Chapter 14

檔案系統的製作

章節目標

- 描述製作區域檔案系統和目錄架構的細節
- 討論區塊分配和閒置區塊演算法及權衡
- 探索檔案系統效率和效能問題
- 看看檔案是如何從系統錯誤中恢復
- 將 WAFL 檔案系統描述為一個具體範例

14.1 檔案系統結構

- 為了增進 I/O 效率，在主記憶體和磁碟之間的 I/O 傳遞是以區塊 (block) 為單位
  - 每個區塊是一個或多個磁區
- 根據磁碟機的不同，磁區可以由 32 位元組到 4,096 位元組
  - 通常都是 512 位元組

14.1 檔案系統結構

- **檔案系統** (file system) 藉由允許資料能方便地儲存、找到及重新取出，以提供有效且方便的存取磁碟方法
- 檔案系統在設計上有兩個非常不同的問題
  - 第一個問題是如何定義檔案系統讓使用者看起來像什麼
    - 包括對檔案的定義和它的性質、在一個檔案上允許的作業，以及目錄組織檔案的結構
  - 需要建立一些演算法和資料結構來將邏輯檔案系統映射到實體輔助儲存器上

14.1 檔案系統結構

- 檔案系統本身一般是由許多不同層次所組成，如圖 14.1 中所示的結構是層次化設計的一個例子
  - 每一層的設計都使用比它低層的特性來建立一個比它高層使用的特性

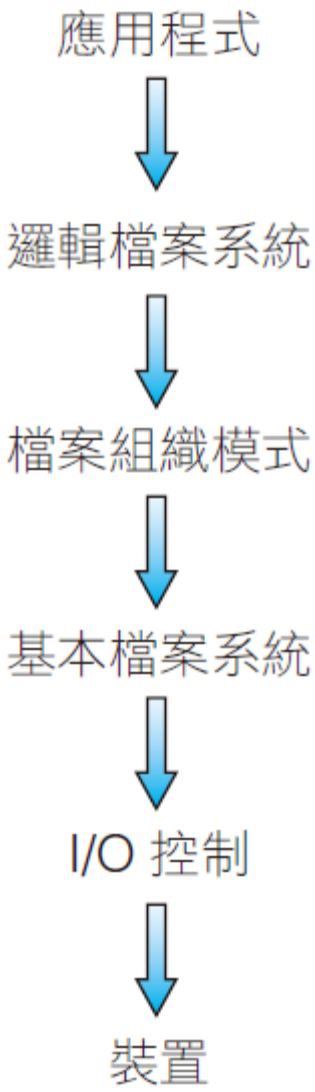


圖 14.1 層次化的檔案系統

14.1 檔案系統結構

- **I/O 控制** (I/O control) 層包括裝置驅動程式 (device driver) 和實際在主記憶體與磁碟系統之間傳遞資訊時的中斷控制程式
  - 裝置驅動程式可以當成是一個轉換程式，它的輸入是由類似“取出區塊 123”的高階命令組成
  - 它的輸出是由低階，並和硬體有關的指令組成，這些指令將交由硬體控制器使用，硬體控制器是 I/O 裝置和系統其餘部份的介面
  - 裝置驅動程式通常會把特定的位元樣型寫到 I/O 控制器之記憶體的特定位置，以告訴控制器在裝置的哪個位置動作，以及採取什麼動作

14.1 檔案系統結構

- **基本檔案系統** (basic file system)
  - 在 Linux 中稱為“區塊 I/O 子系統”(block I/O subsystem)
  - 只需要發出一般性命令給適當的裝置驅動程式，就可以讀或寫儲存裝置上的區塊，每個磁碟區塊都用它的磁碟位址數字來識別
  - 這個層次也管理持有不同檔案系統、目錄和資料區塊的記憶體緩衝區與快取
  - 在一個磁碟區段的轉移可以發生前，會先配置緩衝區的一個區塊

14.1 檔案系統結構

- 當緩衝區滿的時候，緩衝區管理器必須找到更多緩衝記憶體或釋放緩衝空間，以允許一個要求的 I/O 完成
- 快取被用來存放經常使用的檔案系統元資料以增進效能，所以管理它們的內容對於最佳的系統效能是很重要的

- **檔案組織模組** (file-organization module) 知道檔案和它們的邏輯區塊
  - 每個檔案的邏輯區塊皆由 0 (或 1) 開始數到 N
  - 檔案組織模組還包括閒置空間管理程式，此程式追蹤未配置區塊，並在需要時提供這些區塊給檔案組織模組

14.1 檔案系統結構

- **邏輯檔案系統** (logical file system) 管理元資料的資訊 (metadata information)
  - 元資料包括所有檔案系統結構，但不包含真正的資料 (亦即檔案的內容)
  - 邏輯檔案系統管理目錄結構，以提供檔案組織模組對符號式檔案名稱需要的一些資訊
  - 它經由檔案控制區段維護檔案
- **檔案控制區塊** (file-control block, FCB)
  - 在 UNIX 檔案系統是 inode) 包含關於檔案的資訊，
    - 包括所有權、允許權和檔案內容所在的位置

14.1 檔案系統結構

- 目前許多使用的檔案系統，大多數作業系統支援一種以上的檔案系統
  - 例如，大部份CD-ROM 是以 ISO 9660 格式寫入，這是一種 CD-ROM 製造商同意的標準格式
  - 除了可移動媒體的檔案系統之外，每個作業系統都有一種或多種以磁碟為基礎的檔案系統
  - UNIX 使用 **UNIX 檔案系統** (UNIX file system, UFS)，UFS 以 Berkeley 快速檔案系統 (Berkeley Fast File System, FFS) 作為基礎
  - Windows 支援 FAT、FAT32 和 NTFS (即 Windows NT File System) 等磁碟檔案系統格式，另外也支援 CD-ROM 和DVD 檔案系統格式

14.1 檔案系統結構

- Linux 支援超過 130 種不同檔案系統
  - 標準的 Linux 檔案系統稱為延伸檔案系統 (extended file system)
  - 最常見的版本是 ext3 和 ext4
  - 還有由一個或多個客戶電腦跨越網路掛載之伺服器檔案系統的分散式檔案系統

14.2 檔案系統製作

- 一個**啟動控制區塊** (boot control block) (每個卷區) 可能包含作業系統需要的資訊，以便將作業系統從該卷區啟動
  - 如果磁碟沒有包含作業系統，這個啟動控制區塊可能是空的
  - 通常它是卷區的第一個區塊
    - 在 UFS 中，此區塊稱為**啟動區塊** (boot block)
    - 在 NTFS 中，它是分割區啟動磁區 (partition boot sector)

14.2 檔案系統製作

- 一個**卷區控制區塊** (volume control block) (每個卷區) 包含卷的詳細資料
  - 例如此卷區的區塊數目、區塊大小、閒置區塊的數目、閒置區塊的指標、閒置 FCB 數目與 FCB指標
  - 在 UFS 中，這稱為**超級區塊** (superblock)
    - 在 NTFS，它被儲存在**主檔案表** (master file table)
- 目錄結構 (每個檔案系統) 被用在組織檔案
  - 在 UFS 中，其中包含檔案名稱和連接的 inode 數字
  - 在 NTFS 中，儲存在主檔案表

14.2 檔案系統製作

- 每個檔案都有的 FCB 包含許多檔案的細節，它有唯一的識別碼以允許關聯到目錄條目
  - 在 NTFS 中，這項資訊實際上儲存在主檔案表中
  - 主檔案表使用關聯式資料庫結構，每個檔案占用一列

14.2 檔案系統製作

- 記憶體中的**掛載表格** (mount table) 包括每個已掛載卷區的資訊
- 記憶體中的目錄結構快取，保有最近存取目錄的資訊
  - 對於被安裝卷區的目錄，可以包含一個指向卷區表格的指標
- **全系統的開啟檔案表格** (system-wide open-file table) 包含每個開啟檔案之 FCB 的複製及其它資訊
- 每個行程的**開啟檔案表格** (per-process open-file table) 包含指標
  - 指到全系統開啟檔案表格的適當條目及其它資訊
- 當它們被讀取硬碟或寫入硬碟時，持有檔案系統區塊的緩衝區

