JFFS2 檔案管理系統可靠度之探討

*黃文增　　*陳俊達　　**陳錦杏　　*蕭榮修

W. T. Huang,　C. T. Chen,　C. H. Chen,　R. S. Hsiao

*國立台北科技大學電子系 **中台醫護技術學院資訊管理系

{wthuang, s3419002}@ntut.edu.tw, chchen@ctc.edu.tw, rshsiao@ntut.edu.tw

摘 要

JFFS2為Linux系統檔案管理之一，主要用以支援反及閘快閃記憶體存取機制。因快閃記憶體的物理特性限制，固有之檔案管理系統並非亦以運用，因此JFFS2油然而生。再者，快閃記憶體管理中，使用壽命及額外系統成本花費而導致的處理時間效能方面，是本文中所要探討重點。本文中將探討JFFS2管理機制，且將以反及閘快閃記憶體特性為基礎進而提出合適的管理方案，提高反及閘快閃記憶體管理效能；我們使用SMDK24 10X嵌入式發展系統作為驗證平台，以JFFS2作為比較基礎，驗證本文所提出之管理機制，進而以提供貼切合適的反及閘快閃記憶體管理系統。

關鍵詞：JFFS2、反及閘快閃記憶體、嵌入式系統

ABSTRACT

JFFS2 is a file system of Linux, in order to support the NandFlash management. Because of NandFlash characteristic limits, so inherent file system can not used. Furthermore, in NandFlash file system, the wear-leveling and the system costs aspect of processing time is the important issues. In this paper, we will discuss JFFS2 manage manners and base on the NandFlash characteristics offer suitable for NandFlash management to advance management effect.We use the SMDK2410X embedded system development platform to test and verify. JFFS2 is used as the comparison the foundation, identification our management performance and offer a appropriate file system for NandFlash.

Keywords: JFFS2, NandFlash, Embedded system.

一、 導 論

由於 Linux 系統具備公開性及充分發揮 CPU 之特點，並且提供多樣化檔案管理系統，所以眾多嵌入式系統中亦以此系統作為開發系統；再者，由於嵌入式系統產品輕、小及省電的特性，反及閘快閃記憶體作為儲存媒體最為適合[5] [13] ，簡稱為 NandFlash，如 SmartMedia Card、CF Card、SD Card [9] [19] [21] 。Linux 檔案管理系統中，提供 JFFS2 專作為快閃記憶體管理之用途，因此 JFFS2 管理將影響系統使用快閃記憶體的壽命及效能。我們首先研讀 NandFlash 基本原理、JFF2 管理機制及相關快閃記憶體探討資料[1] [3] [5] 在管理上可分為損壞區塊管理、區塊均勻使用管理以及零散資料頁收集回收等部分作探討[2] [10] 。因此，本文將以區塊均勻使用管理以及零散資料頁收集回收作為探討主軸，並且以 JFF2 作為比較基礎[25] ，SMDK2410X 作為驗證平台，驗證我們所提出的管理方式，可提供優越的可靠度及效能。

本文的架構安排如下。在第二部份中描述 NandFlash 的基本特性原理。JFF2 之探討於第三部份中討論。第四部份是主要研究結果。最後，第五部分為結論。

二、 快閃記憶體基本特性

NandFlash 將記憶體分割為數個固定的區塊(Block)，每個區塊切割成數個大小相同的資料頁(Page)。其中區塊是清除的基本單位，資料頁則是 I/O 的操作單位，也是 NandFlash 與主記憶體間資料交換的單位。NandFlash 基礎架構圖，如圖 一所示；由圖 一中可發現 I/O 匯流排(Bus)為 8-bits，資料頁由(512+16) Bytes 所組成，每 32 個資料頁組成一區塊單位(16K+512) Bytes [19] [21] 。

