Linux 上目錄暨標籤式檔案系統實作與研究

許洛豪
國立暨南國際大學資管系
brianhsu.hsu@gmail.com
廖祥智
國立暨南國際大學資管系
s1213003@ncnu.edu.tw
莊尚益
國立暨南國際大學資管系
s1213018@ncnu.edu.tw
江明勳
國立暨南國際大學資管系
s1213033@ncnu.edu.tw
姜美玲
國立暨南國際大學資管系
joanna@ncnu.edu.tw

摘要

隨著近年來大量容硬碟裝置的技術發展與價格下降，有愈來愈多使用者將各式各樣的文件與多媒體資料，以數位化的方式儲存，因此如何讓使用者在短時間內找到自己所需要的檔案，成了一大課題。然而傳統階層式檔案系統的架構，由於不夠靈活，使得使用者在進行檔案組織及管理時有諸多不變，本文嘗試利用『標籤』這個最早使用在管理電子郵件方面的概念，企圖解決這個問題。

標籤係指獨立於階層式目錄架構之外的屬性，一個檔案與目錄可同時具有多個標籤的屬性，使用者可以透過事先替檔案設定標籤的方式，在需要時以標籤進行條件式的檔案的搜尋，快速並且精確地找出所需的檔案。

關鍵詞：檔案系統 檔案搜尋 檔案管理 Linux 標籤

一、緒論

（一） 研究動機

隨著個人電腦的普及，有愈來愈多的家庭擁有並且使用個人電腦，以及其他相關的數位化設備，包括了數位相機、隨身碟和 MP3 Player… 等。並且隨著相關軟硬體技術的成熟，在現在這個時代中，以電腦進行文書處理、試算表或是簡報等工作，也是家常便飯。同時因為電腦的便利，使用者數位化的資料也愈來愈多，包括從早期的文件、試算表等檔案，到現在經常使用的 MP3 甚至是 MPEG 這類的影音視訊，有愈來愈多的使用者將大量的檔案儲存在大容量的硬碟之中。

在面對如此大量的檔案時，要如何才能夠讓使用者分類這些檔案，並且快速地在這麼多檔案之中找到自己所需要的檔案，將會成為一個無法避免的課題。

（二） 研究問題

在現存的檔案系統中，大多數都採用了所謂的『目錄（資料夾）樹』的分類方式，也就是採用階層式的方式來進行檔案的整理以及規劃。在這樣的架構中，最上層是根目錄(root)，而每一個檔案都只屬於某一個特定的分枝[2]，如下頁圖 1-1 所示。

這種目錄樹的方式，被廣泛地應用在各種作業系統上的檔案系統，包括 MS Windows 上的 FAT/NTFS 以及 Linux 上的 Ext2/Ext3 等等。在

這樣的架構中，一個檔案只屬於一個目錄。但是對使用者而言，一個檔案不一定只會在一個分類，把整個架構設計成一個檔案只能有一個分類，是不恰當的。

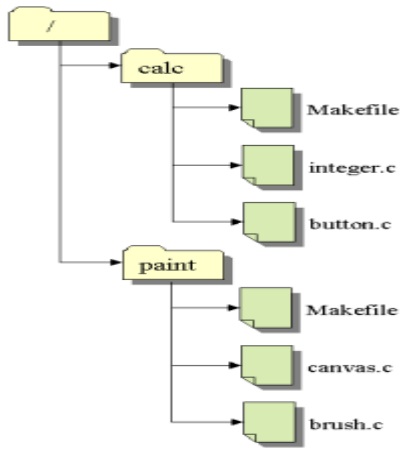

圖 1-1 [3]：目錄樹架構圖

如前所述，在使用者所擁有的檔案愈來愈多的電腦系統中，要如何才能夠快速並且精確地讓使用者找到自己所需要的檔案，是一個必須解決的問題。在階層式的檔案架構中，使用者可以利用目錄分類，做為尋找檔案的依據，但也有許多使用者在儲存檔案時，不會特別規劃自己的磁碟架構，而是一股腦地往『桌面』或『我的文件』資料夾存放，等到時間一久，要整理出自己所需要的檔案，就要花費更久的時間。

然而大多數的作業系統，都有實作出以檔名做為關鍵字的搜尋功能，使用者可以輸入他所記得的檔名，作業系統會檢查整個檔案系統或使用者所指定的目錄節點，並且把相符合的檔案列出，告知使用者。如 Windows 下的搜尋功能，以及 Un*x 系統下的 find 指令等。

但在進行這項動作時，是相當耗時的，以標準的 Linux 檔案系統 Ext3 為例，其檔案是切割成數個 block，存放在磁碟各處，最後再用一個 inode 資料結構，以 index 的方式來指出檔案位於磁碟中的哪一個位置。磁碟中則有一 super block 儲存了根目錄的 dentry [4] 資料結構，這個資料結構包含了根目錄下檔名與其 inode 相關的對應資料。若要對根目錄下的子目錄進行存取，則必需要再重新產生一個 dentry 資料結構，並填入子目錄與子目錄下檔名與 inode 間的對應資料，過程如圖 1-2 所示[4]。由此可見，完整地對磁碟進行以檔名為基礎的搜尋，是相當耗時的。

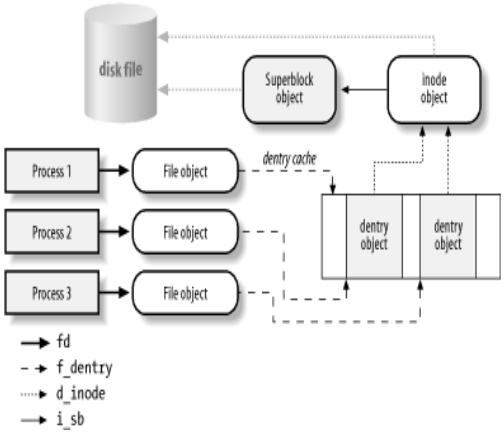

圖 1-2 [4]：Process 搜尋檔案的過程

即便在一個規劃良好的樹目錄之中，要尋找檔案也受到相當大的限制，只能依照原有的規劃來尋找檔案，無法靈活地組合出各種使用者可能想要尋找的資料組合。

為了更清楚地描述傳統階層式檔案系統的缺失，我們將用一個規劃過的階層式檔案架構來舉例，提出一些我們希望能夠達到的目標，以及現有的一些可能解決方案，最後再提出我們自己的解決方案。在這個範例之中，使用者的檔案是作業系統的種類來進行分類，其下再分成 Software 以及 SourceCode 兩個子目錄，分別存放軟體的可執行檔以及其原始碼。整個目錄樹架構，如下頁圖 1-3 所示。