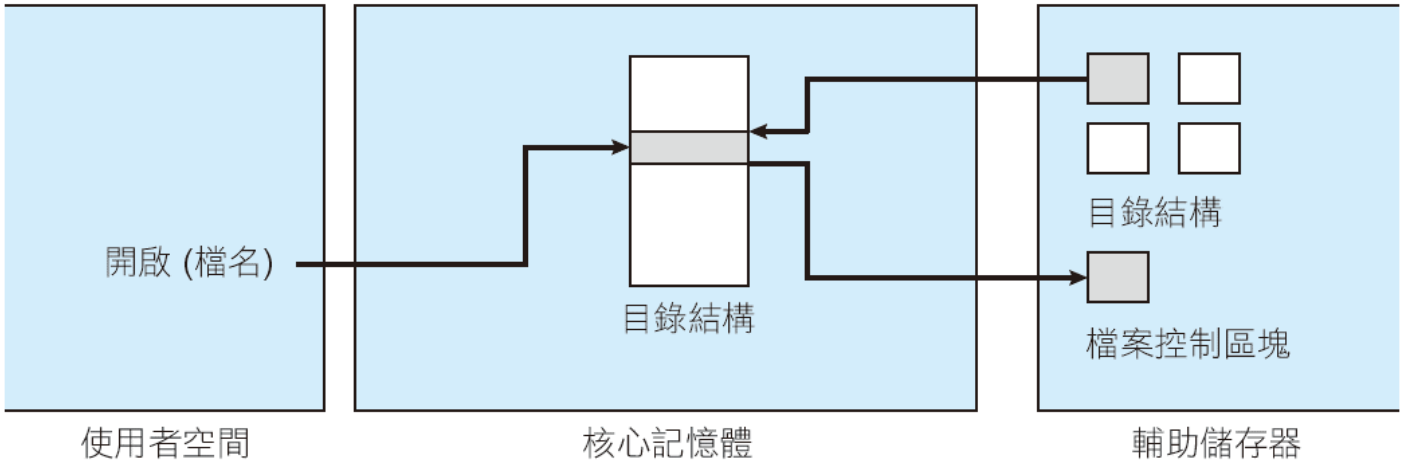
圖 14.2　典型的檔案控制區塊 (FCB)

檔案許可權
檔案日期 (建立、存取、寫入)
檔案擁有者、群組、ACL
檔案大小
檔案資料區塊或檔案資料區塊指標

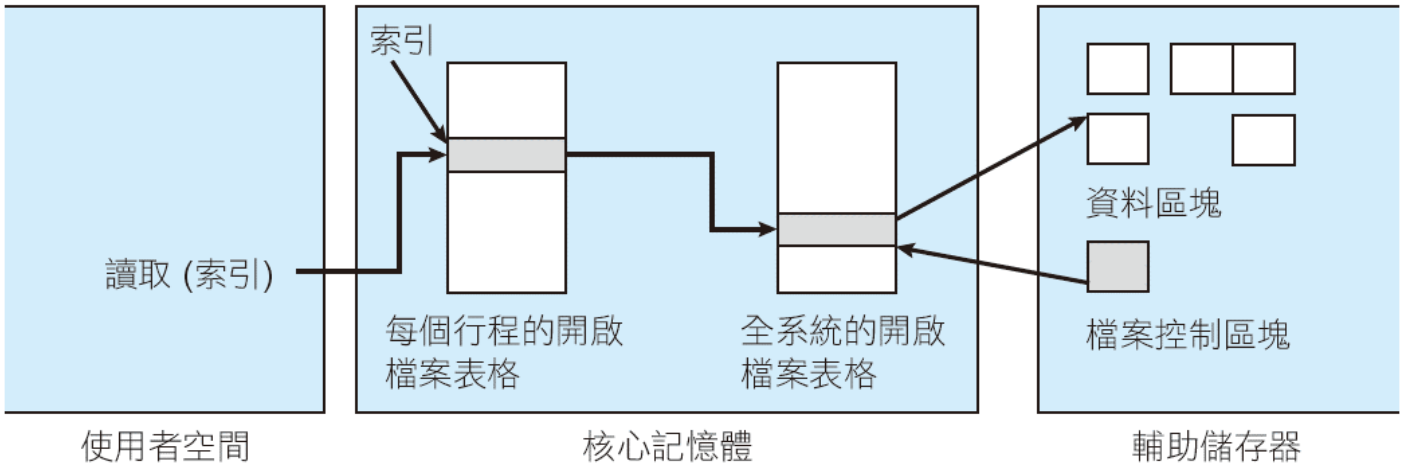
14.2 檔案系統製作

- 在每個行程開啟檔案表格中建立一條目，其中包含指向全系統的開啟檔案表格指標和一些其它的欄位
- 檔案可以被快取，以對後來開啟相同檔案節省時間。條目的名稱不同
  - UNIX 系統稱它為**檔案描述符** (file descriptor)
  - Windows 稱為**檔案處置** (file handle)

圖14.3　記憶體中的檔案結構



(a) 檔案開啟



(b) 檔案讀取

14.3 目錄製作

- 選擇目錄配置和目錄管理的演算法
  - 線性串列
  - 雜湊表格

14.3.1 線性串列

- 製作一個目錄最簡單的方法是，使用一個線性串列的檔案名稱，並以指標指向資料區塊
  - 這種方法很容易寫程式，但執行上很浪費時間
  - 產生一個新檔案時，我們必須先尋找目錄以確定沒有同名的檔案存在
  - 在目錄尾端加入新的條目。刪除一個檔案時，我們搜尋目錄中指名的檔案，然後釋放配置給它的空間
  - 重新使用此目錄條目時，我們可有以下一些作法。我們可以標示此條目為未使用
  - 設定為一個特殊名字

14.3.1 線性串列

- 例如一個皆為空白的名字，為它分配一個無效的 inode 編號 (例如 0)
- 或是在每個條目中包含一個使用過—未使用過位元的設定

- 或是我們可以把它加入一串閒置目錄條目所組成的串列
- 第三個選擇是複製目錄最後的條目到閒置的位置，然後減少目錄的長度
  - 鏈結串列也可以用來減少刪除一個檔案的時間

14.3.1 線性串列

- 線性串列組成的目錄條目之真正缺點在於線性尋找檔案
  - 目錄資料經常被用到，較慢的存取作法會被使用者注意到
  - 許多作業系統製作軟體的快取方式來儲存經常用到的目錄資訊
  - 快取的命中可以避免經常地重新從磁碟讀取資料
  - 一個排序過的串列允許使用二位元搜尋，也可以降低平均的搜尋時間
  - 對於確保串列維持成排序狀態的要求，也可能使建立和刪除檔案變得複雜，因為我們可能必須移動大量的目錄資訊才能維持排序的目錄

14.3.1 線性串列

- 一個更複雜的樹狀資料結構
  - 例如平衡樹，在這裡可能會有幫助
- 排序串列的優點是不需要分別的排序步驟，就可以產生一個已經排序的目錄串列

14.3.2 雜湊表格

- 另外一種曾用在檔案目錄的資料結構是雜湊表格 (hash table)
  - 在這種方法中，一個線性串列儲存目錄的條目，但同時也使用一個雜湊方法的資料結構
  - 雜湊表格從檔案名稱計算出一個數值，並且傳回一個指標給線性串列中的檔案名稱，因此它可以大量降低目錄搜尋時間
  - 插入和刪除也很直接，雖然對於碰撞——兩個檔案名稱經雜湊法指向同一位置時必須有些對策

14.3.2 雜湊表格

- 雜湊表格的最主要困難是雜湊表格的大小通常固定，以及雜湊函數決定雜湊表格的大小
  - 例如，假設我們製作一個擁有 64 個條目的線性插入雜湊表格
  - 雜湊函數轉換檔案名稱為 0 到 63 的整數 (例如由使用除以 64 的餘數)
  - 如果稍後我們試圖建立第 65 個檔案，就必須擴大此目錄的雜湊表格
    - 例如增加到 128 個項目
  - 需要一個新的雜湊函數，此函數必須映射檔案名稱為 0 到 127，而且必須識別出現存的目錄條目，以反映出它們新的雜湊函數值

14.3.2 雜湊表格

- 可以使用鏈結溢位雜湊表格
  - 每個雜湊表格的進入點可以是一個鏈結串列，而不必只是一個個別值
    - 因此可以藉著加入新的條目到鏈結串列，以解決碰撞的問題
  - 此方法在查詢時可能會較慢，因為搜尋檔案名稱可能需要一步步地經歷碰撞表格項目的鏈結串列
    - 但是這個方法好像是比經由整個目錄的線性搜尋快上很多