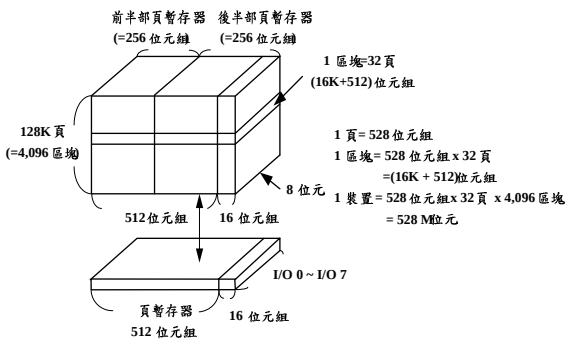


圖 一、NandFlash 基礎架構圖 [19] [21]

表一、NandFlash 規格特性 [19] [21]

NandFlash	特 性
資料暫存器單位	(512+16) × 8 Bytes
資料頁編程單位	(512 + 16) Bytes
區塊清除單位	(16K + 512) Bytes
隨機讀取時間	12us(Max.)
循序讀取時間	50ns(Min.)
編程時間	200us(Typ.)
區塊清除時間	2ms(Typ.)
使用程度	100K 編程/清除
資料保存期限	10 年

在表一中，將針對 NandFlash 的存取時間特性及基本限制作一探討，NandFlash 規格特性中，可歸納以下探討重點。

(一)、NandFlash，資料和位址線採用同一匯流排，實現串列讀取。

(二)、NandFlash 適合連續資料性讀取優點。

(三)、NandFlash 使用時不能覆寫，在新資料儲存之前，原位置的資料區塊必須先清除後，新資料才能重新寫入；而清除所花費的時間，是所有操作中最耗時的。

(四)、清除次數限制為 100,000 次。

關於上述第三及第四點兩限制，導致傳統式硬碟檔案管理策略不適用於快閃記憶體，因此 JFFS2 之目的在於提供一針對快閃記憶體的管理機制。

三、 JFF2 之探討及改善

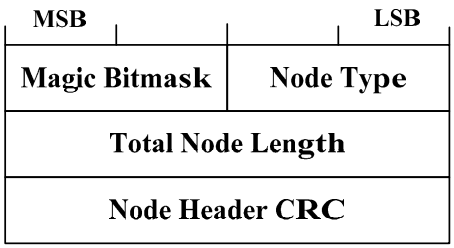
在 JFFS2 管理中，以超級區塊(Super block)方式記錄快閃記憶體資訊，包含快閃記憶體總容量、每區塊容量、區塊數及區塊使用狀況…等。除此之外，JFFS2 根據快閃記憶體每個區塊的使用狀況，以不同的列表(Linked list)分類管理。如未使用的區塊，以 free_list 列表來管理。JFFS2 所有的管理區塊種類之列表，如表二所示。

JFFS 最初是純粹以 Log-structured 日誌型檔系統（LFS）方式的管理系統，這意味著它基本上是一長列節點 [18] 。節點包含資料和仲介資料(metadata)，循序的存儲在快閃記憶體晶片上，在存儲空間中嚴格按照線性推進。在最初 JFFS 中，日誌中只有一種節點(jffs_raw_node)。每一節點對應一個索引節點(inode)。此節點的頭部包含它所屬的索引節點號和此索引節點當前檔系統的仲介資料。此節點還可能攜帶不定長度的資料。即使節點所屬的索引節點相同，它們的順序也是不同的。每個節點都保存一個版本號。每個節點都比和它屬於同一索引節點的所有老節點擁有更高的版本號。除了普通索引節點的仲介資料，每個節點還包含所屬索引節點的名字和父索引節點的索引節點號。此外，每個節點還可能包含一定長度的資料，當資料存在時，節點並記錄了資料在檔中的位置。

表二、JFFS2 管理區塊種類之列表 [25]

欄位名稱	管理之區塊種類
clean_list	存滿有效資料的區塊
dirty_list	存有無效資料的區塊
eraseable_list	存滿無效資料的區塊
erasing_list	區塊正進行清除工作
erase_pending_list	區塊目前是需被清除
erase_complete_list	區塊已清除完畢 但尚未標示可用
free_list	區塊已清除完畢 且已標示可用
bad_list	損壞區塊