在這樣子規劃的檔案系統中，若使用者要進行較為複雜的搜尋，將會無法有效並且精確地找到自己所需要的資料。以圖 1-3 所規劃的檔案存放

架構，舉例條列說明如下：

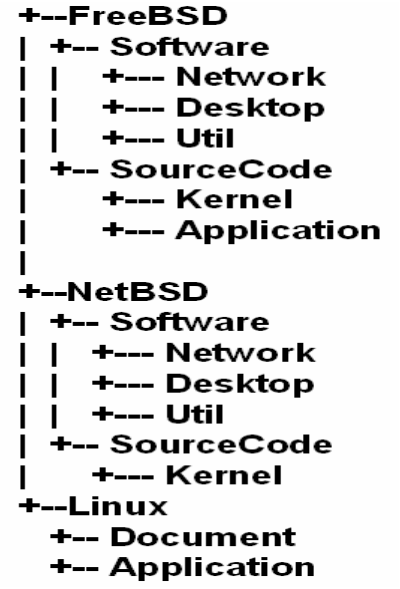

圖 1-3 ：範例目錄樹架構

- 使用者如何在現有的軟體分類架構（依照作業系統以及其軟體種類）中，找尋特定授權合約（例如 GPL 或 BSD）的軟體？

- 使用者如何能夠一次就找出並且瀏覽所有 NetBSD 的桌面應用程式以及所有 FreeBSD 的中的網路相關應用程式？

- 使用者如何能夠找一次找出找出並且瀏覽所有 NetBSD 與 FreeBSD 的網路相關應用程式，以 Linux 的 Application ？

上述所提出的搜尋問題，利用傳統的檔名搜尋方案，並無法精確地找出使用者所所需要的檔案，在下一章中，我們將會針對現有的檔案管理及搜尋方式加以說明，並說明其中不足之處。

二、相關研究

在這一章中，我們將針對檔案搜的問題，描述現有的解決方式，並說明其中的不足之處，再於第三章中提出我們的解決方式。

（一） 檔名搜尋

如前章所述，使用者要在眾多的檔案中尋找到所需的檔案，使用檔名進行搜尋是最常見的方式，各種作業系統也均有提供相關的功能。使用者可以透過這些程式，尋找特定檔名的檔案。

然而利用這一種方式進行搜尋，主要有兩項缺點：精確性的不足，以及搜尋過時間於冗長。在精確性方面，由於使用者只能依靠檔名進行搜尋，除非使用者能夠知道完整的檔名，倘若使用者使用了萬用字元，則搜尋程式將無法精確的只找到使用者所需要的檔案。而在搜尋速度方面，目前大多數以檔名為關鍵字的的搜尋程式，都是當使用者進行搜尋後，才去掃描使用者所指定的目錄及其子目錄，因此將會花費大量的時間在讀取磁碟內容上。特別當使用者不記得當初檔案所儲存的位置時，搜尋程式就必須從磁碟的根目錄開始找起，所花費的時間更是可觀。

另一方面，對於從數位相機等此類裝置所匯出，其檔名不具意義（通常為流水編號）的檔案，這樣的搜尋方式也無法有效地替使用者找到所需的檔案。

（二） Link

在 UNIX-like 系統下，提供了一項檔案管理的方式，稱做 link（即 Microsoft Windows 下所提供的『捷徑』功能），使用者可以視 link 為原始檔案的『別名』。

使用這個方式的最大缺點，在於系統無法自動同步，例如在使用 symbolic link 時，若使用者刪除原始檔案， link 隨之無效。倘若使用 hard link ，則使用者必須刪除所有的 hard link 以及原始檔案，方能釋放出該檔案所佔用的磁碟空間[5]。因此當使用者的分類日漸增多，如何維持 link

與原始檔案的一致性，將會成為一個問題。

另外，使用 link 的方式，當主要的分類目錄有檔案新增時，亦無法立刻反應在使用者所建立，用來存放 link 的目錄內容上。回顧圖 1-3 中所規劃的檔案分類架構，倘若使用者新增了一個 /NetBSDSoft 的目錄，並且將圖 1-3 中所有 /NetBSD/Software 底下所有檔案，以及其子目錄下的檔案，均在 /NetBSDSoft 建立連結。但往後若使用者新安裝了 NetBSD 的相關軟體，必須重新至將其連結至 /NetBSDSoft 目錄，才能夠維持 /NetBSDSoft 目錄與主目錄樹的同步。

（三） 應用程式

第三種方式，是利用各軟體開發商所開發出的應用程式，例如使用 iTunes 管理音樂，Picasa2 來管理照片等。

使用這類方式的主要缺點，在於使用者必須學習新的操作環境，以及得要依照相對應的檔案類型，找尋適用的程式。另外，由於這類程式由於並未與作業系統以及檔案系統進行整合，因此當使用者利用其他的程式（例如 Microsoft Windows 中的檔案總管，UINX-like 作業系統中的 cp/mv 等指令）來進行檔案操作時，這類程式並無法自行得知相關的資訊，進而無法與使用者的磁碟目錄樹架構進行同步的動作。

（四） 全文檢索

除了上述的兩種作法外，還有另外一種作法，是對硬碟內的檔案建立索引，而使用者可以利用檔名以及內容中的相關文字，進行全文搜尋，這也是目前 Google Desktop Search 所採用的方式。

使用這種方式的 Google Desktop Search，可以在不到一秒內找出電腦內的檔案（我們實際使用的結果，在約 1.5GB 的資料內，以關鍵字搜尋，查出九筆相關資料的時間為 0.52 秒），並且 Google Desktop Search 亦被矽谷人士公認為一種偉大的功能。

僅管如此，這種方式也有相當大的缺點存在，在我們實際操作使用過後，發現其最明顯的缺點莫過於所尋找出的資料不夠精確。例如我們欲找尋 Linux 的相關說明文件，因此在 Google Desktop Search 之中輸入關鍵字 Linux ，並進行搜尋。然而搜尋結果其中大多數都是 Linux 核心原始碼，是我們不感興趣的東西，這些檔案會被列出來，只是因為它們所在的目錄以及其內容，有 Linux 字樣而已。

Google Desktop Search 這樣的處理方式其實並沒有問題，因為其定位就是『桌面全文搜尋系統』，但由這個例子可以看出，以關鍵字進行全文搜尋，其所得的結果，不一定能夠滿足使用者的需求。

三、系統目標、語意定意及實作分析

（一） 系統目標

為了解決上一節內所述，傳統利用檔名搜尋速度上的瓶頸，以及全文搜尋上精確性的不足，我們採用了標籤的概念，讓使用者在傳統目錄樹階層式管理的方式上，再加上以標籤進行管理的功能。

所謂的標籤，是一種類似於目錄的概念，用來替檔案進行分類，但其中並無階層式的從屬關係，並且一個檔案可以擁有零至多個標籤。這個管理文件的方式，最先被 Google 應用在其所屬的 GMail 網頁式電子郵件服務，以及 Picasa2 圖片管理員式上。

在這樣的架構下，使用者可以在原有的目錄樹架構下，自由建立並且替檔案貼上標籤，滿足使用者替檔案進行多重分類的需求。在一個標籤管理良好的系統下，第一章第二節所提的三個搜尋問題，可以進而簡化成如下列所示的條件式搜尋判斷