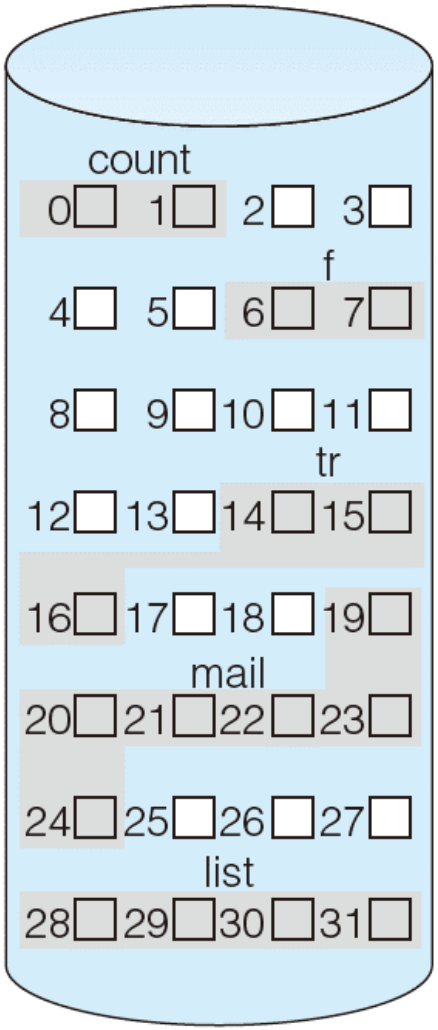
14.4 配置方法

- 三種主要的分配輔助儲存器空間方法正被廣泛使用中：
  - 連續分配
  - 鏈結分配
  - 索引分配

14.4.1 連續分配

- **連續分配** (contiguous allocation) 需要令每個檔案占用一組裝置上的連續區塊
  - 裝置位址定義一個線性裝置上排列次序
  - 使用此順序時 (假設只有一個工作對裝置做存取)，在存取 b 區塊之後存取 $b+1$ 區塊時一般不需要移動磁頭
  - 若是要移動磁頭 (由一磁柱的最後一個磁區移至另一磁柱的第一磁區)，也只是移動一條磁軌到下一條而已
  - 對 HDD 而言磁碟在存取連續分配檔案的尋找次數將可達到最小，且搜尋時間亦為最短 (假設區塊及封閉邏輯位址實際上是封閉的)，只有在最終需要時搜尋

圖 14.4　磁碟空間連續分配



目錄

檔案	開始	長度
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

14.4.1 連續分配

- 有一種防止外部斷裂現象造成大量磁碟空間被損失的策略是
  - 把整個檔案系統複製至另一裝置上。然後把原裝置全部空出來，建立一個大的連續閒置區域
  - 再將複製的那些檔案複製回原裝置，並且連續地把這一個大區間分配給各檔案
  - 這個方法有效地**聚集** (compact) 所有的閒置空間為一個連續空間，解決斷裂的問題
  - 聚集法所費的代價就是時間，然而對於大型硬碟而言，時間成本特別地明顯
  - 聚集這些裝置可能要花好幾小時，也可能要幾週

14.4.1 連續分配

- 有些系統要求這個功能在**離線** (off-line) 來做，這個時候檔案系統未掛載
- 在這段**停止時間** (down time)，通常不能允許正常的系統操作，就量產的機器成本而言，這些聚集無論如何也要避免
- 大部份需要解決分散問題的現代系統在正常系統運作期間需要**線上** (on-line) 執行重組，但是效能負面的影響較明顯

14.4.1 連續分配

- 為了減少這些缺點，有些作業系統使用修正的連續分配法，這種方法一開始分配一塊連續的空間
  - 然後當此空間不夠大時，另一塊連續空間 [叫作延伸 (extent)] 就加到原先的配置
  - 此檔案的位置則記錄成某一位址和一個區塊計數，再加上一個鏈結到下一段延伸開頭的指標
  - 在某些系統上，檔案擁有者可以設定延伸的大小
    - 但如果使用者設定不正確，則將造成沒有效率
    - 如果延伸部份太大，內部斷裂依然是一個問題
    - 大小可變的延伸部份配置或取消配置時依然會有外部斷裂的問題

14.4.1 連續分配

- 商業的 Symantec Veritas 檔案系統使用延伸使其效能最佳化，它對標準 UNIX UFS 而言，是高效能替換

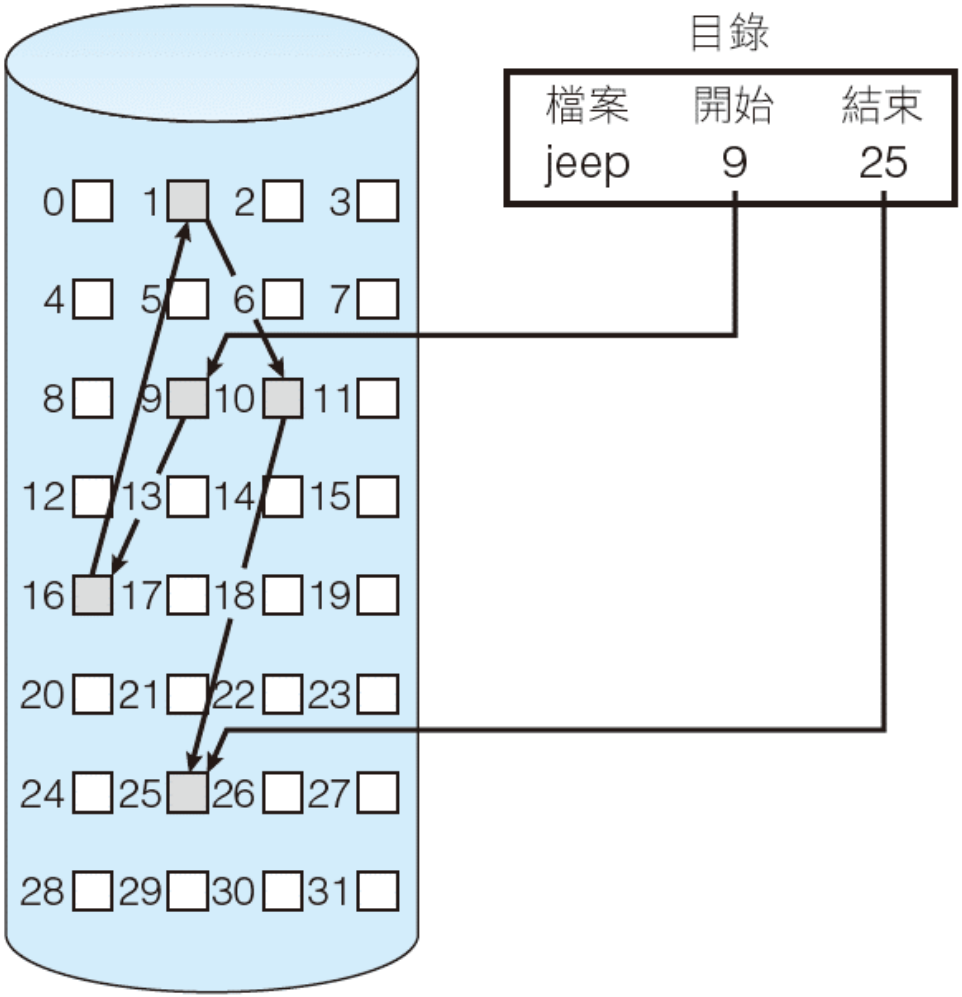
14.4.2 鏈結分配

- **鏈結分配** (linked allocation) 可以解決連續分配所造成的問題
  - 使用鏈結分配，每個檔案都是儲存器區段的鏈結串列 (linked list)；儲存器上的區塊可能散佈到裝置上的任何地方
  - 目錄中存放著指向檔案第一個和最後一個區塊的指標
    - 舉例來說，由區塊 9 開始的一個擁有五個區塊的檔案，可能由區塊 9 接到區塊 16，然後接到區塊 1、區塊 10，最後到區塊 25 (圖 14.5)

14.4.2 鏈結分配

- 每個區塊都含有一個指向下一個區塊的指標，這個指標是使用者無法取得
  - 如果每個區塊是 512 位元組，而每個區塊位址 (指標) 需要 4 位元組，那麼使用者可用的有 508 位元組