在 JFFS2 中，儲存節點格式是更有彈性的。它予許定義新的節點型態。每個節點結構都會有相同的開頭，包含 Magic BitMask、節點型態、節點總大小及 CRC(cyclic redundancy checksum)，四個欄位，如圖二所示。



圖二、JFFS2 節點共同的開頭 [25]

表三、JFFS2 節點共同開頭欄位的資料型態與作用 [25]

名稱	資料型態	作用
Magic Bitmask	16 位元無號數	分辨檔案系統
Node Type	16 位元無號數	設定節點型態
Total Node Length	32 位元無號數	節點總大小
Node Header CRC	32 位元無號數	節點開頭 CRC

(一) JFFS2 的串列鏈結列表(Linked List)

JFFS2 根據快閃記憶體每個區塊的使用狀況，以不同的串列鏈結列表(Linked list)來分類管理。如未使用之區塊，以 *free_list* 這個列表管理。

列表間管理的區塊是互斥的，也就是一個區塊不可能由兩個列表同時管理。**錯誤! 找不到參照來源。**為 JFFS2 所有的列表及其管理之區塊的種類。其中，*clean_list*、*dirty_list* 及 *free_list* 是 JFFS2 管理機制中較重要的幾個列表，*clean_list* 與 *dirty_list* 是 JFFS2 回收區塊清單，而 *free_list* 則是空間配置時，可使用區塊之清單。這些列表將在 JFFS2 掛載時，經由掃描整個快閃記憶體，依照區塊的使用情況，分別歸入各所屬列表中。JFFS2 掃瞄演算法，如 JFFS2_Scannning ()所示。

```
Algorithm JFFS2_Scannning ()
 If (the block is Last_Block) then End;
   Get a block;
     If (it is erased and usable)
        Then Add it to free_list;
        Else if (there is invalid data in it)
            Then Add it to dirty_list;
          Else if (there is a fully valid data)
                Then Add it to Clean_list;
        End if;
     End if;
 End if;
End;
```

(二) JFFS2 回收機制

JFFS2 的操作行為是以順序配置方式，寫入不同類型的節點，當一個區塊寫滿時，由 *free_list* 中取出新區塊，從新的區塊的起始處繼續寫入。當 *free_list* 所擁有的區塊數量到達臨限時，回收機制便啟動，清除舊區塊、產生新區塊。JFFS2 回收機制使用隨機方法決定選擇那一個區塊進行清除。若一個隨機量除以 100 的餘數為非 0，由 *dirty_list* 中選擇一個區塊進行清除。反之，若餘數為 0，從 *clean_list* 中選擇一個區塊進行清除。此種回收機制，在 99%的情況下，會重新利用了那些含有被拋棄節點的區塊，提高了清除效率，減少了不必要的清除；在 1%的情況下，清除存滿有效節點的區塊，來保證資料在快閃記憶體上迴圈移動，從而達到均勻清除[25] [25] 。JFFS2 回收機制演算法，如 JFFS2_retrieve()所示。

```
Algorithm JFFS2_retrieve()
   Get a random_value;
      If (random_vlaue mod 100 ≠ 0)
              Get a block form dirty_list;
            Else Get a block from Clean_list;
      End if;
   Erase the candidate block of dirty_list or
   Clean_list;
END;
```

雖然，由 JFFS2 回收機制似乎是可以達到均勻清除均勻寫入的功能，且兼顧效率問題，以下針對 JFFS2 作分析探討。

(1) JFFS2 不能切確掌握每個區塊實際的清除次數：回收機制是以機率的方式選擇所要回收區塊，並非依據每區塊實際清數次數為考量。使用機率的方式有可能發生回收時所選擇的區塊，清除次數是較其他區塊多，使得此區塊(清除次數高的)將提前毀損，造成資料的遺失。

