.

[B02] “Hệ thống tệp FAT” của Andries Brouwer. Tháng 9 năm 2002. Có trực tuyến tại: http://www.win.tue.nl/~ aeb/linux/fs/fat/fat.html. Một mô tả sạch đẹp về FAT. Loại hệ thống tệp, không phải loại thịt xông khói. Mặc dù bạn phải thừa nhận rằng mỡ thịt xông khói có lẽ ngon hơn.

[C94] “Bên trong Hệ thống tệp Windows NT” của Helen Custer. Microsoft Press, 1994. Một cuốn sách ngắn về NTFS; có thể có những cái với nhiều chi tiết kỹ thuật hơn ở những nơi khác.

[H + 88] “Quy mô và Hiệu suất trong Hệ thống Tệp Phân tán” của John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, Michael J. West. ACM TOCS, Tập 6: 1, tháng 2 năm 1988. Một hệ thống tệp phân tán cổ điển; chúng ta sẽ tìm hiểu thêm về nó sau, đừng lo lắng.

[P09] “Hệ thống tệp mở rộng thứ hai: Bố cục bên trong” của Dave Poirier. 2009. Có tại: http://www.nongnu.org/ext2-doc/ext2.html. Một số chi tiết về ext2, một hệ thống tệp Linux rất đơn giản dựa trên FFS, Hệ thống tệp nhanh Berkeley. Chúng ta sẽ đọc về nó trong chương tiếp theo.

[RT74] “ Hệ thống chia sẻ thời gian UNIX ” của M. Ritchie, K. Thompson. CACM Tập 17: 7, 1974. Bài báo gốc về UNIX. Đọc nó để xem nền tảng của hầu hết các hệ điều hành hiện đại.

[S00] “UBC: Hệ thống con vào / ra hợp nhất và bộ nhớ đệm hiệu quả cho NetBSD” của Chuck Silvers. FREENIX, 2000. Một bài báo hay về việc tích hợp bộ đệm hệ thống tệp và bộ đệm trang bộ nhớ ảo của NetBSD. Nhiều hệ thống khác cũng làm điều tương tự.

[S + 96] “Khả năng mở rộng trong Hệ thống tệp XFS” của Adan Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, Geoff Peck. USENIX '96, tháng 1 năm 1996, San Diego, California.
Nỗ lực đầu tiên để làm cho khả năng mở rộng của các hoạt động, bao gồm những thứ như có hàng triệu tệp trong một thư mục, là trọng tâm trung tâm. Một ví dụ tuyệt vời về việc đẩy một ý tưởng đến cực điểm. Ý tưởng chính đằng sau hệ thống tệp này: mọi thứ đều là một cái cây. Chúng ta cũng nên có một chương về hệ thống tệp này.

Bài tập về nhà (Mô phỏng)

Sử dụng công cụ này, vsfs.py, để nghiên cứu cách trạng thái hệ thống tệp thay đổi khi các hoạt động var ious diễn ra. Hệ thống tệp bắt đầu ở trạng thái trống, chỉ với một thư mục gốc. Khi quá trình mô phỏng diễn ra, các hoạt động khác nhau được thực hiện, do đó sẽ từ từ thay đổi trạng thái trên đĩa của hệ thống tệp. Xem README để biết chi tiết.

Câu hỏi 1.

Chạy trình mô phỏng với một số hạt ngẫu nhiên khác nhau (ví dụ 17, 18, 19, 20) và xem liệu bạn có thể tìm ra hoạt động nào phải diễn ra giữa mỗi lần thay đổi trạng thái hay không.

2. Bây giờ làm tương tự, sử dụng các hạt ngẫu nhiên khác nhau (ví dụ 21, 22, 23, 24), ngoại trừ chạy với cờ -r, do đó khiến bạn đoán trạng thái thay đổi khi được hiển thị hoạt động. Bạn có thể kết luận gì về các thuật toán phân bổ inode và khối dữ liệu, về việc chúng thích phân bổ khối nào hơn?

3. Bây giờ hãy giảm số lượng khối dữ liệu trong hệ thống tệp xuống số rất thấp (giả sử là hai), và chạy trình mô phỏng cho một trăm hoặc hơn yêu cầu. Những loại tệp nào kết thúc trong hệ thống tệp trong bố cục bị ràng buộc cao này? Những loại hoạt động nào sẽ không thành công?

4. Bây giờ làm tương tự, nhưng với inodes. Với rất ít inodes, những loại thao tác nào có thể thành công? Cái nào thường sẽ thất bại? Trạng thái cuối cùng của hệ thống tệp có thể là gì?