圖 14.5　磁碟空間的鏈結分配



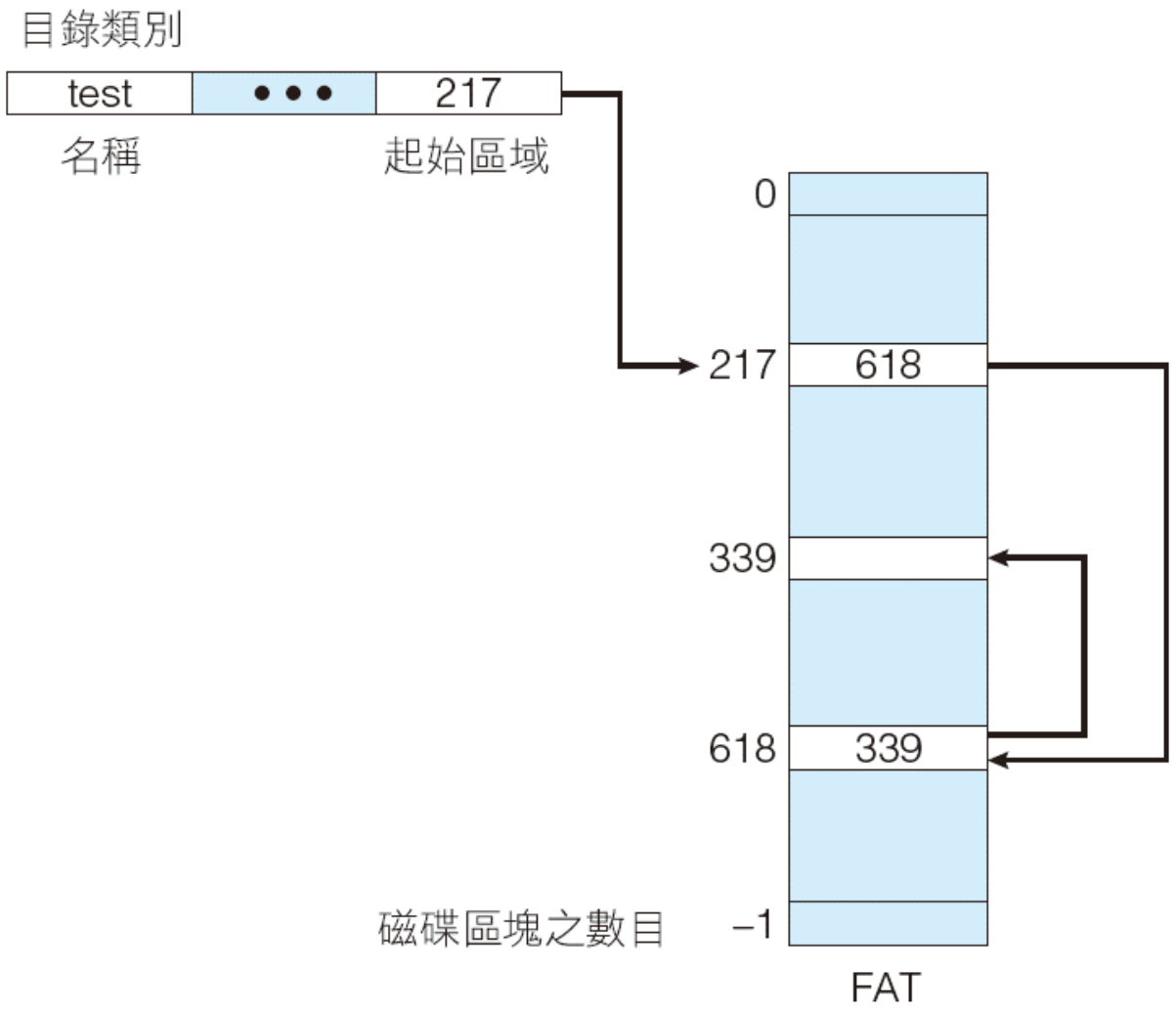
14.4.2 鏈結分配

- 鏈結分配的一項重要變化方式是採用**檔案配置表格** (file-allocation table, FAT)
  - 這種簡單但有效率的磁碟空間配置法用在 MS-DOS 作業系統
  - 每一卷區的開頭儲存器部份都用來存放此表格
  - 該表格為每個區塊保有一份記錄，並且以區塊號碼當作索引
  - FAT 用起來很像鏈結串列
  - 目錄中包含檔案第一個區塊的區塊號碼

14.4.2 鏈結分配

- 以區塊號碼為索引之表格記錄，包含檔案中下一個區塊的區塊號碼
- 如此不斷連接，直到最後一個區塊，其表格記錄是一特殊之檔案結束 (end-of-file) 數值
- 未使用之區塊以 0 的表格數值來表示
- 分配一個新的區塊給檔案只不過是找尋第一個 0 值的表格記錄，然後以新區塊位址來取代先前之檔案結束數值，而 0 值便以檔案結束數值來替代
- 圖 14.6 即為一包含磁碟區塊 217、618 及 339的檔案之 FAT 結構

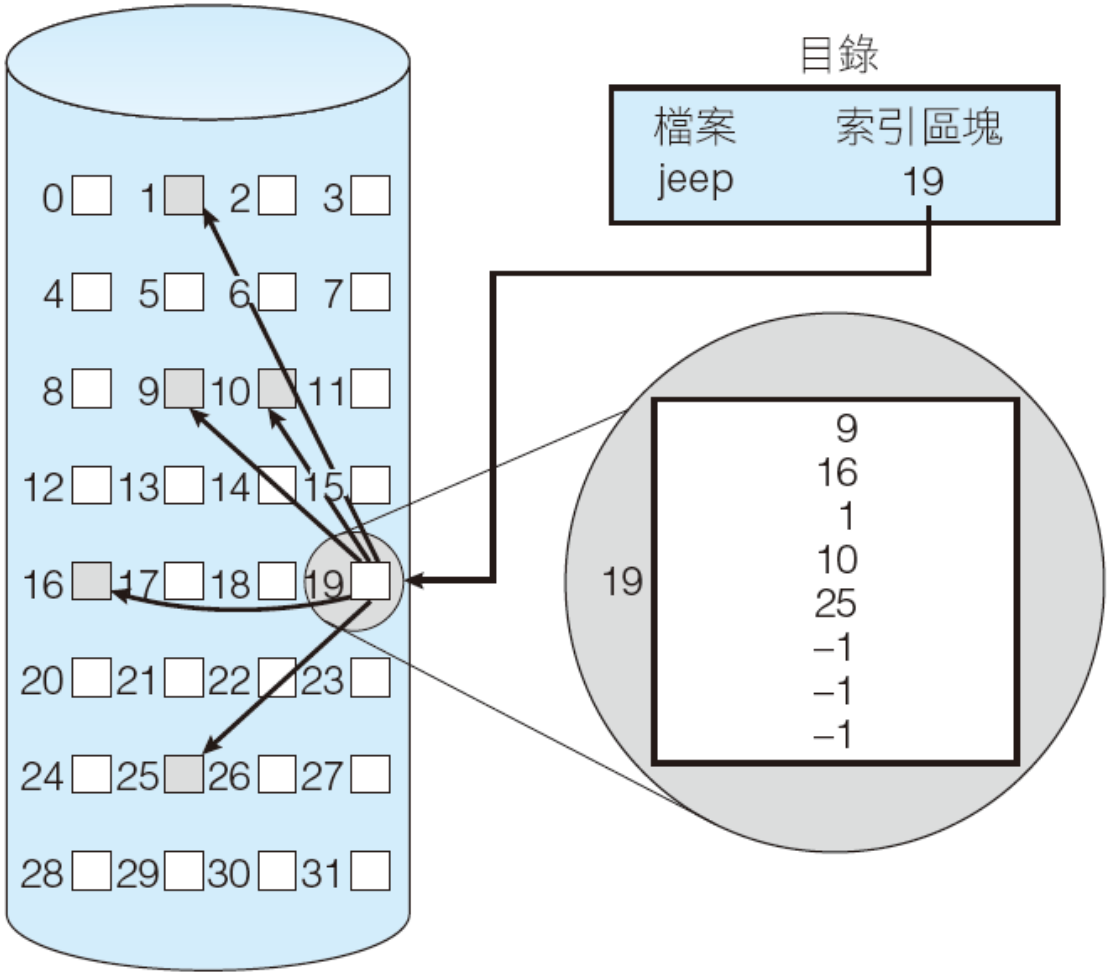
圖 14.6　檔案配置表格



14.4.3 索引分配

- 用鏈結分配可解決外部斷裂和連續分配必須做檔案大小宣告的問題
  - 如果缺乏一個 FAT，鏈結分配就無法有效的支援直接存取，因為檔案的各個區塊散佈在磁碟中
  - 更重要的是，指向各區塊的指標也是散佈在整個磁碟中，而且需要依序取出
  - **索引分配** (indexed allocation) 可以解決上述的問題，因為它把所有的指標都集中放在一個地方：
  - **索引區塊** (index block)

