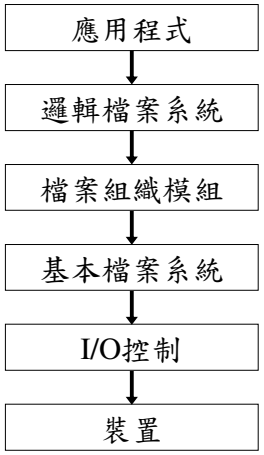
檔案系統實作

資科系
林偉川

檔案系統架構

- 檔案系統的目的
  - 為使用者或是作業系統本身提供大量的資料儲存空間
- 存取效率上的考量
  - 在主記憶體與磁碟之間的資料傳輸都是以區塊作為存取的基本單位。
  - 大部分的檔案系統都會將數個區塊集合成一個磁簇，並以磁簇(cluster)為單位進行檔案資料的存取。
- 磁碟主要可分為磁盤與讀寫頭兩大部分
  - 磁盤中的磁軌是用來儲存資料。
  - 讀寫頭是用來在磁盤上讀取或是寫入資料。

檔案存取



行程發出讀取檔案的需求，並提供檔案路徑

測試使用者對於檔案存取要求的權限

將邏輯檔案的路徑轉換為磁碟機的實體區塊位置

對磁碟機進行讀取實體區塊的要求

進行實際的 I/O 控制

從儲存裝置中實際讀出資料

3

檔案系統製作

1. 一個啟動控制區段 (boot control block)可能包含作業系統需要的資訊以便將作業系統從該分割區啟動。如果磁碟沒有包含作業系統，啟動控制區段可能是空的。通常它是分割區的第一個區段。在UFS(Unix file System)中，此區段稱為啟動區段，在NTFS中，它是分割區啟動磁區 ，
2. 一個卷區控制區段(volume control block)包含卷(或分割區)的詳細資料， 例如此分割區的區段數目、大小、未使用區段的數目、未使用區段的指 標、未使用FCB(File Control Block)數目與指標。在UFS中稱為超級區段;在NTFS稱為儲存主檔案表(master file table)。(MBR??)

4

檔案系統製作

3. 目錄結構被用在組織檔案。在UFS中，其中包含了檔案名稱和連接的inode數字。在NTFS中儲存在主檔案表。(92tpu1(d))
4. FCB包含許多檔案的細節，其中包括了: 檔案允許權、所有權、大小和資料區段的位置。在UFS中這稱為inode。在NTFS中資訊是儲存在主檔案表中，主檔案表使用關連式資料庫結構，一個檔案佔用一列。

檔案許可權
檔案日期（產生、存取、寫入）
檔案擁有者、團體、ACL
檔案大小
檔案資料區域或檔案資料區域指標

圖 11.2 典型的檔案控制區段（FCB）

5

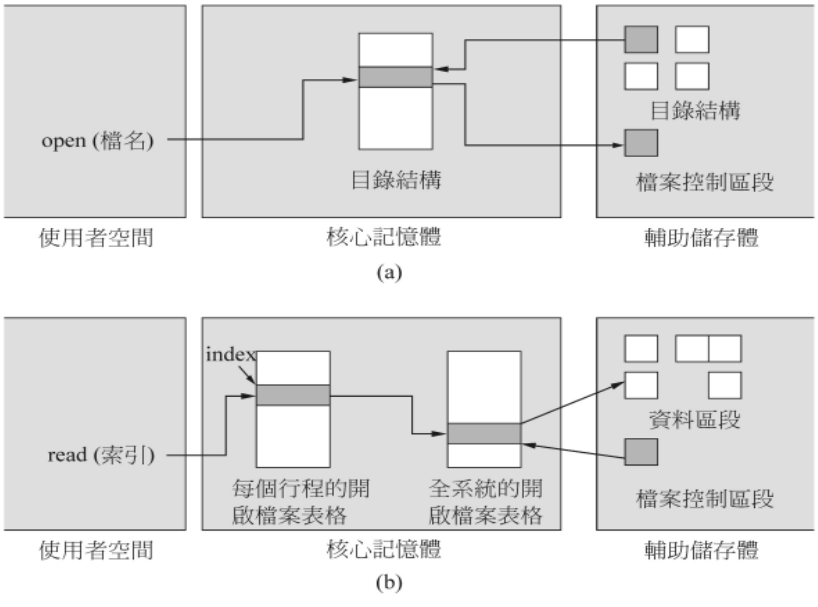
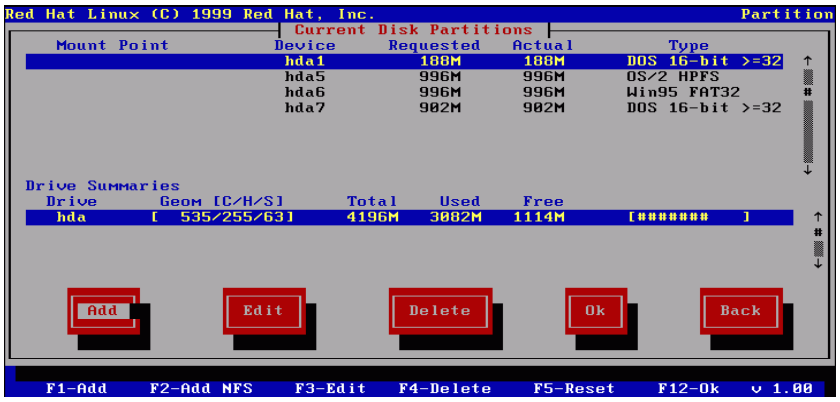