式。

- 使用者如何在現有的軟體分類架構（依照作業系統以及其軟體種類）中，找尋特定授權合約（例如 GPL/BSD）的軟體？
  - 搜尋標籤：(GPL OR BSD)

- 使用者如何能夠一次就找出並且瀏覽所有 NetBSD 的桌面應用程式以及所有 FreeBSD 的中的網路相關應用程式？
  - 搜尋標籤：(NetBSD AND Desktop) OR (FreeBSD AND Network)

- 使用者如何能夠找一次找出找出並且瀏覽所有 NetBSD 與 FreeBSD 的網路相關應用程式，以及 Linux 的 Application ？
  - 搜尋標籤：((NetBSD OR FreeBSD) AND Network) OR (Linux AND Application)

經由以上的方式，使用者可以針對特定感興趣的文件，進行標記標籤的工作，往後即可靠標籤進行條件式的精確搜尋，並且由於尋找的範圍只限定於使用者曾經設定過的相關檔案與目錄，因此可以減少不必要的搜尋時間。

（二） 語意定義

若要提供使用者利用標籤的方式管理檔案，我們必須先詳細定義出整個標籤式檔案管理系統所使用的相關操作名稱及其語意後，使用者才能依照本系統所定義的作操來進行檔案管理。其語意如下頁表 3-1 所示。

當使用者利用表 3-1 的 addTag 此指令替檔案或目錄加上標籤的屬性後，當使用者對該檔案與目錄進行鏈結、搬移以及複製時，標籤必須要有相對應的行為，以維持標籤的一致性，茲整理如上頁表 3-2 所示。

由於在這套檔案系統中，使用者不僅能夠對檔案設定標籤，也可以對目錄進行設定標籤的動作，而其所隱含的意義為『目前此目錄下的所有檔案，以及將來在此目錄下所建立的檔案，都具有該標籤的屬性』。如此一來，使用者只需要對目錄進行設定標籤的動作後，將來在該目錄及其子目錄下所新增的檔案，即具有該標籤的屬性，就可以解決『相關研究』一段中利用 link 方式時，使用者無法在新增檔案時讓主目錄樹以及其他分類同步的問題。

（三）實作方式分析

接下來，我們將會針對要在系統中哪一層實作標籤功能，進行探討，並說明為何我們選擇將標籤功能整合至核心以及檔案系統中，而不僅僅只是開發上層的應用程式，來完成此項功能。

1. 在檔案內嵌入 metadata

這是目前最常用的方式，也就是在檔案格式內加入其餘的 metadata 資料，例如經常被使用在 MP3 音樂管理上的，就是 MP3 檔案格式內所使用的 ID3 metadata 資料。使用這種處理方式的好處，在於 metadata 的傳遞較不受電腦系統以及儲存媒介的限制，不論使用者將檔案存放在哪種媒介，也不論是否使用的是自己的電腦，只要管理程式有支援，就可以進行 metadata 的操作。

然而這種方式的缺點，在於現今各種型式的檔案，都已經有了標準的檔案格式（例如音樂的 MP3/WAV/WMA，影片的 MPEG/RM 等），我們不可能獨立發展另外一個通用的檔案格式，並且強迫使用者進行轉

換。

表 3-1 語意定義

操作	指令（函式）名稱	作用對象	說明
建立標籤分類	mkTag	標籤分類	新增一標籤分類，供使用者使用。
刪除標籤分類	rmTag	標籤分類	刪除標籤，並且將所有具有該標籤屬性的檔案，移除該標籤屬性。
新增標籤	addTag	檔案、目錄	對檔案或目錄新增該標籤的屬性，若是對目錄進行此動作，則目錄下所有檔案及子目錄自動繼承該標籤屬性。*
移除標籤	delTag	檔案、目錄	移除檔案或目錄內該標籤的屬性。
列出標籤分類	lsTag	系統	列出目前系統內所有的標籤分類。
列出檔案	lsTagFiles	標籤分類	列出系統內所有具有該標籤屬性之檔案。
列出標籤	lsFileTags	檔案、目錄	列出所有該檔案所具備的標籤屬性。
搜尋檔案	Search	查詢條件式	依照所用者給輸入之條件式搜尋式，回傳所有條件相符的檔案列表給使用者。

*當使用者將檔案自被標記該標籤屬性的目錄搬移至該目錄外，則該檔案不再具有該標籤之屬性。

表 3-2 檔案操作與標籤系統回應對應表

標籤對象 / 操作	檔案	目錄
鏈結	將鏈結標上相同標籤	將鏈結標上相同標籤
搬移	搬移後檔案保持原有標籤屬性	搬移後目錄保持原有標籤屬性
複製	複製後檔案不具有原檔案所標記之標籤屬性	複製後目錄及其子目錄不具有原目錄所標記之標籤
移動子目錄（檔案）至標籤目錄外	X	被搬移之目錄與檔案，不再具有因父目錄繼承而得之標籤屬性

但如果將我們的系統建構在原有的各種檔案格式上，將會面臨另一個問題，那就是各家檔案格式存放 metadata 的方式以及內容，均不相同，無法利用一個通用的處理程序來達成我們的目標。更甚至，如純文字檔此類的檔案，礙於其檔案架構的限制，是完全無法存放 metadata 的。因此，在我們的設計中，不採用這樣的方式。

2. 使用應用程式自行建立處理檔案庫

另一種可行的方案，是自行發展一套通用性的檔案管理程式（例如 Microsoft Windows 上的檔案總管），並且加入標籤管理的功能。這是看起來最直覺而方便的方式，其好處在於雖然其 metadata 資料無法如第一點所提的方案一樣，在不同程式上能獲得相同的結果，但也提供了相當程度的可攜性。只要使用者在不同電腦上使用相同的程式，metadata 即可進行轉移。

但是經過試用幾套相似做法的音樂與圖片管理程式（如 iTunes,Picasa2）後，我們發現利用這樣子的作法，在與目錄樹同步方面，會有相當大的問題。

以 iTunes 及 Picasa2 為例，兩者均要求使用者在使用前先『匯入』使用者所擁有的檔案，方能進行管理。在這一個方面，iTunes 的處理方式，是將使用者的檔案，移動到它自己所設計的目錄架構內，因此將會更動使用者原先設計用以分類的目錄樹架構。而 Picasa2，則是單純地對使用者的所匯入的檔案進行索引，不需要更改使用者原先的分類架構。

但不論是 iTunes 亦或 Picasa2 ，其最大的致命缺點在於兩者均不是整合在檔案系統與作業系統層面，因此無法隨時監控檔案樹的變動，並且立即產生相對應的動作。