(2) 列表 *dirty_list* 未考量區塊使用率：
dirty_list 是存放含有無效空間的區塊，依照 JFFS2 的分類機制，只要區塊中含有部分無效空間，此區塊將被歸至 *dirty_list*。換言之，當回收時所選擇的區塊正好是這一類的區塊時，其清除效率將會大幅下降。因此，*dirty_list* 的區塊應依據其使用率作為考量，使用率低的區塊優先候選先回收。

(3) 並未針對冷資料熱資料作適當處理：
資料依照更新的情況可分為冷資料與熱資料，若某資料經常被更新，則稱為熱資料；反之，不常更新則為冷資料。JFFS2 並不知每區塊清除次數狀況，無論是寫入新資料、更新資料或者是回寫回收時所收集的有效資料，皆取 *free_list* 最先前的候選區塊使用，因此可能造成從 *free_list* 取得的區塊，其清除次數已經過高，而寫入的資料卻又是熱資料，進而加快此區塊的損毀。

(三) JFFS2 回收機制改善方法

基於 JFFS2 的回收機制，我們經由前述之理論基礎，提出了具體方法進行改善：

(1) 在每個區塊加入一個計數器，用以紀錄每個區塊清除的次數，如此在回收機制啟動時便可以依照每個區塊清除次數，選擇清除次數少的區塊進行回收，對各個區塊最貼切管理。

(2) 原 JFFS2 中亦紀錄每個區塊目前的使用狀況，包括目前儲存有效資料的空間、無效資料的空間及可用空間…等。我們可以利用這些資訊以選擇有效空間最少的區塊當作優先回收的目標，增加回收效率。

對於冷資料與熱資料，我們對於寫新資料或資料更新時，定義為熱資料屬性，而將回收時所收集的有效資料，定義為冷資料屬性。此外，亦將 *free_list* 的區塊候選加以排序，若儲存資料屬性為熱資料屬性，則分配清除次數最少的區塊予以使用，若儲存資料為冷資料屬性，則分配清除次數最大的區塊，以達到均勻使用之目的。

綜合以上(1)、(2)方法，我們透過清除指標方式，依照區塊使用率及清除率考量，提供最佳候選。而此指標可以動態調整方式，提供在不同情況下最佳化之處理，其 *CleaningIndex* 式如下[19] 。

$$CleaningIndex = (1-l)u_i + l\left(\frac{\in_i - \in_{\min}}{\in_{\max} - \in_{\min} + 1}\right)$$

u_i 表示區塊 i 的使用率；i 表第幾區塊；
$\hat{I}_i$ 表示區塊 i 清除計數；i 表第幾區塊；

$\hat{I}_{min}$ 表示所有區塊最小的清除計數值；
$\hat{I}_{max}$ 表示所有區塊最大的清除計數值；
λ 表示正規化齊平度。

由 *CleaningIndex* 式中，可發現若 λ 值越大，則清除次數小的區塊將優先作回收候選，所以較注重清除次數；反之，λ 值越小，則使用率低的區塊將會被優先成為回收候選，所以注重回收清除效率。因此，可以依據系統現況需求，λ 值進行調整。

(四) NFFS$_{Nand}$ 架構

NFFS$_{Nand}$ 是本文以 JFFS2 作為研究基礎，所提出的 NandFlash 高可靠管理架構。在實作驗證上，我們加入變數 *ecounts*、*e_max* 及 *e_min*，分別用以計數每個區塊清除的次數、所有區塊最大清除次數及最小清除次數。此外，我們定義一個新的節點 *jffs2_ecount_node*，將清除次數儲存至每個區塊中。這些變數及結構資訊，提供我們掌握目前快閃記憶體每區塊的清除情況，如表四所示。