圖 11.3 記憶體中的檔案結構：(a) 檔案開啓；(b) 檔案讀取

6

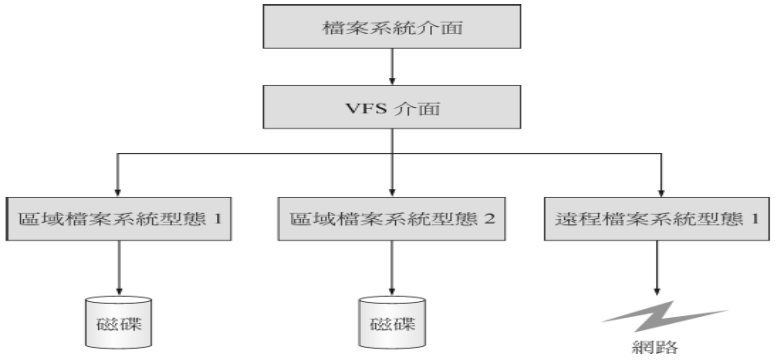
分割和安裝

- 根分割區(root partition)包含作業系統核心和其它系統檔案，它在啟動時被安裝(mounted)。
- 成功的安裝動作中，作業系統藉由要求裝置驅動程式讀取裝置目錄，並且驗證目錄中有所期望的格式來完成。如果格式不是有效，分割區必須做一致性檢查，並且有可能的話做修正。最後，作業系統在它的記憶體中之安裝表格記錄檔案系統已安裝，和此檔案系統的型態。



虛擬檔案系統

- 它藉由定義一項乾淨的VFS介面可將檔案系統的一般操作和製作方式分開。在同一台機器上可能其有不同製作VFTS介面的方式，使得局部架設的不同檔案可允許透明的存取。
- VFS對獨一無二的網路檔案提供一個機制。VFS是基於一種稱為vnode的檔案結構，vnode包含一項對於整體網路之檔案是獨一無二的數值指示器。整體系統的獨特性對於網路檔案系統的支援是必要的。核心為每一個工作的節點(檔案或目錄)維護一份vnode結構。



8

圖 11.4 虛擬檔案系統的圖形表示

開啟檔案表

- 檔案系統要進行檔案存取的動作
  - 到目錄結構中搜尋指定的檔案
  - 開啟並記錄在一個開啟檔案表中
- Linux 系列的作業系統
  - 每個被開啟的檔案會有一個相對應的檔案描述器 (file descriptor)
- Windows 中有類似檔案描述器的功能
  - 檔案處理器(file handler)

9

開啟檔案表

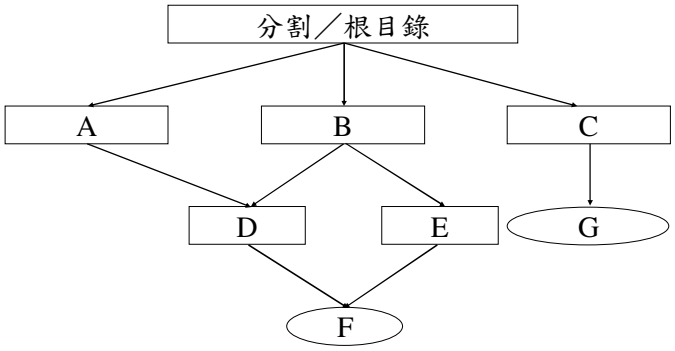
- 當資料有所更改時
  - 先在此開啟檔案表中進行修改，待檔案正常關閉之後，才會將這些資訊寫入磁碟中。
- 開啟檔案表相關資訊
  - 檔案名稱、存取權限、存取日期以及檔案指標等