圖 14.7　磁碟空間的索引分配



14.4.3 索引分配

- 鏈結方式：
  - 一個索引區塊一般是一個儲存器區塊，因此它可以直接由自己來讀取或寫入
  - 為了允許大型檔案使用，我們可以把幾個索引區塊鏈結在一起
    - 例如，一個索引區段可以保存一個提供檔案名稱的小標頭，並且將前 100 組磁碟區塊的位址存在裡面
  - 下一個位址 (索引區塊的最後一個字) 是 null (對小檔案來說)，或是指向另一個索引檔案的指標 (對大檔案來說)

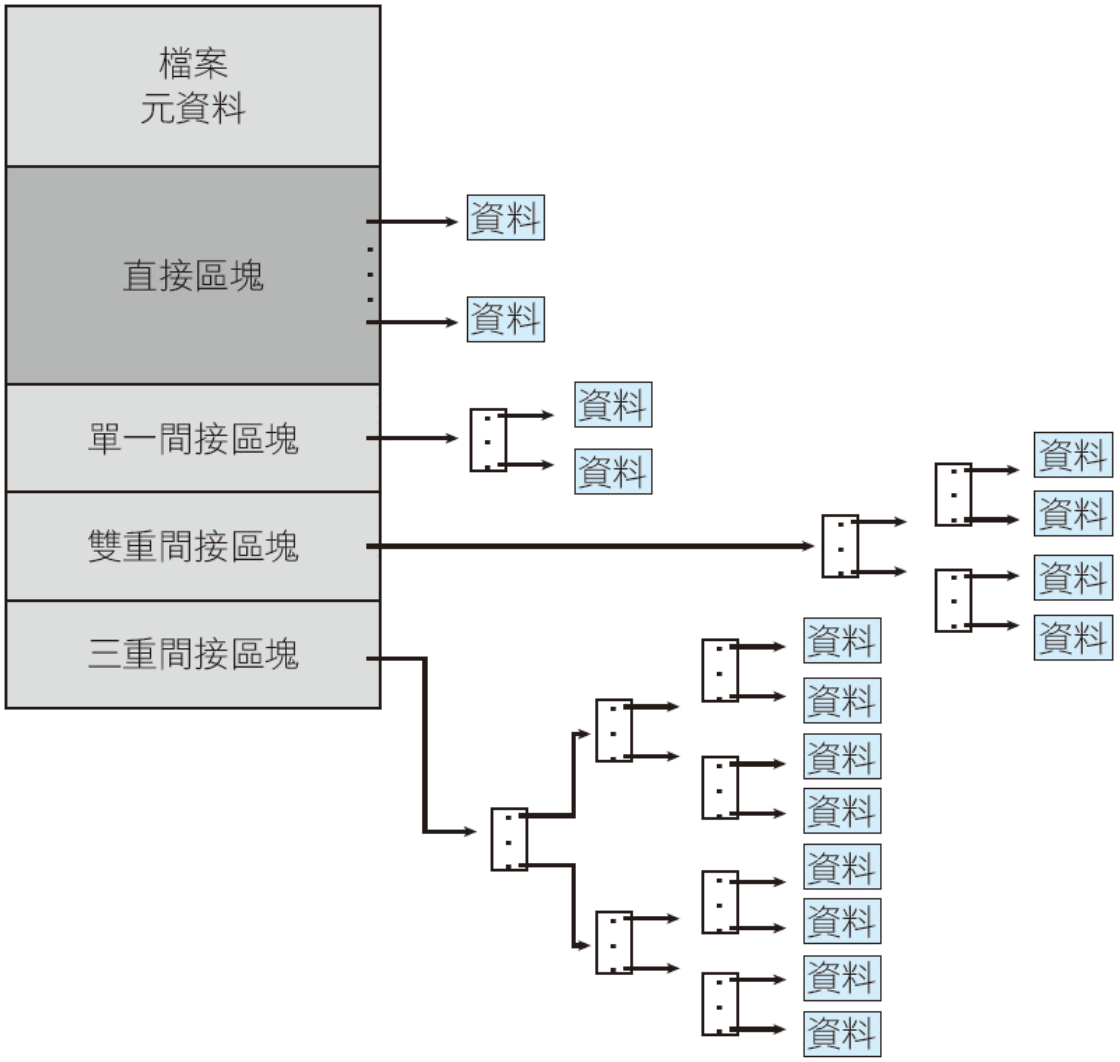
14.4.3 索引分配

- 多層索引：
  - 一種不同的鏈結表示法是使用第一層索引區塊指向第二層索引區塊組，然後再指向檔案區塊
  - 為了存取一個區段，作業系統使用第一層索引找出第二層索引區塊，再用這個區塊找出想要的資料區塊
  - 這個方法可以延續到第三層或第四層，這是依照想要的最大檔案之大小決定
  - 使用 4,096 位元組的區塊，我們可以儲存 1,024 個 4 位元組指標在索引區塊中
  - 兩層的索引允許 1,048,576 個資料區塊，且允許一個檔案最大到 4 GB

14.4.3 索引分配

- 組合方式：
  - 另一種使用於 UNIX 為基礎之檔案系統中的方式，是保存前 15 個索引區塊指標在檔案的 inode 中
  - 這些指標的前 12 個指向**直接區塊** (direct block)
    - 直接包含保有檔案資料的區塊位址
  - 三個指標則是指向**間接區塊** (indirect block)：
    - **單一間接區塊** (single indirect block)，索引區塊的內容不是資料，而是包含著資料區塊的位址
    - **雙重間接區塊** (double indirect block)，包含一個區塊的位址，甚至更多區塊位址
    - **三重間接區塊** (triple indirectblock) 的位址

圖 14.8 UNIX 的 inode



14.4.4 效 能

- 在各種不同分配方法之間的儲存效能差異很大，同時對資料區塊存取所花費的時間也各有不同
  - 兩者在作業系統製作時，該選擇的正確方法是非常重要的準則
- 選擇一種分配方法之前，必須決定這些系統到底是如何使用的
  - 對於主要是循序存取系統使用的方法應該和主要是隨機存取系統不相同

14.4.4 效 能

- 無論是用哪一種存取方式，連續分配只需要用一次存取來得到一個區塊
  - 因為我們可以很方便地將檔案的起始位址保存在記憶體中，所以可以立刻算出第 *i* 個區塊的位址 (或下一個區塊的位址)，並且將它直接讀取
- 對於鏈結分配來說，我們也可以將下一個區塊的位址保存在記憶體中並直接讀取
  - 這種方法適用於循序存取，但是對於直接存取，對第 *i* 個區塊的存取可能需要做 *i* 次讀取的工作
  - 這個問題就說明為什麼鏈結分配不該用在需要直接存取的事例上

14.5 閒置空間管理

- 因為只有有限的儲存體空間，因此新建的檔案就必須使用被刪除檔案所占用的空間
  - 一次寫入光碟只允許寫入檔案到任何特定磁區，因此再使用實際上是不可能的
  - 為了記錄閒置磁碟空間的磁軌，系統採用一種**閒置空間串列** (free-space list)
- 閒置空間串列記錄所有閒置裝置的區塊
  - 也就是尚未分配給任何檔案的記憶空間

14.5 閒置空間管理

- 若要建一個檔案，我們就搜尋閒置空間串列來找尋所需的空間量
  - 並且將它分配給新的檔案，於是這些空間就從閒置空間串列中移除
  - 當一個檔案被刪除後，它的磁碟空間就加到閒置空間串列上
  - 閒置空間串列 (儘管其名稱是串列) 是不可以用串列來製作的

14.5.1 位元向量

- 閒置空間串列是以**位元映像** (bitmap) 或**位元向量** (bit vector) 來執行工作每個區塊用 1 位元來表示
  - 如果某區塊是閒置的，該位元為 1
  - 若某區塊已被配置，該位元為 0