舉例而言，當使用者利用 Microsoft Windows 檔案總管移動一個已經匯入 Picasa2 內的照片資料夾至新的磁碟位置時，使用者再次開啟 Picasa2 後，Picasa2 僅能偵測到原先所匯入的檔案『不存在』，因此不再顯示。當使用者將移動過後的資料夾再次匯入 Picasa2 後，其 metadata（例如使用者在 Picasa2 替照片貼上的標籤）無法復原到原先的狀態。

我們認為這是由於這些程式無法同步得知目錄樹相關變動資訊所造成的結果，因此，若要使用 Application 的解決方案，也隱含著我們必須利用其他的方式，在目錄樹變動之後，與之進行不對稱的同步。而這也代表著 Application 將要付出更多的心力，來維持自身的檔案結構與目錄樹檔案結構的同步。

同時，單純地發展新的 Application ，意味將會強迫需要使用標籤管理功能的使用者，放棄原先所慣用的程式，而造成使用習慣上的改變，增加使用者抗拒的心理[1]。

最後，僅僅開發 Application，就算最後將程式碼以自由軟體的方式釋出，第三方的程式開發者，依然無法簡地的重複使用我們的程式碼在自行開發的檔案管理程式上，必須想辦法將其中標籤管理功能的部份切出，並移植到自己的程式上。

3. 製做函式庫中介層，供其餘應用程式使用

第三種作法，是改進第二點直接寫成 Application 的作法，將標籤管理系統製成函式庫，做為其他應用程式與底層資料結構與相關操作的中介層模組。使用這樣子模組化架構的好處，是應用程式開發者所需花費的時間與成本，將會降低[6]，因此我們可以假設這樣的作法，會使得具有意願開發標籤管理功能的管理程式的第三方程式開發者增加。

然而這樣的方式，同樣地也面臨到第二點，採用自行實作 Application 時的問題，如檔案的同步，使用者的抗拒等。不過由於使用者所使用的檔案管理程式，有可能應用此函式庫，將檔案管理程式加入標籤管理功能，這麼一來，對於使用者習慣的衝擊，將會降到『低』的程度。

但是倘若使用者極需使用具有標籤管理功能的程式，其所熟悉的程式卻未採用我們所發展的函式庫，使用者也等同於被迫轉換至其他的軟體，使得使用者使用習慣的衝擊，與上述第三點實作 Application 一樣，成為『高』的程度。

4. 更動作業系統核心，提供相關功能

最後一種方式，是直接在檔案系統以及作業系統核心的層面進行標籤管理的設計與實作。用這樣的方式來達成標籤管理的功能，在四種方式中，其 metadata 的可攜性是最低的，因為使用者必須依附在特定的作業系統核心以及檔案系統上，才能使用標籤的相關功能。

舉例來說，當使用者將位於本機電腦使用 CatFS 磁區中，具有標籤屬性的檔案，複製到 Microsoft Windows 所使用的 VFAT/NTFS 磁區時，由於該檔案系統不支援標籤管理功能，因此會將標籤的屬性忽略。而即便使用者同樣使用 Linux 讀取該 VFAT/NTFS 磁區上複製過後的檔案，也無法取得標籤的相關屬性。

而在第三方發展相關應用程式以及使用者衝擊方面，因為我們會提供相對應的 system call 與函式庫，因此與第三項以 library 實作的方式並無太大的不同。

最後，也是我們選擇在檔案系統與作業系統層次實作的主要原因，是倘若我們將所有的資料結構、標籤搜尋以及其餘相關功能，整合進核心與檔案系統層實作後，則只要使用我們所提供的函式庫，就可以直接與檔案系統內的檔案樹架構進行同步，不需要花額外的精神處理相關的追蹤問題。

7. 綜合比較

綜合上述內容，我們對於以上四種方式，進行『Metadata 可攜性』、『綜合式檔案管理應用程式開發難度』、『使用者衝擊』以及『Metadata 處理速度一致性』進行比較，產生表 3-3 的結果。

其中『Metadata 可攜性』系指當使用者使用不同的電腦系統時，相關標籤資訊是否能夠完整保留，而『綜合式檔案管理應用程式開發難度』，指第三方應用程式供應者，在開發具有標籤功能、通用性檔案管理程式時，是否容易；『使用者衝擊』則代表了使用者在使用具有標籤管理的系統時，是否需要更改原先程式的使用習況；『Metadata 處理速度一致性』，則是看是否在讀取、寫入以及搜尋相關 Metadata 時，不同應用程式是否能夠具有相同水準的表現，不致於落差過

大。最後,『與原始檔案系統內容同步程度』,是指應用程式在使用者以作業系統提供的檔案操作工具進行檔案操作後，是否能夠即時反應出相關的變動。

表 3-3 ：原始實作方式評分表

	Metadata 可攜性	綜合式檔案管理應用程式開發難度	使用者衝擊	Metadata 處理速度一致性	與原始檔案系統內容同步程度
檔案嵌入 Metadata	高	高	中	低	中
Application	中	--	高	--	低
Library	中	中	中	中	低
核心層實作	低	中	中	高	高

由上表 3-3 可知，倘若將標籤管理這項功能，整合進檔案系統以及作業系統核心中實作，雖然降低了 metadata 的可攜性，但對於 metadata 處理速度的一致性以及與原始檔案系統架構的同步上，有正面的效果。

由於這個系統的主要目的，在於提供使用者一個『快速』並且『精確』的檔案管理以及搜尋模式，因此在實作上，我們選擇在先天設計上不受到與原始檔案同步問題困擾的方式，將這個系統在檔案系統以及作業系統上的層次實作，確保這套系統能夠完整而快速地對使用者在任何情況下所做的目錄樹改變，都能夠立刻產生正確的回應。

利用將系統架構在檔案系統以及作業系統層次，確保同步性的問題解決後，接下來我們會針對以檔案系統實作標籤管理系統的不足之處，說明我們的解決之道。依據使用者科技接受模式來看，系統導入的效果會受到易用性以及有用性認知的影響，而這兩項變數同樣會受到使用者使用習慣的影響[1]。表 3-3 中，我們可以發現直接將標籤管理功能於檔案系統中實作，第三方開發程式難度，以及對於使用者習慣的衝擊，仍然無法降到『低』的程度。

為了解決這個問題，我們提出了一個解決方案，稱做虛擬標籤目錄檔案系統（詳見第四章第二節），透過這一層需擬目錄,我們預期能夠降低第三方程式設計師撰寫支援標籤功能的程式的難度,以及使用者使用習慣上的衝擊,進而使得表一內的各項評分，成為表 3-4 的評分結果。在這個表中，我們可以看見,若是採用檔案系統實作的方式，整個評分優是優於其餘三種方式的,這也是我們採用實作檔案系統的主要原因。

表 3-4 ：採用虛擬標籤檔案系統後的評分表

	Metadata 可攜性	綜合式檔案管理應用程式開發難度	使用者衝擊	Metadata 處理速度一致性	與原始檔案系統內容同步程度
檔案嵌入 Metadata	高	高	中	低	中
Application	中	--	高	--	低
Library	中	中	中	中	低
核心層實作	低	低	低	高	高