索引	檔案名稱	存取權限	存取日期	其他資訊	檔案指標
0	main.c	RWX R - - - - -	2001/12/1	…	
1	mail.txt	RWX RW - R- -	2002/1/23	…	
2	a.out	RWX RWX RWX	2002/4/10	…	
…					
n					

10

共享檔案

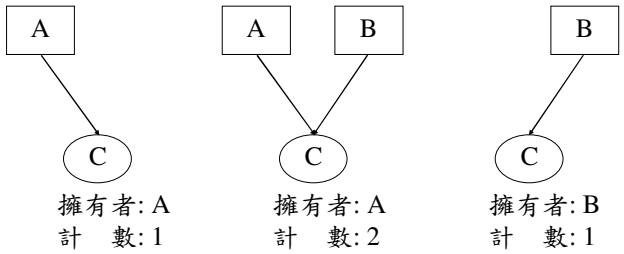
- 共享檔案
  - 一個檔案可能會被鏈結在兩個以上的目錄。
  - 如圖目錄 D 同時被目錄 A 及 B 所鏈結。



11

共享檔案

- 任一個鏈結到此共享檔案的目錄中將此檔案刪除時，不會將此共享檔案真正地從磁碟實體區塊中刪除，而只是將此計數減 1 。(ln指令)
- 當計數等於 0，才將共享檔案刪除。



12

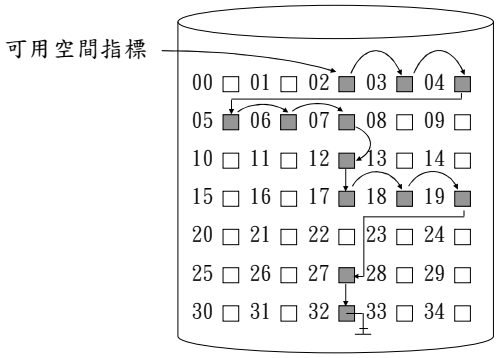
可用空間管理

- 可用空間管理機制
  - 讓應用程式在新增檔案時，可以從未使用的空間中找到足夠的空間來存放資料。
- 常用方法
  - 鏈結串列(linked list)
  - 位元向量(bit vector)
  - 計數(counting)

13

鏈結串列

- 將所有的可用空間利用指標串連起來。
- 使用一個開頭指標指向此可用空間鏈結串列的第一個區塊。



14

鏈結串列

- 主要缺點
  - 搜尋效率不佳必須循序地讀取才能得到所有可用空間的區塊資訊。
- 常用解決方法
  - 犧牲第一個區塊的磁碟空間，將其他所有可用空間的實體位置以索引的方式儲存在此區塊中，以加速可用空間的搜尋。
- 若以磁簇為資料存取單位
  - 可降低額外的紀錄空間(儲存指標)。
  - 可提高存取的速度。

15

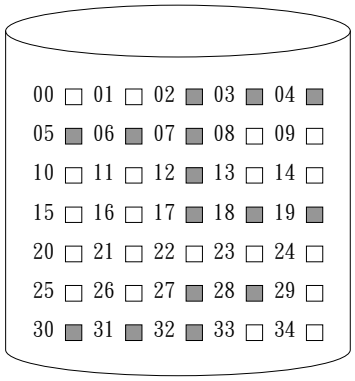
位元向量

- 每個區塊都是一個獨立的單位，並以一個位元來記錄該區塊是否為可用空間。
- 若一個磁碟中的第 1、2、4、6、10、11、12、19、23、24、25 個區塊屬於可用空間(0)，其可用空間的位元向量可表示為：
  0010101110001111110111000.....
- 若以磁簇為基本存取單位，則可節省更多額外的紀錄空間。

16

計數

- 可用空間表中的主要紀錄只包含了磁碟位置和計數值
- 如圖從第 2 個區塊開始，有連續 6 個可用空間區塊



索引	起始值	計數
1	2	6
2	12	1
3	17	3
4	27	2
5	30	3

17

計數

- 可用空間所在的位置過於分散，也就是外部斷裂(external fragmentation)的情形較嚴重時，這種可用空間的記錄方式反而會比較浪費空間。

18

檔案配置方法

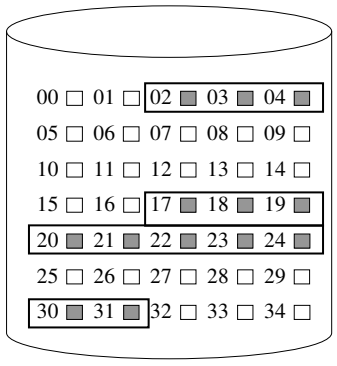
- 檔案配置時，必須考慮的重點
  - 磁碟空間的有效利用
  - 檔案能否被快速地存取
- 常用的三種磁碟空間的配置方法(96nttu6, 94tpu5)
  - 連續式配置 (contiguous allocation)
  - 鏈結串列式配置(linked list allocation)
  - 索引式配置(indexed allocation)