原 JFFS2 在選擇所要回收的區塊時，其來源為 *dirty_list* 及 *clean_list*；而我們的機制是選取 *CleaningIndex* 最小的區塊來進行回收，因此只需要一個列表管理已使用過的區塊，並按照 *CleaningIndex* 加以排序。我們保留 *dirty_list*，捨棄 *clean_list*。為了方便排序，我們增加一輔助列表 *tmp_dirty_list*，用以存放將要加入 *dirty_list* 的區塊。下面將說明上述這些變數、列表及結構如何運用。

(1) *ecounts* 及 *jffs2_ecount_node*：*ecounts* 用以計數每個區塊清除次數，*jffs2 _ecount_node* 則是將每個區塊上的清除次數儲存至區塊中，*ecounts* 加 1 及 *jffs2_ecount_node* 寫入的時機，我們選在 JFFS2 的函數 *jffs2_mark_erased_ blocks* 中完成。此函數原目的是 JFFS2 在完成某區塊的清除工作後，會先對此區塊進行檢查，包括區塊是否完全清除及區塊是否有損壞...等，檢查後便利用此函數在此區塊的起始點，寫入一個標記以表示此區塊是清除完畢且確定可使用。故我們選擇在此函數中進行清除次數累加，並且與此標記一同寫入區塊中。

(2) *e_max* 及 *e_min*：JFFS2 當掛載(mount)後，首先將掃描整個快閃記憶體，以便掌握快閃記憶體的現況，如快閃記憶體的容量、共有幾個區塊、那些區塊是損壞(Bad)及目前各區塊的使用狀況…等。待完成掃描快閃記憶體，我們便可得知各個區塊的清除次數，進一步取得清除次數之最大與最小值。另外，在 *ecounts* 加 1 後，也需檢查最大最小值是否有改變。

NFFS$_{Nand}$ 掃瞄演算法，NFFS$_{Nand}$ _Scannn ing()如下所示:

```
Algorithm NFFSNand _Scannning()
 If (it is a Last_Block) then End;
      Get a block;
      If (it is erased and usable) then
           Add it to free_list and Sort the blocks by
           the ecounts;
      Else
           Compute CleaningIndex and Sort the
           blocks by this CleaningIndex;
           Add them to gc_list;
      End if;
   End if;
END;
```

(3) *gc_list*：原 JFFS2 在掃描快閃記憶體時，每掃描完一個區塊，會依照這個區塊的使用狀況，分別歸納入 dirty_list、clean_list、erase_pending_list 及 free_list 等列表。在我們的機制下，原先要放入 dirty_list 及 clean_list 的區塊，全改放入 gc_list，而要放入 gc_list 之前，皆須先經過函數 jffs2_cleaning_index 的計算後，且由排序函數 jffs2_ insert_ecount_ref 加以排序，再將此區塊加入 gc_list；另外，當 JFFS2 正在使用的區塊無可用空間或可用空間不足時，此區塊將被歸納至 gc_list，當然也必須加入 jffs2_cleaning_index 及 jffs2_insert_ecount_ref 兩函數計算及排序。

NFFS$_{Nand}$ 回收演算法，NFFS$_{Nand}$ _Scannning()如下所示:

```
Algorithm NFFSNand _retrieve ()
  IF (there is a block in tmp_dirty_list) then
     Get all blocks from tmp_dirty_list;
     Store them to gc_list and then Sort them by
  CleaningIndex;
  End if;
  Get a block from gc_list;
  Erase the candidate blocks of gc_list;
END;
```

(4) λ 值及 *tmp_gc_list*：λ 值可用來調整回收機制啟動時，首要注重回收效率或者是清除次數是否均衡。在我們的機制中，以最大及最小的清除次數相差不超過某最大值(自訂)時，就需重視回收效率，因此 λ 值設定較小的值；反之，若最大及最小的清除次數相差超過某最大值時，則注重清除次數是否均衡，直到最大及最小的清除次數相差小於某數值(比某最大值小)為止，才將 λ 值調為注重清除效率。如此當 λ 值做了改變後，我們必須把 gc_list 所有的區塊放入 tmp_gc_list 然後區塊再重新排序，提供最佳候選。調整 λ 之演算法，如 Adjust_λ ()所示。

```
Algorithm Adjust_λ ()
  If (Δ Î > threshold)
           Emphasize the cycle-leveling;
     Then if (Δ Î < threshold)
           Emphasize the erasure efficiency;
     End if;
  Moved all blocks from gc_list to tmp_dirty_list;
  End if;
END;
```