四、系統架構

（一） 概述

如前章所述，我們針對實作方式進行分析後，決定將這個系統整合至核心層，以必免實作於應用程式層時，可能遇到的檔案系統內容與標籤系統一致性損毀等問題。同時我們也預計提供使用者虛擬標籤目檔案系統（下文所述之 VTagFS）及應用層程式，以減少使用者習慣上的改變，讓使用者衝擊降低。

在介紹我們的檔案系架構統前，必須要先介紹Linux 作業系統在處理檔案系統的問題上所採用的架構。在 Linux 中，為了支援各種不同型式的檔案系統，採用了 Virtual Filesystem Switch (VFS) 的方式，做為上層 system call API 與下層驅動層式的抽象介面層。

簡單的說，就是上層 Application 所呼叫，與檔案系統操作相關的system call ，都會經由 VFS 轉換至相對應的底層檔案系統驅動程式，做不同的操作，以符合檔案系統的規格。而使用者以及上層的應用程式，則無需理會下層所使用的檔案系統，可以使用一致的 system call 以及操作方式，對不同類型的檔案系統進行操作[4]。圖 4-1 即是上層的程式如何利用同樣的 system call ，對不同的檔案系進行操作的示意圖。

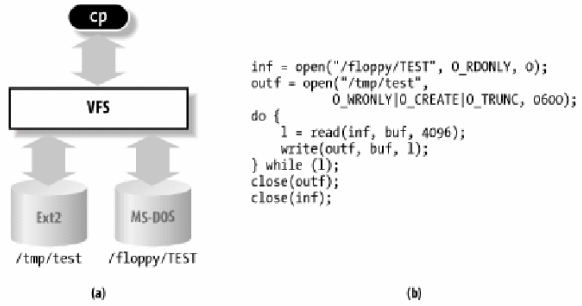

圖 4-1 [4] ：利用 VFS 做為抽象介面層

本系統依據此項特性，將系統切割為 Cat Filesystem Switch (CatFS) 與 VTagFS 兩部份，其中 CatFS 為一系列位於 VFS 內，與檔案系統結構無關的標籤操作 system call（詳見表 3-1 的語意定意），提供使用者對於檔案進行標籤系統的各項操作。

而 VTagFS 則為虛擬標籤目錄檔案系統，透過讀取 CatFS 所建立之相關資訊，產生一虛擬的檔案系統，讓使用者以傳統操作檔案的方式，讀寫利用 CatFS 所搜尋出的檔案。

整個系統如圖 4-2 所示，使用者經由 Application 層呼叫位於 VFS 內的CatFS 相關system call ，取得硬碟內所儲存的相關標籤資訊並且進行操作。而當使用者進行查詢時，則可利用 VTagFS ，將搜尋結果當成一般的硬碟磁區進行存

取。

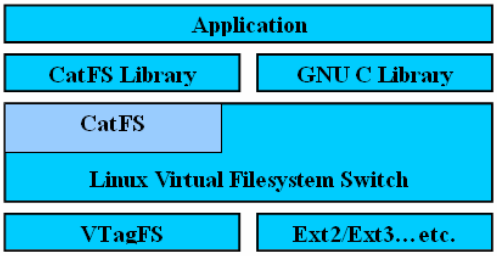

圖 4-2 ： CatFS 系統模塊圖

（二） CatFS

CatFS 是這整個系統內真正儲存使用者檔案所使用的檔案系統，而這套檔案系統能夠為使用者的檔案加上標籤的 metadata ，以利使用者進行檔案的搜尋。

為了要能夠讓使用在進行平台轉換時，不致於損失原先的資料，我們的 CatFS 將會與 Linux 目前的標準檔案系統 Ext3 有向下相容的能力。其原理是讓 CatFS 完全沿用 Ext3 的設計規格，不去更動任何現有的檔案儲存架構，而將 metadata 以檔案的方式儲存在磁碟分割區中。

從整個磁碟內部的儲存架構來看，在 Ext3 中，檔案是以 inode 儲存其所在的 block 的索引，而 super block 則存放了該檔案系統的根目錄 inode 以及dentry 的相關資訊，以指出根目下有哪一些檔案 [4] ，如圖 4-3 所示。

因此在 CatFS 中， 我們只是單純地將儲存標籤的 metadata 視做一個一般的檔案，擁有一個唯一的 inode 編號，並且在核心 CatFS 相關 system call 中，將其視為一般檔案，並且進行操作。並且修改 Linux VFS 相關 system call ，使其不會傳回任何關於這個 metadata 檔案的相關資訊，以隱藏實作細節。

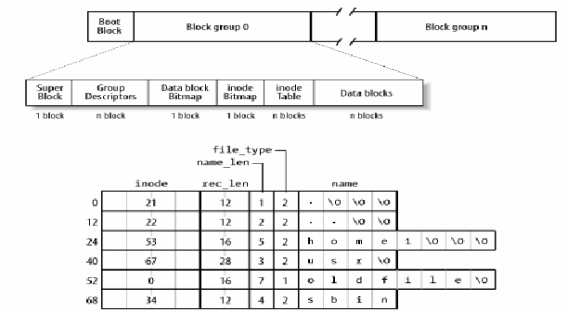

圖 4-3 [4] ： Ext3 檔名索引方式

在此同時，雖然我們選擇以 Ext3 檔案系統做為測試與實作平台，但由於 Linux Kernel 採用了 VFS ，將來只要能將此 metadata 檔案設計與檔案系統無相依性，即可套用在其他非 Ext3 檔案系統中。

在流程方面，當使用者開啟核心中 CatFS 的功能時，當使用者將一 Ext3 磁區掛載至目錄樹的同時，我們就會檢查該磁區是否具有 CatFS 所需要之 metadata 檔案。若有，則 CatFS 會負責將其讀取，並解析成位於記憶體中的相關資料結構，並與先前所掛載的磁區標籤資訊，進行整合。若該磁區無此 metadata ，則當使用者針對該磁區的檔案進行標籤操作時， CatFS 會自動替該磁區產生所需要的 metadata 。

由於讀寫硬體需要機械動作，速度遠較隨機存取記憶體來得慢[8]，為了能夠讓處理標籤系統的 metadata 檔案不至於過慢，在該 metadata 檔案我們採用了 B-Tree 這種可以大幅縮減磁碟 seek time 的資料結構[7]，儲存標籤與 inode 的相關對應。

（三） VTagFS

由於 Linux VFS 的特性，在 Linux 作業系統中，『檔案系統』並非真的需要實際的儲存媒介，而可以是『虛擬』的檔案系統[4]，不佔用任何的磁碟空間。底層的檔案系統，只需要填好相關的函式指標，以及相關資料結構的欄位，則不論下層的