19

連續式配置

- 搜尋適合可用空間有下列幾種常用的方法：
  - 最先適合(first fit)
  - 最佳適合(best fit)
  - 最差適合(worst fit)

檔案名稱	起始值	記數
address	2	3
list	17	8
tree	30	2



20

連續式配置

索引	起始值	計數
1	2	6
2	12	1
3	17	3
4	27	2
5	30	3

以FF、BF、WF配置方法，檔案需要(FF)3個區塊時?需要(BF、WF)2個區塊時?

21

連續式配置

- 連續式配置的主要問題
  - 搜尋可用空間的問題
  - 磁碟空間的浪費問題 因多次的使用與歸還產生
    - 外部斷裂➔磁碟重組
    - 內部斷裂➔因以磁簇為基本單位為32k而只用1k
- 利用鏈結串列式配置可以解決問題

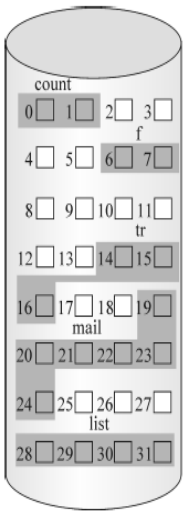
22

連續式分配法

- 連續分配法需要令每個檔案佔用一組磁碟上的連續位址。磁碟位址定義了一個線性的磁碟上排列次序。此時的順序 (假設只有一個工作對磁碟做存取)，在存取b區段之後存取b+1區段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區段移至另一磁柱的第一區段)，也只是移動一條磁軌而已。
- 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 (即最終需要搜尋時)。IBM的VM/CMS作業系統使用連續式分配，因為此方式提供較好的效率。

連續式分配法

1. 最主要問題在判斷一個檔案需要多少空間？當建立一個新檔案的時候，必須找到且分配必須的空間
2. 預先分配其空間可能是沒有效率。對一個長期成長緩慢的檔案而言，必須為其最後大小分配到足夠的空間，縱使大部份的空間長久都沒有用到。所以最後檔案有大量的內部斷裂。



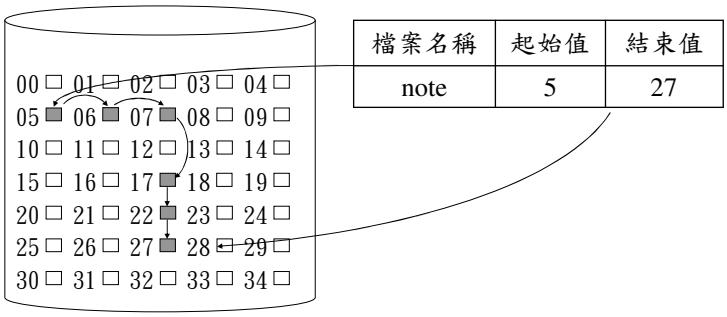
目錄

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

圖 11.5 磁碟空間連續分配

鏈結串列式配置

- 如下圖所示，檔案開頭在第5 個區塊
  - 第5 個區塊指向第6 個區塊
  - 直到第 27 個區塊為止



25

鏈結串列式配置

- 優點
  - 不會有外部斷裂問題
  - 建立新檔案時並不需同時就宣告檔案大小
- 缺點
  - 檔案只能循序存取
  - 可靠度的問題
    - 若某個檔案的一個指標被破壞，整個檔案就無法再被存取
    - 可利用雙向鏈結串列(double linked list)來解決

26

鏈結串列式配置

1. 最大的問題是只能在循序存取檔案有效地使用。若要找出檔案第i區段，須由該檔案的起點開始並且循著指標找下去，直到找到第i區段為止。每次對指標的存取都需要做磁碟讀入。無法提供直接存取的能力。
2. 鏈接分配的另一項缺點就是指標本身所要佔用的空間。如果一個指標就佔用一個512位元組區段中的4個位元組，那麼磁碟空間的百分之0.78被利用在指標上了，而不是用於存放資料。每個檔案就需要更多的空間。
3. 更嚴重地是可信度的問題。因為檔案是由散佈在磁碟中的指標鏈接起來，一旦有一個指標錯了或被破壞，作業系統軟體的錯誤或是硬碟上的損壞都會造成指標錯誤指向