14.5.1 位元向量

- 硬體特性驅動軟體的功能
  - 整個向量保留在主記憶體作為大部份存取，否則位元向量是無效用的
    - 偶然地被寫入磁碟做為回復所需
  - 對較小的裝置保持位元向量在主記憶體是可行的，但不能在大裝置上這麼做
    - 一個以 512 位元組為區塊的 1.3 GB 磁碟需要 332 KB 的位元映像以追蹤閒置區塊
    - 若將 4 個群聚成一個叢集，則每一磁碟可以降低為約 83 KB

14.5.1 位元向量

- 一個以 4 KB 為區塊的 1 TB 磁碟需要 32 MB 儲存位元映像
  - $2^{40} / 2^{12} = 2^{28}$ 位元 = 2^{25} 位元組 = 25 MB
- 磁碟大小不斷增加時，位元映像的問題也會繼續升級

14.5.2 鏈結串列

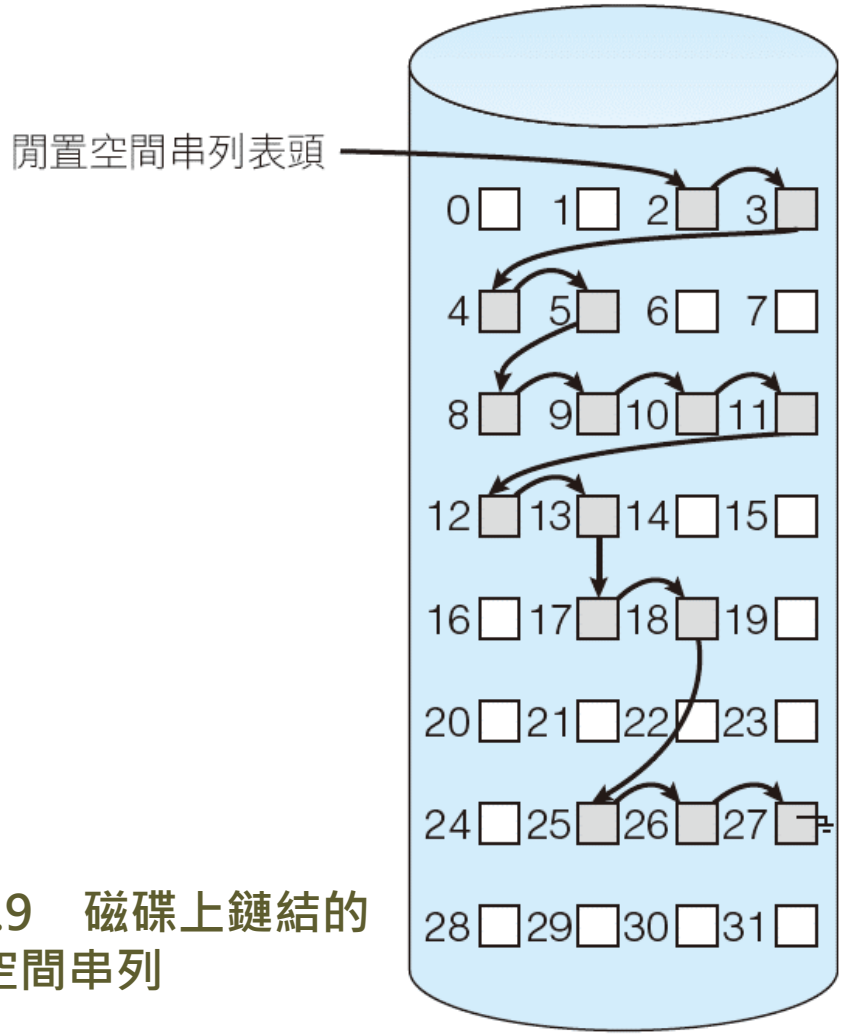


圖 14.9 磁碟上鏈結的閒置空間串列

14.5.3 群 組

- 修正閒置串列方法是將 n 個閒置區塊的位址存放在第一個區塊中，前 $n-1$ 個是實際閒置區塊，最後一個是存放另一個含有另外 n 個閒置區塊在磁碟上的位址
- 這樣執行的重要性是使得大量閒置區塊的位址閒置地找到，和標準鏈結串列方法被使用時的情況不一樣

14.5.4 計 數

- 另一種方法是利用一般情況下許多連續區塊同時被分配或同時被釋放的事實，尤其是在使用連續分配演算法或透過叢集的時候
  - 因此，並不需要保有含 n 個閒置區塊位址的串列，我們只需要記住第一個閒置區塊的位址及其後面所連接的 n 個連續閒置區塊的數量即可
  - 於是在閒置空間串列中的每個條目中只包含一個裝置位址和一個計數值
  - 雖然每個條目比起單一磁碟位址占用較多的空間，但是整個串列變短了一些，只要該數值平均上比 1 大就可以

14.5.4 計 數

- 注意，這個追蹤閒置空間的方法和分配區塊的延伸方法類似
- 這些條目可以存入一個平衡樹，而不必使用鏈結串列，以做有效率的查詢、插入和刪除

14.5.5 空間地圖

- Oracle 的 ZFS 檔案系統 (在 Solaris 和其它作業系統可找到) 被設計成可以擁有大量的文件、目錄
  - 甚至是檔案系統 (在 ZFS 中，我們可以建立檔案系統層次結構)
  - 在這些規模下，元資料的 I/O 對效率有很大的影響
    - 舉例來說，如果閒置空間陣列是由位元地圖來實現，則位元地圖必須在何時分配區塊及何時釋放區塊這兩點來做修改
    - 從 1 TB 的磁碟中釋放 1 GB 的資料時，會導致數千個位元地圖的區塊被更新，因為這些資料散佈在整個磁碟中
    - 這種系統的資料結構可能很大且沒有效率

14.5.5 空間地圖

- 在閒置空間的管理上，ZFS 使用結合的技術來控制資料結構的大小和降低管理這些結構所需的 I/O
  - 首先，ZFS 建立 **metaslab** 來分割裝置上的空間，成為可管理大小的塊。
  - 一個卷區可能包含數百個 metaslab，每個 metaslab 有一個相關的空間地圖
  - ZFS 使用計數演算法來儲存關於閒置區塊的資訊
  - 其計數演算法使用日誌結構的檔案系統結構技術來記錄，而不將計算結構使用寫入硬碟
  - 空間地圖是一個按照時間順序和計數格式記錄所有區塊活動的日誌 (分配和釋放)

14.5.5 空間地圖

- 當 ZFS 決定要從metaslab 分配或釋放空間時，便把關聯的空間地圖讀入平衡樹架構的記憶體中，以偏移量為索引，並重播日誌到架構上
- 記憶體上的空間地圖是一個有關 metaslab 中分配和釋放空間的準確表示法
- ZFS 也藉由結合連續閒置區塊到單一項次，以盡可能地凝聚地圖
- 最後，在磁碟中的閒置空間陣列被更新，以做 ZFS 中以交易為導向運作的一部分
- 在收集和整理階段裡，區塊的請求仍有可能發生，而 ZFS 從日誌中來滿足這些要求
- 本質上來說，日誌加上平衡樹就是使用串列

14.6 效率和效能

- 效　率
  - 儲存器裝置空間的有效使用，主要是取決於分配和目錄演算法的使用方法
  - 例如，UNIX 的 inode 是預先分配在一個卷區
  - 即使一個空的磁碟也有一定比例的空間因 inode 而損失
  - 但是藉由預先配置 inode，並且散佈到卷區，我們增進檔案系統的效能
  - 這項增進的效能是 UNIX 分配和閒置空間演算法的結果，這些方法試圖使一個檔案資料區塊接近檔案 inode 區塊，以降低搜尋時間

14.6 效率和效能

- 效　能
  - 有些系統維護一塊獨立的主記憶體作為**緩衝區快取** (buffer cache)，保存在緩衝區快取的區塊都是假設它們馬上會被利用到
  - 其它系統使用**分頁快取** (page cache) 將檔案資料儲存快取
  - 分頁快取使用虛擬記憶體的技巧將檔案資料以分頁的方式快取，而非以檔案系統的區塊做快取
  - 使用虛擬位址快取檔案比從實體磁碟區塊快取時快很多，因為是以虛擬記憶體為存取介面而非檔案系統

14.6 效率和效能

- 有一些系統，包括 Solaris、Linux 和 Windows
  - 使用分頁快取來快取行程分頁和檔案資料
  - 稱為**一致性的虛擬記憶** (unified virtual memory)
- 有些版本的 UNIX 和 Linux 提供**一致性的緩衝區快取** (unified buffer cache)
- 為了闡述一致性的緩衝區快取之好處，考慮以這兩種作法開啟和存取一個檔案
  - 一種作法是使用記憶體映射
  - 第二種作法是使用標準的系統呼叫 read() 和 write()