檔案是不是真正儲存在磁碟中，上層的應用程式都可以將其視為一般的檔案系統進行存取。

利用這個功能，我們可以達到使用者不需要用任何特殊的程式，即可進行搜尋的功能，而上層的應用程式也不需要進行任何的修改，就可以把搜尋過後的結果，當成傳統 Linux 目錄樹下的一個目錄，進行讀取，寫入等檔案操作。

我們的做法，是撰寫一套虛擬檔案系統，稱為 VTagFS ，這套檔案系統架構在 Linux 所提供的 tmpfs 檔案系統上，本身不佔用任何的磁碟空間，而是將 RAM 切出些許空間，當成磁碟機使用。而 VTagFS 所要進行的動作是當使用將這套檔案系統掛載至目錄樹的某個節點時， VTagFS 將會分析使用者所傳入的參數(使用者所要尋找的標籤條件式)，並且至具有 CatFS 相關標籤資料的磁區，找尋符合條件的檔案，再將所找尋的檔案在該需擬磁碟內建立軟式鏈結，而Application 層的程式，只需要將尋找過後的結果，當做一般的檔案進行存取即可，整個流程如圖 4-4 所示。

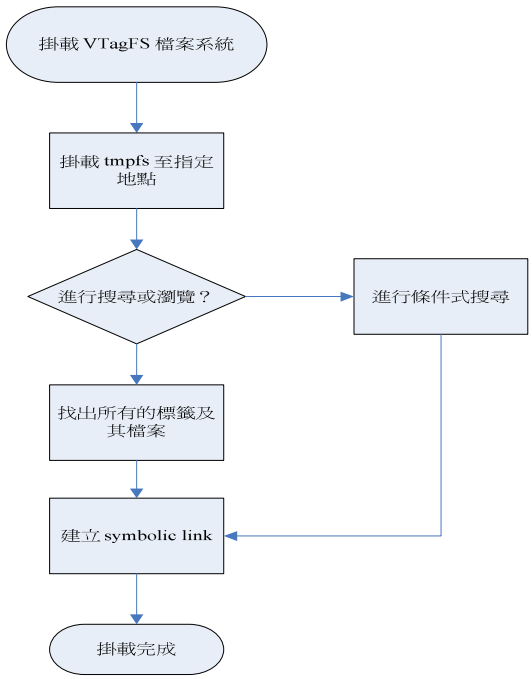

圖 4-4 VTagFS 掛載流程圖

透過這套 VTagFS ，使用者將可以不用學習新的指令，就進行標籤尋找的功能，並且將尋找的結果掛載在目錄樹中，使用自己所慣用的應用程式來進行瀏覽的動作。舉例而言，倘若使用者要搜尋所有的 FreeBSD 與 NetBSD 資料，那使用者只需要在命令列模式中，下達如下的指令：

```
mount –t vtagfs –o search=(FreeBSD OR NetBSD) none /mnt/search
```

使用者就可以在 /mnt/search 目錄中，觀看到搜尋的結果。當然，這整個掛載的動作，使用者也可以利用自己所習慣的圖型介面工具來進行。

最後，由於在整個系統架構中，我們預計提供標籤管理的函式庫和工具程式（包含建立、新增、移除與搜尋標籤等功能），上層的應用程式和使用者也可以使用我們所提供的函式庫以及工具程式，來進行標籤的相關操作，所以使用 VTagFS 的方式是選擇性的。

五、系統實作

（一） CatFS 記憶體內資料結構

為了在標籤內擁大量檔案時，能夠快速地找尋到特定檔案以及其相關資訊，我們特別設計了一套記憶體內的資料結構，減少字串比對的次數，以增加針對檔案進行標籤操作時的速度。

在記憶體內，我們以 hashtable tree 的方式來進行資料的儲存，其父節點為標籤的 hash table，並將檔案依每一層的目錄名稱進行雜湊計算後，將其以 tree 的方式進行儲存，則找尋特定檔案時所需的時間，則將受限於檔案路徑的深度，而與標籤內檔案的多寡無關。

利用此種方式，當檔案路徑前半部份相同時，因為只有後半部份不同的路徑，才會產生新的節點，因此也可以節省隨機存取記憶體的使用。

以 /first/second/A 和 /first/second/B 均位於 Test 標籤為例，其示意圖如圖 5-1 所示（簡化 hashtable 碰撞部份）。

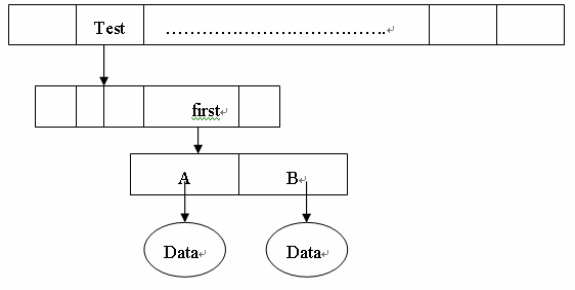

圖 5-1 CatFS RAM 中資料結構示意圖

（二） CatFS 磁碟內資料結構

在磁碟資料結構內，每一個磁碟分割區均會有相對應的 metadata 檔案，由於產生及讀取此 metadata 時，我們採用的是 VFS 的 system call，因此只要 Linux 有支援讀寫功能的檔案系統，就能加上 CatFS 的標籤屬性。

在檔案內容設計方面，考慮到磁碟相較於隨機存取記憶體，其存取速度較為緩慢，因此我們採取了將檔案設計成 B-Tree 的架構，並且以 B-Tree 的方式來進行存取，以減少磁碟的 seek time。

而當使用者將檔案系統掛載後，即會將 metadata 內容解析成上一節所述之記憶體內資料結構，詳細流程請見第五章第二節。

在成功分析並建立好隨機記憶體內的資料結構後，則所有使用者對於標籤所做的查詢以及搜尋，都是以記憶體為主，只有當使用者進行需要改變標籤內容的操作時，才會進行磁碟的寫入，借此減少對於磁碟的操作次數。

（三） 實際操作情形

以下為實際使用本系統時的情況，測試資料為 JPEG 圖檔，MP3 音樂檔以及 MPEG 及 RM/RMVB 檔影像檔，共約 1.5 GB，檔案總數 701 個。標籤部份分為五個標籤，圖 5-2 為使用 VTagFS 將虛擬標籤檔案系統掛載後，使用 GNOME 桌面環境所提供的檔案管理程式瀏覽該掛載點的情況。

從圖 5-2 可以看出，該檔案系統將系統內的標籤以目錄的方式列出，圖 5-3 與 5-4 則分別為使用終端機程式及檔案管理程式瀏覽特定標籤下的檔案，並且該檔案管理程式建立縮圖的功能，也可以正常使用。