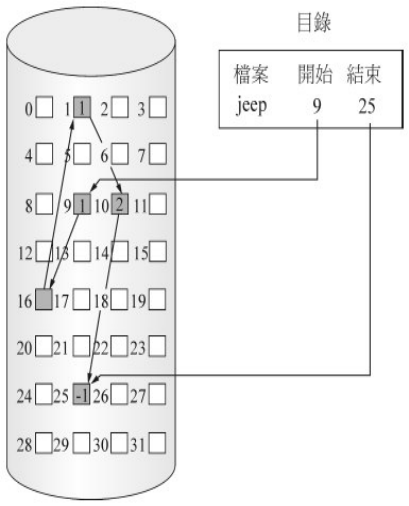


圖 11.6 磁碟空間的鏈接分配

鏈結串列式配置

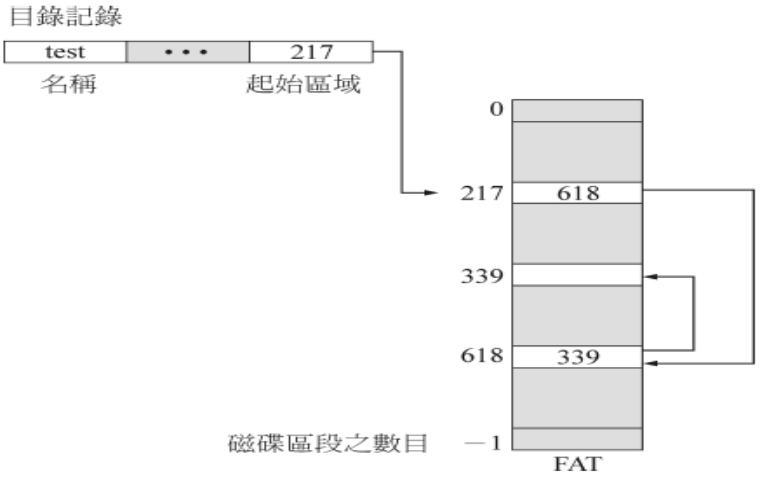
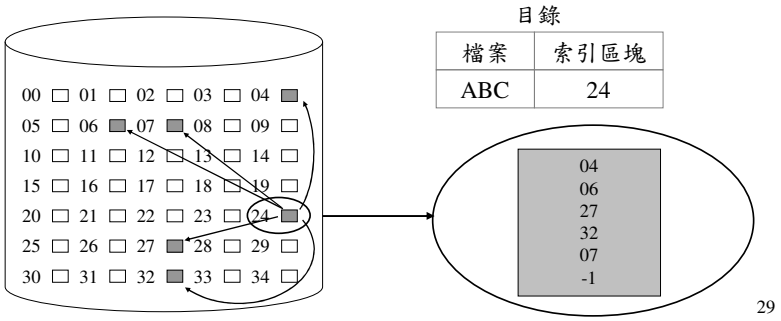


圖 11.7 檔案配置表格

索引式配置

- 解決鏈結串列式配置的無法隨機存取的問題
- 所有的區塊指標都集中起來存儲存在一個固定的索引區塊(indexed block)中(編號24)
- -1 代表整個檔案結束的符號



29

索引式配置

- 用鏈結串列式配置可解決外部斷裂和連續分配必須做檔案大小宣告的問題。但是，因缺乏一個FAT，鏈接分配無法使用直接存取，因為檔案的各個區段散佈在磁碟中。更重要的是指向各區段的指標也是散佈在整個磁碟中而且需要依序取出。索引分配把所有的指標都集中起來放在一個地方➔索引區段

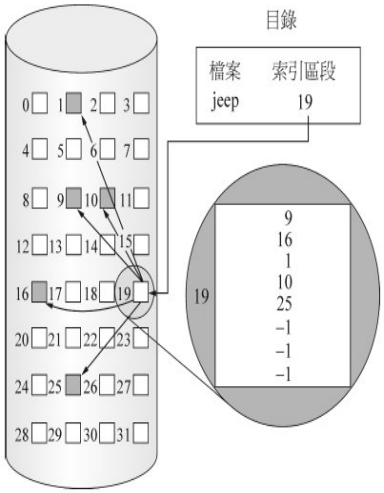
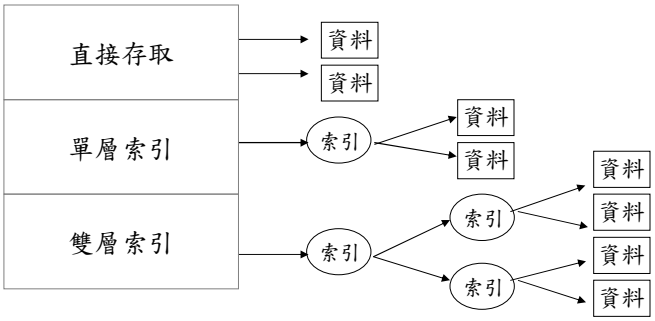