14.6 效率和效能

- 沒有一致性的緩衝區快取時，我們面臨的狀況和圖 14.10 相似
  - 在這種情況下，read() 和 write() 系統呼叫會經歷緩衝區快取
  - 記憶體映射的呼叫需要使用兩種快取—分頁快取和緩衝區快取
  - 記憶體映射是從檔案系統的磁碟區塊讀入，然後存入緩衝區快取
  - 因為虛擬記憶體系統不能和緩衝區快取介面，因此緩衝區快取的檔案內容必須被複製到分頁快取
  - 這種情況就稱為**雙重快取** (double caching)，並且需要對檔案系統的資料快取兩次

圖 14.10　沒有一致性的緩衝區快取之 I/O

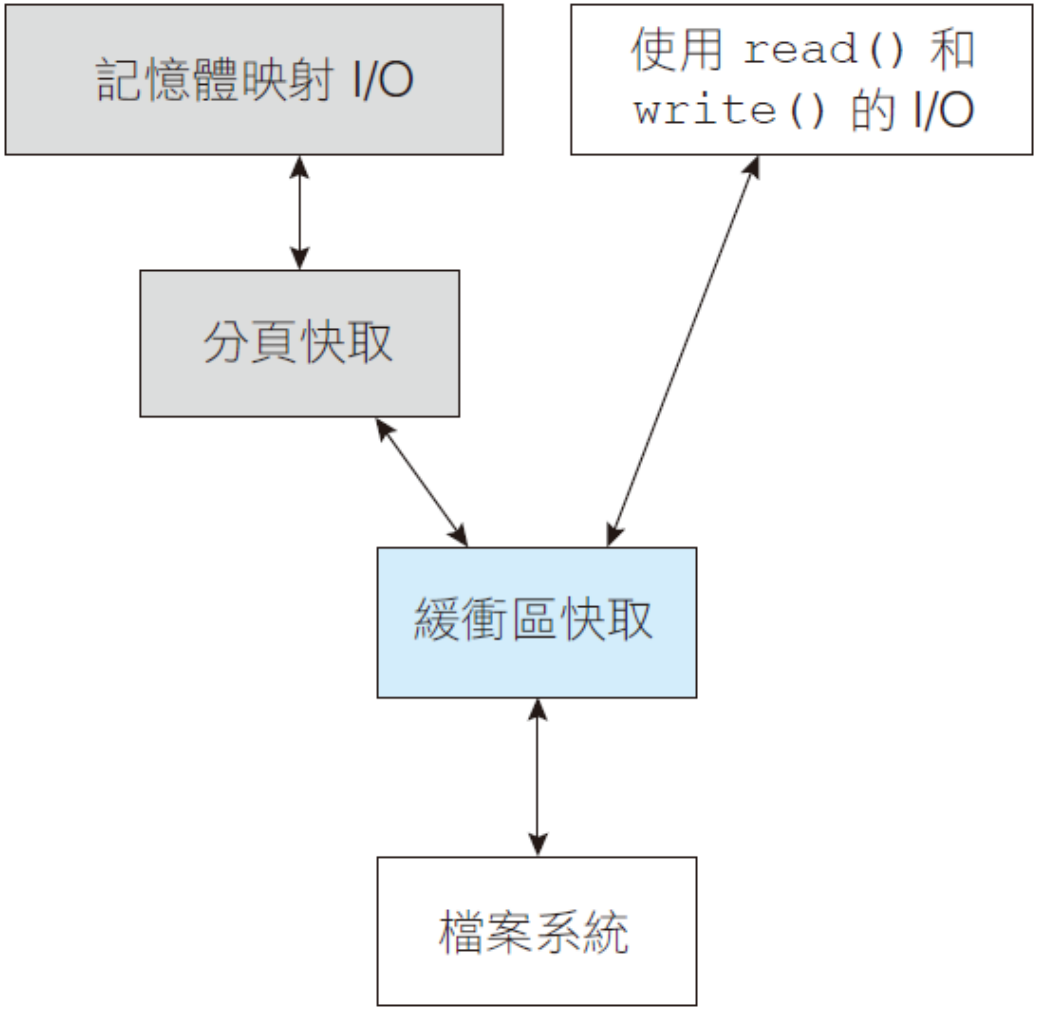
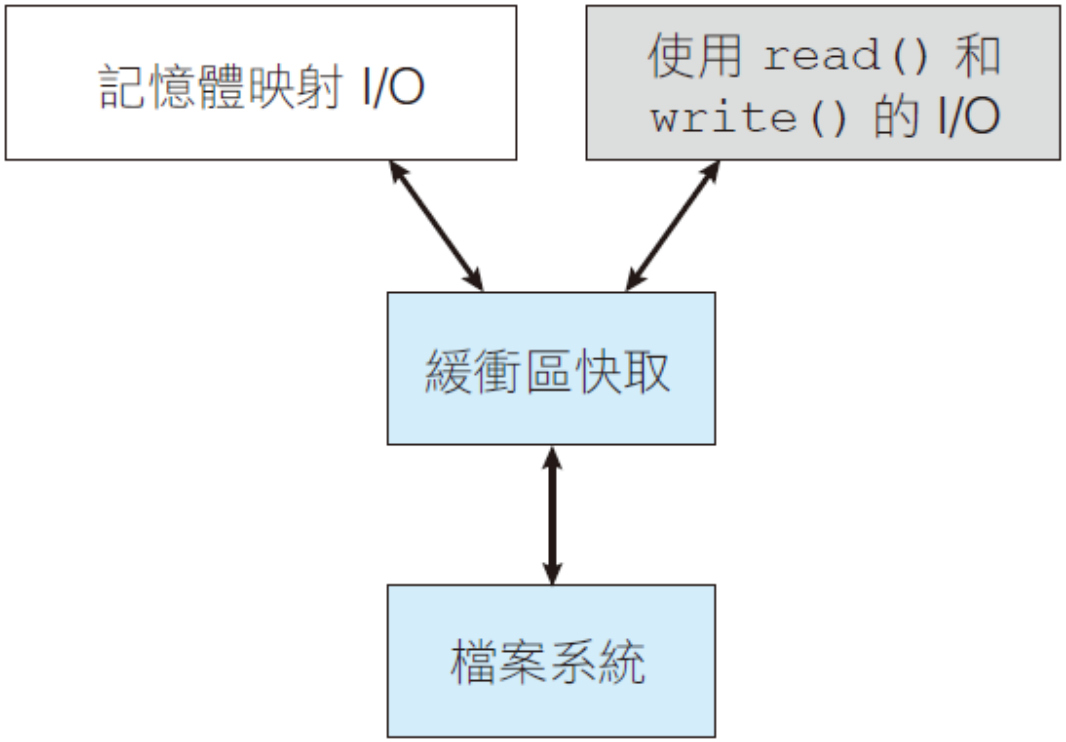


圖 14.11　使用一致性的緩衝區快取之 I/O



14.7 復 原

- **一致性檢查器** (consistency checker) 是一個系統程式
  - 如 UNIX 的 fsck
    - 比較目錄結構的資料和儲存器上的元資料，並且試著修正所發現的任何不一致
- 分配方法和閒置空間管理演算法支配檢查器可能發現什麼類型的問題，以及在修正這些問題時能成功多少
  - 例如，如果使用鏈結分配，從任何區塊都有鏈結接到下一區塊，所以整個檔案可以從資料區塊重建，而目錄結構也可以重建
  - 反之，在一個索引分配系統遺失一項目錄條目可能是很大的災難，因為資料區塊對於其它區塊一無所知

14.7 復 原

- 因為這個原因，UNIX 檔案系統將目錄條目存入快取以便讀取使用，但任何資料寫入快取造成空間配置或其它元資料的改變，都會在對應資料區塊被寫入之前同步地被執行
- 如果同步寫入被快取中斷，則問題仍持續發生
- 部份 NVM 儲存器裝置包含電池或超級電容器，即使在斷電期間也能提供足夠的電力，可將資料從裝置緩衝區寫入儲存媒介，從而不會丟失資料
- 即使是那些預防措施也不能防止崩潰造成的損壞