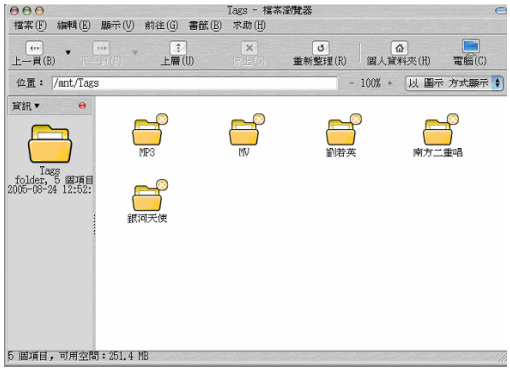

圖 5-2 掛載後的虛擬標籤檔案系統

圖 5-3 使用終端機瀏覽特定標籤

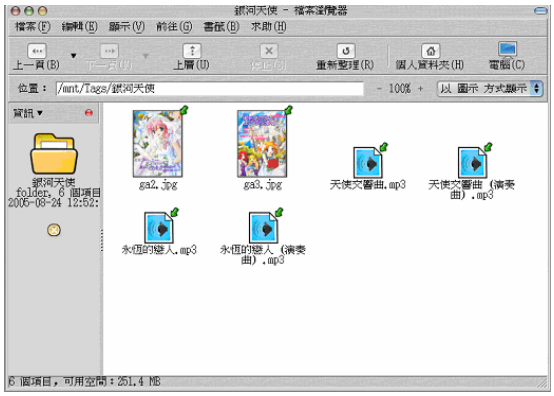

圖 5-4 使用檔案管理程式瀏覽標籤

由此可見，我們的標籤檔案系統，搜尋過後的結果，可以以一般檔案的方式呈現，使用者可以利用原先慣用的方式進行檔案的存取，進而減少使用習慣上的變更。

同時為了增加易用性，我們也實作出了增標籤系統整合自 GNOME 桌面整合環境中的檔案管理程式，使用者可以透過這個介面，建立、刪除標籤，或是替檔案新增或移除標籤屬性，如圖 5-5

所示。

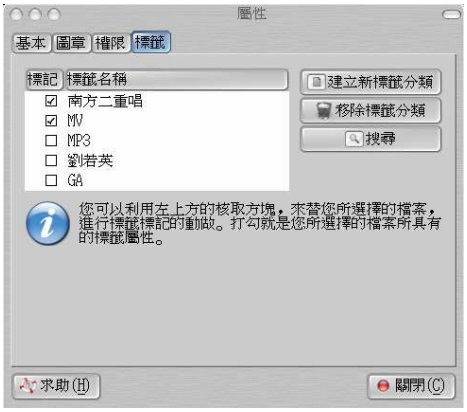

圖 5-5 整合至檔案管理程式的使用者介面

圖 5-6 分別為兩個不同的標籤，從圖中可以看出，相同的檔案（以紅色圓型外柱表示）可以位於不同的標籤內，而應用程也可以正常處理，產生縮圖。

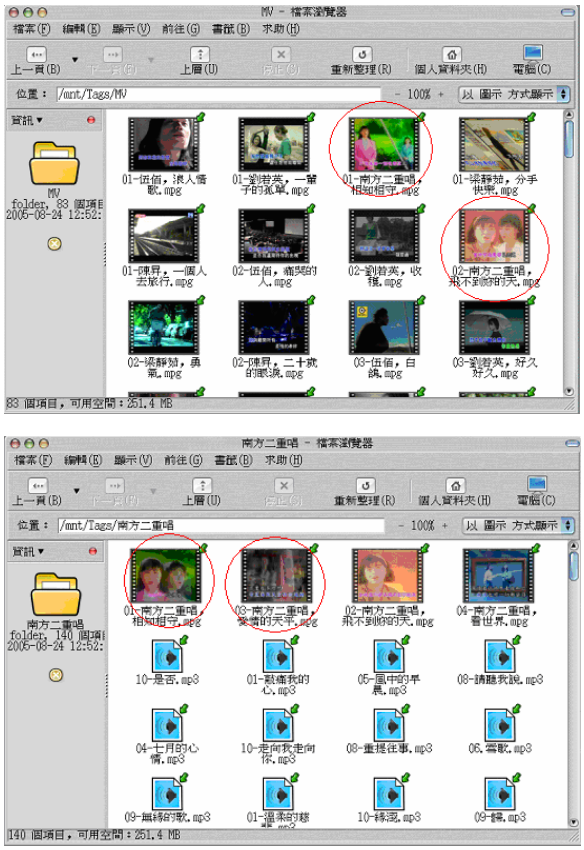

圖 5-6 同樣檔案位於不同的標籤內

接著圖 5-7, 5-8, 5-9 則顯示了本標籤系統可以與檔案系統進行同步，圖 5-7 為原先標籤內所擁有的檔案，圖 5-8 為使用原有的 rm 指令刪除該標籤內所有副檔名為 mp3 的原始檔案。

雖然並非刪除 VTagFS 掛載點內的檔案，但經過刪除檔案後，我們可以看見圖 5-9 內該標籤中，相關的檔案也一併移除，並不會有使用 symbolic link 時，造成無效的 link 的情況。

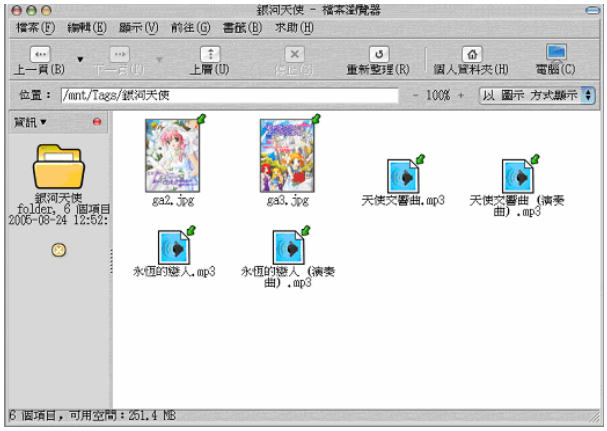

圖 5-7 原先標籤內共有六個檔案

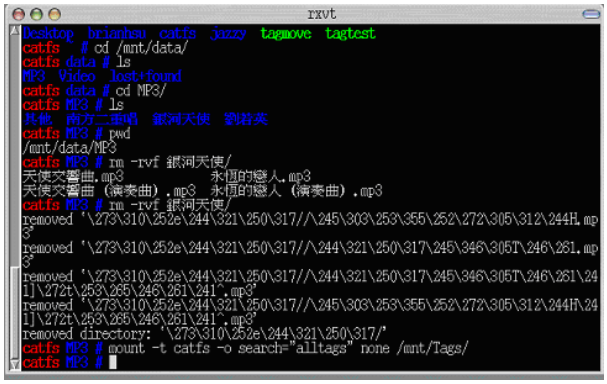

圖 5-8 於終端機內刪除原始檔案

圖 5-9 標籤內的檔案也同步變更

接著來看此標籤式檔案系統在條件式搜尋功能方面實際應用的情況，假設使用者將檔案依檔案

類型（音樂，Music Video，照片）分別存放於不同的目錄之內。若假如使用者欲找出特定藝人（此例為劉若英）的音樂與 MV ，則可以使用『(MV|音樂) & 劉若英』以及『(MV|劉若英)&(音樂|劉若英)』兩種搜尋條件式，達成相同的目標。圖 5-10 即為搜尋後之結果。