圖 11.8 磁碟空間的索引式分配

索引式配置

- 適度地控制索引區塊的大小，有下列幾種方法
  - 鏈結索引
  - 多層索引
  - 組合索引(如圖)



31

索引式配置

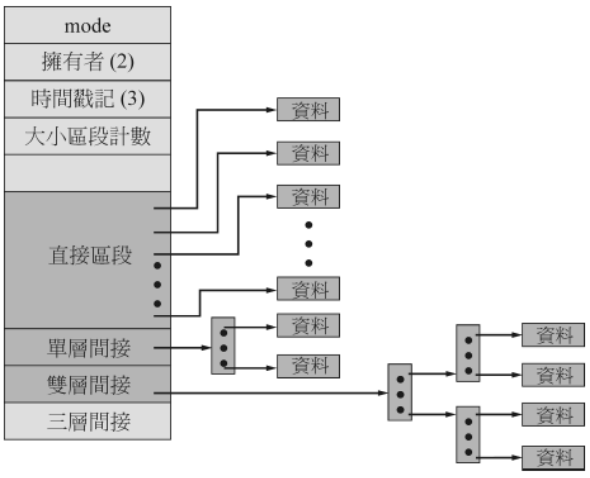


圖 11.9 UNIX 的 inode

32

檔案目錄實作

- 製作目錄結構時，必須考慮到存取效率與空間使用率的問題
- 較常用的方法
  - 線性串列(linear list)
  - 雜湊表格(hash table)

33

線性串列

- 將所有檔案以及子目錄都串連起來。
- 實作上比較容易，但不是很有效率。
- 增加檔案的搜尋效率
  - 利用 B 樹或 B+ 樹來實作。
  - 或利用多方搜尋(Multi-Way Search)的方式來搜尋檔案 。

34

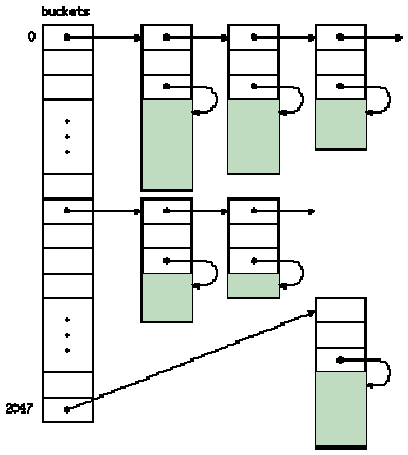
線性串列

- 製作一個目錄最簡單的方法是使用一個線性串列的檔名，並以指標指向資料區段。這種方法很容易寫程式但執行上很浪費時間。
- 線性串列所組成的目錄進入點之真正缺點在於線性尋找檔案。目錄資料經常的被用到，較慢的存取做法會被使用者注意到。
- 確保串列維持成排序狀態的要求也可能使新增和刪除檔案變複雜，因為必須移動大量目錄資訊才能維持排序的目錄
- 一個更複雜的樹狀資料結構 (例如B-tree)在這裏可能會有幫助。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。

35

雜湊表格

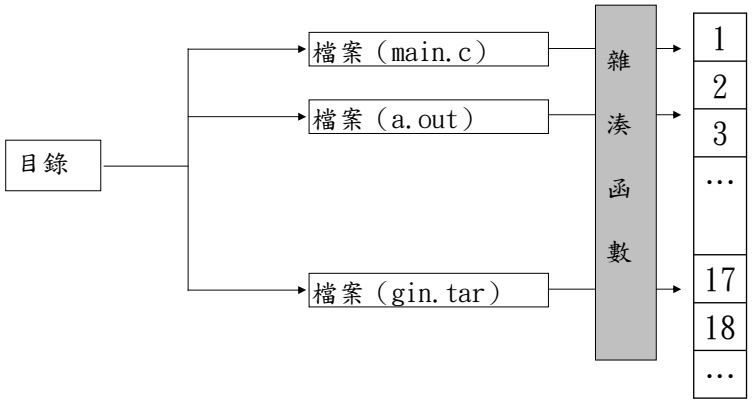
- 曾經用在檔案目錄的資料結構是雜亂表格
- 在這種方法中，一個線性串列儲存目錄的進入點，但同時也使用了一個雜亂方法的資料結構。雜亂表格從檔名計算出一個數值，並且傳回一個指標給線性串列中的檔名。因此它可以大量的降低目錄搜尋時間。