表四、$NFFS_{Nand}$ 新增之變數、結構及函數

名 稱	資料型態	功 能
ecounts	32 位元無號數的指標陣列	計數每個區塊的清除次數
e_max	32 位元無號數	記錄目前最大的清除次數
e_min	32 位元無號數	記錄目前最小的清除次數
gc_list	列表	將原本放在 dirty_list 及 clean_list 的區塊放入此列表
jffs2_ecount_node	結構	新定義的結構，將每個區塊的清除次數存入快閃記憶體中
p_ecount	32 位元無號數指標	紀錄此區塊目前的清除次數
jffs2_insert_sort_ref	無回傳值函數	某區塊要加入某列表時，依照某數值的大小排序
jffs2_cleaning_index	無回傳值函數	計算抹區塊的 Cleaning Index
jffs2_insert_ecount_ref	無回傳值函數	某區塊要加入 free_list 時，依照此區塊的清除次數大小來排序。

(5) *free_list*：JFFS2 在掃描快閃記憶體時，一旦發現某區塊上有表示此區塊是清除完畢且確定可用的標記，就會停止對此區塊掃描，並將此區塊加入 *free_ list*，於此，我們也必須進行排序的工作。另外在上面提到的函數 *jffs2_mark_erased_ blocks* 中，也需針對剛完成清除的區塊加以排序。

四、 主要研究結果驗證

本文中，使用 Samsung 公司的開發平台 SMDK24 10X，作為測試平台，如圖三所示。採用此平台乃是由於此平台核心為 ARM，可以使用 Linux 作業系統，並且含有 Smartmedia card 的控制模組，因此可支持本文驗證實驗。

圖三、SMDK2410X

本實驗所使用的測試資料來源為一些檔案大小不同的檔案，如表五所示。

表五、測試資料清單

項次	檔名	大小	項次	檔名	大小
1	Pattern-1	9.8KB	**10**	Pattern-10	97.7KB
2	Pattern-2	19.5KB	**11**	Pattern-11	293.0KB
3	Pattern-3	29.3KB	**12**	Pattern-12	585.9KB
4	Pattern-4	39.1KB	**13**	Pattern-13	683.6KB
5	Pattern-5	48.8KB	**14**	Pattern-14	781.2KB
6	Pattern-6	58.6KB	**15**	Pattern-15	878.9KB
7	Pattern-7	68.4KB	**16**	Pattern-16	976.6KB
8	Pattern-8	78.1KB	**17**	Pattern-17	1.9MB
9	Pattern-9	87.9KB	**18**	Pattern-18	2.9M B

(一) 測試程序

(1) 開啟實驗平台，進入開機程式的命令模式，鍵入燒錄 Linux 映像檔指令，將編譯好的 Linux 核心燒錄至 Smartmedia Card。
(2) 將實驗平台重新開機後，進入 Linux 環境。
(3) 設定實驗平台的網路 IP 位址，將主機測試資料夾掛載(Mount)。
(4) 將 JFFS2 掛載。
(5) 進入主機掛載的測試用資料夾。
(6) 執行 shell script first.sh，開始拷貝資料。拷貝完畢後，重新開機。
(7) 重複上述(2)~(5)步驟，執行 shell script second.sh，開始拷貝資料。拷貝完成後，開發板重新開機。
(8) 重複步驟(7)，但每次要執行的 shell script 依序為 second.sh、third.sh、fourth.sh、fourth.sh、third.sh、third.sh 及 second.sh，重新開機，結束測試。