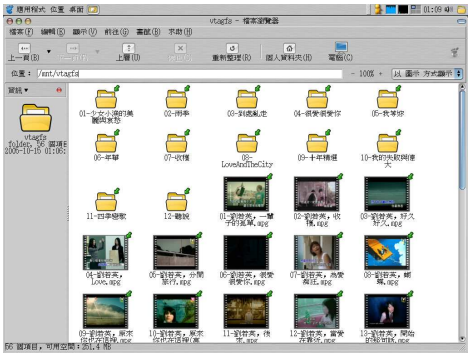

圖 5-10 條件式搜尋結果

六、效能測試

實作完後，接著要做的是驗證此系統的效能，我們利用五千個檔案，分別在原始核心以及修改過後的核心，進行檔案讀取以及搬移的動作。

在修改過後的核心方面，為了明檢測整個系統效能與具有標籤屬性的檔案多寡是否有關，我們每次測試時，分別針對 5000 個檔案樣本中的 0/1250/2500/3750/5000 個檔案附加標籤屬性之後，重新開機，並測試檔案讀取以及搬移的時間。在測試時，每一個分類分別做五次相同的操作，並取其平均值。

圖 6-1 是讀取檔案時間分組平均值的長條圖，在此圖中可以看出，當具有標籤的檔案愈多時，所需的讀取時間愈少。

會造成這種結果，是因為在 Linux 中，核心會將已經解析過的 dentry 資料結構建立 dentry cache ，以待下次使用[4]。而由於本系統在掛載磁碟時，即會透過核心內的 path_lookup 函式解析具有標籤屬性的檔案的路徑，以確定該檔案實際存在於磁碟中。

由於這個原因，結成讀取具有標籤的檔案時，其解析路徑的時間已被轉嫁至掛載磁碟的時間，而在進行測試時，少了解析 dentry 的時間，進而造成圖 6-1 的結果。

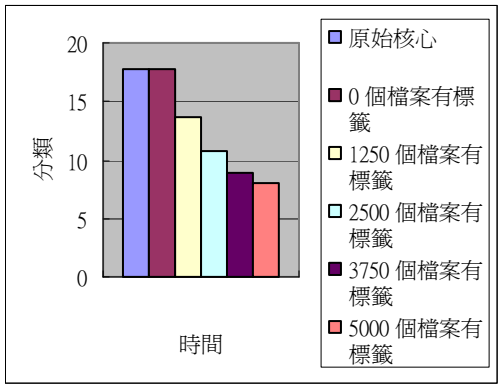

圖 6-1 讀取檔案時間長條圖

圖 6-2 為搬移檔案的時間長條圖，在這張圖中可以看到，當具有標籤的檔案數一多時，系統處理時間明顯增加。這是因為在這一個版本的實作中，我們尚未加入標籤操作的 buffer 機制，而由於搬移檔案時需要修改標籤內檔案路徑的資料，造成每搬移一個具有標籤的檔案，就需要進行相對應的標籤操作，並寫回磁碟，增加了系統的負擔。

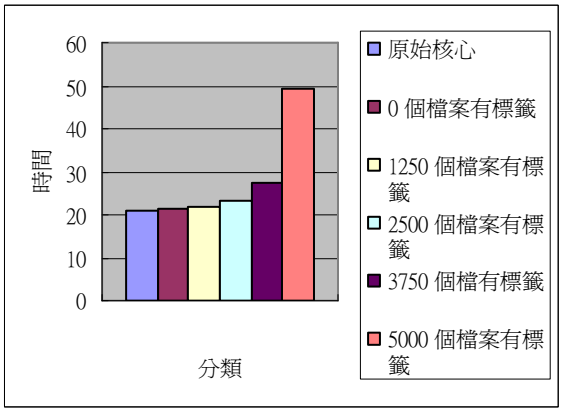

圖 6-2 檔案搬移時間長條圖

七、結論

雖然標籤的概念才被提出不久，但已經被證實是一種有效的管理方式，可以特定項目同時存在於不同的分類之下，增加自由度。而時至今日，標籤的觀念已經被廣泛地應用在各種場合之中，例如 Google 所推出的 Gmail 電子郵件服務、圖型管理程式，以及線上相簿 Flicker 之中。

由於標籤具有相當大的組合彈性，因此我們希望借由這套系統，可以替使用者建製出一套利用標籤來管理與搜尋檔案的解決方案。

借由標籤的建立是由使用者所自行控制，以及標籤多樣的組合性，我們希望能夠讓使用者透過

這樣的標籤示檔案系統，增加搜尋檔案的速度及精確性。

由於目前我們的系統僅止於實驗階段，因此一些諸如多人共用，檔案權限等問題並沒有包含在實作的範圍中，也希望將來有機會能夠發展成更成熟，更適合實際應用的版本。

八、誌謝

終於到了寫誌謝的時間，能夠順利完成這篇論文，首先要感謝姜美玲老師這一年來的指導，給了我們相當多寶貴的想法與建議，讓我們能夠快速地進入狀況，實作出這個系統，以及完成這一篇論文。

另外，我們也要感謝這一年來，許多在我們身邊的朋友，在一個又一個安靜的夜裡，我們在各自的題目上專研著，有歡笑，也有淚水。當我們因為系統的進度而感到快樂時，有你們可以分享，而當遇到了瓶頸，也有你們可以訴苦。

雖然大家總是互相揶揄著對方的進度與作品，但實際上大家都知道這是一種鼓勵與祝福，在相互批評的過程中，我們有了成長。也在這一連串的『互相漏氣求進步』中，做出了各自的系統，那份快樂是無可言喻的。

謝謝大家，要是沒有你們，就不會有這個系統以及這篇論文的誕生。

九、參考文獻

[1] 林東清, “資訊管理－e 化企業的核心競爭能力”, 智勝文化, 2002.

[2] A. Silberschatz, P. B. Galvin, et al, “Operating System Concepts.” John Wiley & Sons, Inc, 2003.

[3] Ben Collins-Sussman, Brian W. Fitzpatrick, et al. “Version Control with Subversion.” O’Reilly, 2004.

[4] Daniel P. Bovet, Macro Ceasti. “Understanding the Linux Kernel, 2nd Edition.” O’Reilly, 2002.

[5] Jeffrey Dan. “LPI Linux Certification In A Nutshell.” O’Reilly, 2003.

[6] Roger S. Pressman. “Software Engineering – A Practitioner’s Approach, 6th Edition.” McGraw Hill, 2005.

[7] Thomas H. Cormen et al. “Introduction to Algorithms.” The MIT Press, 2002.

[8] William Stallings. “Computer Organization & Architecture.” Prentice Hall, 2003