36

雜湊表格

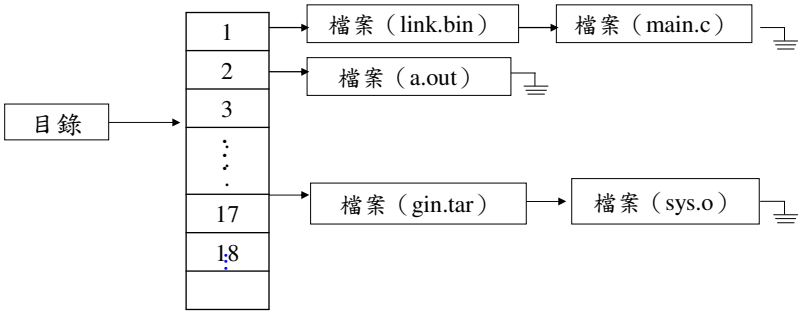
- 雜湊函數輸入值：檔案名稱
- 雜湊函數輸出值：某個數值範圍內的固定值
- 搜尋速度較線性串列快得多



37

雜湊表格

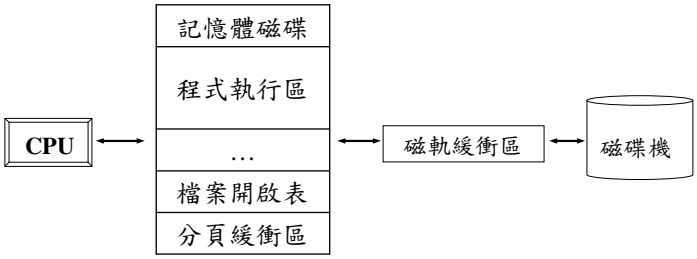
- 碰撞(collision)
  - 利用雜湊函數對不同檔名所產生的數值重複
  - 解決方法
    - 鏈結串列將所有雜湊數值相同的檔案都儲存在這個鏈結串列上 (如圖)



38

性能

- 增進整體檔案系統的性能
  - 利用快取記憶體，可減少磁碟讀寫頭的移動
  - 虛擬磁碟(virtual disk/ram disk)
    - 將一些可以快速存取的記憶體充當磁碟使用
    - 讓部分檔案存取速度可以像在快取記憶體中存取資料一樣快
    - 電腦關閉後於其中的程式也消失



39

性能

- 沒有一致性的緩衝區快取時，read()和write()系統呼叫會經歷緩衝的快取區。記憶體對映的呼叫需要使用兩種快取-分頁快取和緩衝快取。記憶體對映I/O是從檔案系統的磁碟區段讀入，然後存入緩衝快取區。緩衝快取區的檔案內容必須被拷貝到分頁快取區。這種情況稱為雙重快取，並且需要對檔案系統的資料快取兩次。這不只會浪費記憶體，而且還會因為記憶體系統內額外的資料移動浪費可觀的CPU和I/O週期。

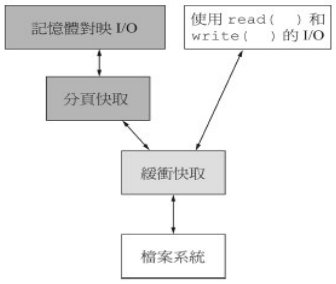


圖 11.11 沒有一致性緩衝快取區的 I/O

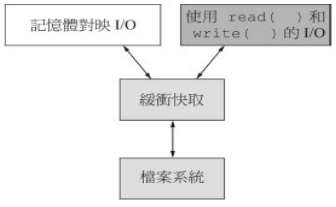


圖 11.12 使用一致性緩衝快取區的 I/O

效率

- 不同的磁碟空間配置方法各有其優缺點：
  - 將日期資訊直接儲存在檔案所屬目錄的鏈結表格中
    - 效率較差(檔案經常被修改)
  - 將日期資訊與檔案本身存放在一起
    - 缺點是在搜尋同一個目錄中的所有檔案時，可能要存取多個其他區塊的日期資訊，也會造成效率低落。
- 對整個檔案與目錄的結構作深入的分析，並採用其中最適合的方法來實作。

41

可靠性

- 備份(backup)
  - 將檔案系統中的所有資料複製一份，並存放在其他的儲存媒體中 (週期性備份)
- 還原(restore)
  - 將備份中資料存回原來的檔案系統中
- 差異備份(differential backup)
  - 執行過第一次的完整備份之後，之後就只需要將差異的那一部分儲存起來。 (短週期)
  - 缺點
    - 任何一次的差異性備份有所損壞或是遺失時，就無法還原成完整的檔案系統。
    - 每隔一個比較長的週期就要做完整備份