14.7.2 登錄結構的檔案系統

- 計算機科學家經常發現，原先應用在一個領域的演算法和技術也可以用在其它領域，這種情況對於資料庫以登錄為基礎的復原演算法也是如此
- 這種登錄演算法已經成功地應用在一致性檢查的問題上，所產生的製作被稱為**以登錄為基礎的交易傾向** (log-based transaction-oriented 或 journaling) 檔案系統

14.7.2 登錄結構的檔案系統

- 所有元資料的改變是以循序方式寫入登錄
- 執行特定任務的每一組操作就是一筆交易 (transaction)
  - 一旦改變寫入登錄時，它們就被視為交付，而系統呼叫即可返回使用者行程，並允許它繼續執行
  - 在此同時，這些登錄條目在整個檔案系統中被重新執行
  - 當變更發生時，有一個指標被更新，以指示哪些已經完成，而哪些仍未完成
  - 當一整筆交付的交易完成後，就從登錄檔案中移除，而登錄檔案實際上是一個圓形緩衝區

14.7.2 登錄結構的檔案系統

- 當進行覆寫較舊的數值時，圓形緩衝區 (circular buffer) 寫入空間的尾端，再接著空間的啟始點
- 我們儘量避免緩衝區寫入覆寫尚未儲存的資料
- 登錄資料可能放在檔案系統的一個獨立區塊，或甚至是另一個儲存器裝置上

14.7.2 登錄結構的檔案系統

- 如果系統毀損的話，在登錄檔案中將有零筆或更多筆交易
  - 這些交易雖然已經被作業系統交付，但對檔案系統卻從未完成，所以它們必須被完成
  - 交易可以從指標的地方開始執行，直到工作完成為止，所以檔案系統的結構依然保持一致性
  - 唯一發生的問題是，當一筆交易已經被取消時—換言之，這筆交易在系統毀損之前尚未交付
  - 對檔案系統執行這些交易的任何改變必須回復，才能再次保有檔案系統的一致性
  - 這項復原是系統毀損後必須執行的，可以消除一致性檢查的所有問題

14.7.4 備份或復原

- 儲存器裝置有時候會發生失效，所以必須小心，以確保因為這種失效的資料不會永久遺失
- 為了達到這個目的，系統程式可以用來從磁碟將資料**備份** (back up) 到另外的儲存
  - 例如磁帶或其它輔助儲存器裝置
- 一個個別檔案或一整個磁碟從遺失的情況下復原，就是將資料從備份**復原** (restore)

14.8 範例：WAFL 檔案系統

- NetApp 的 WAFL (wide-anywhere file layout) 檔案系統是這類最佳化的例子
  - WAFL 對隨機寫入而言是一個有力又嚴謹被最佳化的檔案系統
- 由 NetApp 產生的網路檔案伺服器中，WAFL 是唯一被使用的，並且也當作分散式檔案系統使用
  - 針對 NFS 和 CIFS 設計，可以經由 NFS、CIFS、iSCSI、ftp 和 http 協定提供檔案給客戶
  - 當許多客戶使用這些協定與一個檔案伺服器交談時，伺服器可能看到對隨機讀取有非常大量的需求，以及對隨機寫入有更大的需求

14.8 範例：WAFL 檔案系統

- NFS 和 CIFS 協定經由讀取操作來快取資料，因此寫入是檔案伺服器產生者最關心的
- WAFL 被使用在檔案伺服器上，其中包括一個寫入 NVRAM 的快取
  - WAFL 設計者在特定的架構上執行時，利用前述之穩定儲存器，將隨機輸出入檔案系統最佳化
  - WAFL 的容易使用是它的指導原則之一
  - WAFL 的建立者也設計包括一個新的快照，此功能在不同的時間點產生多個唯讀檔案系統的副本

圖 14.12　WAFL 檔案佈局

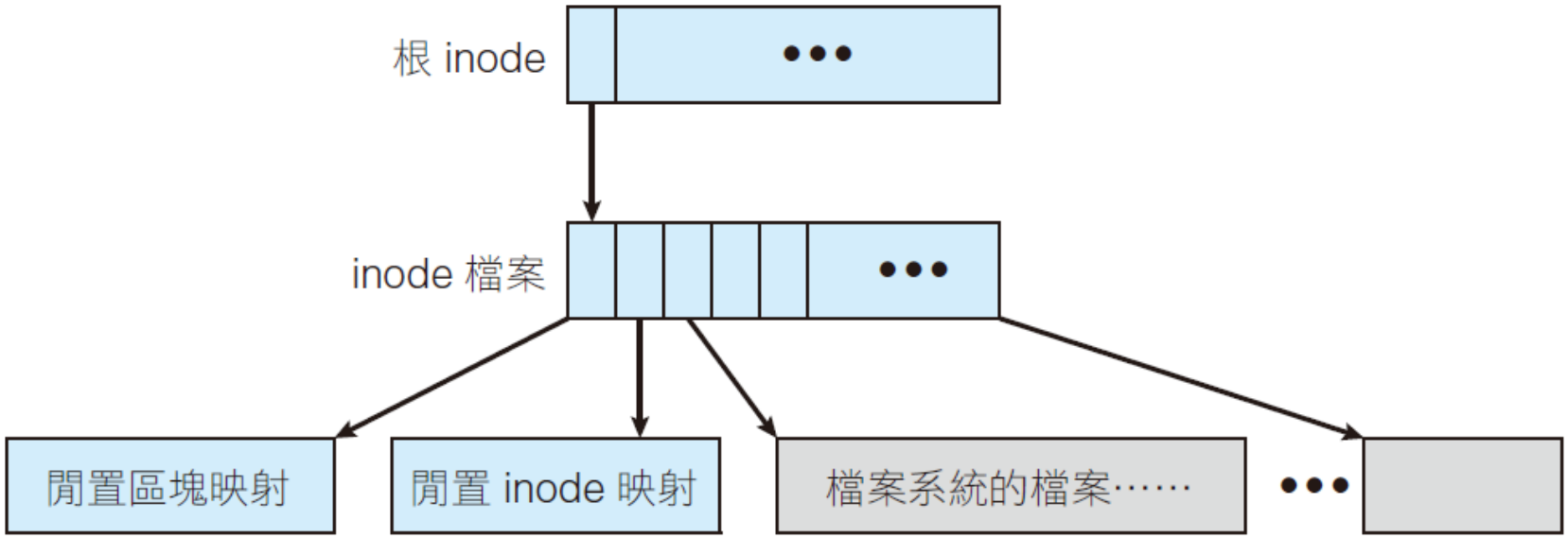
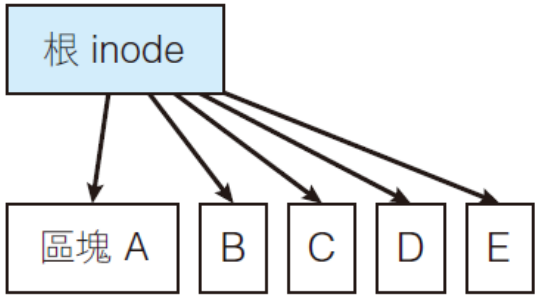
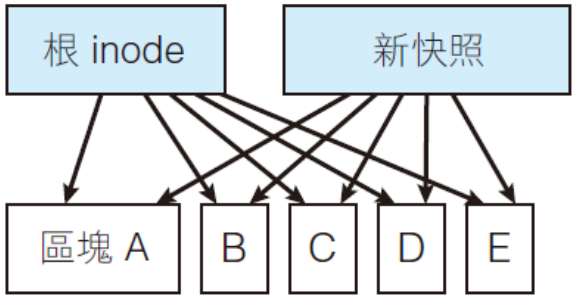


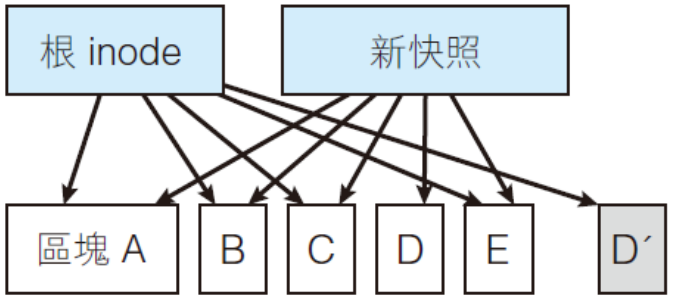
圖 14.13 WAFL 的快照



(a) 快照之前



(b) 快照之後，任何區塊改變之前



(c) 區塊 D 改變為 D´ 之後