42

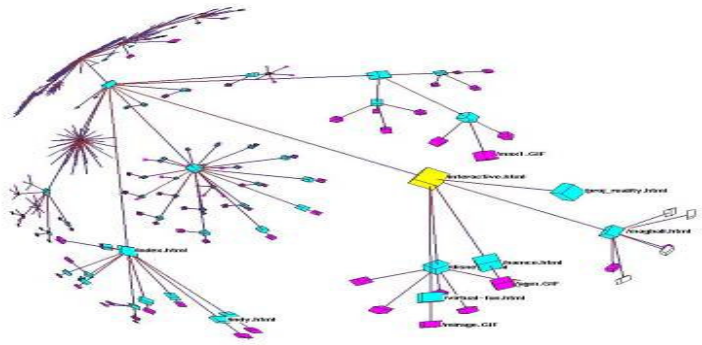
備份或復原

- 因為有時候會發生磁碟失效，所以必須小心以確保資料不會永久遺失。為了達到這個目的，系統程式可以用來從磁碟將資料備份到另外的儲存裝置(例如軟碟、磁帶或光碟)。
- 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)　為了降低所需要的備份，可以使用每一個檔案目錄進入點的資料。典型的備份時間表如下所示:
  - 第一天:從磁碟拷貝所有檔案到備份媒體，這種稱為完全備份 (full backup)。
  - 第二天:拷貝所有從第一天之後改過的檔案到另一媒體，這種稱為增加的備份 (incremental backup)。
  - 第三天:拷貝所有從第二天之後改過的檔案到另一媒體。
  - 第N天:拷貝所有從第n-1天之後改過的檔案到另一媒體，然後回到第一天。

43

登錄結構的檔案系統

- 演算法和技術會從原先的應用轉移到其它的應用領域。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。(registry)
- 所產生的製作被稱為登錄為基礎的交易傾向(log-based transaction-oriented或journal)檔案系統。



44

一致性的檢查(Consistency checking)

- 一部份的目錄資料保存在主記憶體(快取)以加速存取。在主記憶體上的目錄資料通常比相對應存放在磁碟上的資料要新，因為快取的目錄資訊在更新之後並不需要馬上寫入磁碟中。
- 考慮電腦毀損的可能結果。在這種情況下，開啟檔案的表格通常會遺失，因此任何在此目錄所開啟檔案的改變也跟著不見。
- 這種事件可能造成檔案系統成為不一致的狀態:有些檔案的實際狀態和目錄結構所描述的不一致。通常，在重新開機時會執行一個特殊的程式來檢查並校正磁碟的不一致。(不正常關機時會做的事)

45

一致性

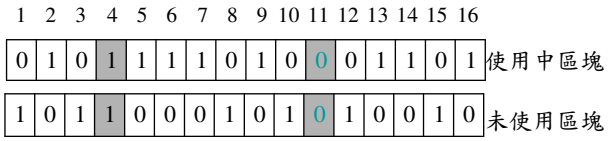
- 要確認檔案系統的一致性
  - 建立兩個包含所有區塊的表格：已使用、可使用
  - 檢查檔案配置以及可用空間的區塊記錄
- 正確的區塊檢查結果表

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	1	0	0	1	1	1	0	1	0	0	0	1	1	0	1	使用中區塊
1	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0	未使用區塊

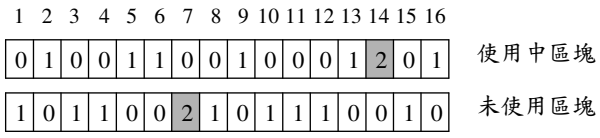
46

一致性

- 錯誤的區塊檢查結果表(重複指派與未指派) ➔0歸還可用空間，1則將之從可用空間刪除



- 錯誤的區塊檢查結果表(重複的區塊指派)
  - 編號7被可用空間重複計算➔重建所有可用空間
  - 編號14被使用空間重複計算➔一個區塊配給不同檔案?



47

摘要

- 檔案系統的實作
  - 實際存取流程
  - 可用空間的管理
  - 檔案資料的區塊配置
  - 檔案目錄實作
  - 檔案系統評估的介紹
- 對檔案進行讀寫
  - 從目錄結構中搜尋並開啟該檔案
  - 每個行程都會有一個開啟檔案表來儲存這些開啟檔案的紀錄

48

摘要

- 可用空間
  - 磁碟中沒有被用來記錄檔案資料的磁碟區塊
- 記錄可用空間的方法有
  - 鏈結串列
  - 位元向量
  - 計數
- 磁碟配置方法
  - 連續式配置
  - 鏈結串列式配置
  - 索引式配置

49

摘要

- 目錄結構較常使用的方法
  - 線性串列
  - 雜湊表格
- 評估一個檔案系統必須考慮
  - 性能
  - 效率
  - 可靠性
  - 一致性

50