(二) 結果與分析

實驗中，我們使用 Smartmedia card 其中 32 個區塊作為測試依據，由圖 4 之數據可以發現 JFFS2 與 $NFFS_{Nand}$ 清除程度之測試結果。觀察得知 JFFS2 每個區塊的清除次數大多比 $NFFS_{Nand}$ 高；此外，JFFS2 各區塊的清除次數呈現巨幅震盪，區塊間的清除次數的曲線起伏很大；相較之下，$NFFS_{Nand}$ 的各區塊清除次數曲線則趨於平緩，意味著每區塊均勻使用並且可降低系統花費提高使用性。圖 5 是 JFFS2 與 $NFFS_{Nand}$ 的總清除次數比較圖示，而第一次、第二次…第八次是指每執行完一次 shell script 檔重新開機，掛載 JFFS2 所得到的總清除次數。觀察可得知，$NFFS_{Nand}$ 比 JFFS2 總清除次數明顯減少，如本實驗第八次寫入檔案結束後，總清除次數已相差 2500 次左右。因此，$NFFS_{Nand}$

不但可以避免某些區塊先行損毀，亦可使所有區塊的清除次數大幅降低，延長了快閃記憶體的使用壽命。此外，由於清除操作時間需長時間進行(2ms/Block)，所以我們將清除次數大幅降低，也意味著執行時間可增速，無須執行不必要的清除行為，浪費系統時間。

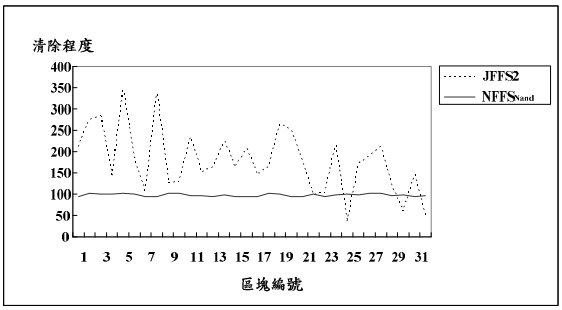


圖 4. JFF2 與 $NFFS_{Nand}$ 清除程度之比較

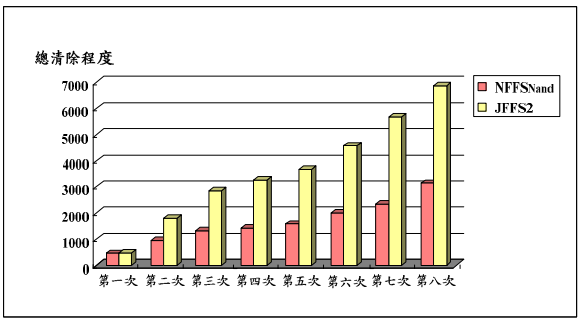


圖 5、JFFS2 與 NJFFS2 總清除次數比

JFFS2 清除一個區塊前，必須先將此區塊的有效節點收集，寫至一個新的區塊中，才能進行清除的工作。因此，Smartmedia Card 清除一個區塊需花費相當久處理時間。因此，我們測試每執行一個 shell script 檔，到執行結束所測得的時間。觀察得知，JFFS2 幾乎每次執行一個 shell script 檔所花費的時間比 $NFFS_{Nand}$ 耗時，主要原因是因為原 JFFS2 的 dirty_list 並未做適當的管理，導致每次空間不足要回收區塊時，所選擇的區塊擁有多的有效資料，這些有效資料又必須找一個空間放置，因此 JFFS2 又必須回收其他區塊(因為原本就空間不足)，萬一回收的區塊屬於擁有相當多的有效資料時，便造成需要花相當多的時間作回收處理。相較之下，$NFFS_{Nand}$ 因為 gc_list 有適當的做排序，在注重清除效率時，使用率最小的區塊會最先被選擇去回收，因此回收的資料少，相對的處理時間變得更為迅速，因此時間處理效能上，$NFFS_{Nand}$ 優於 JFFS2，表六為 JFFS2 與 $NFFS_{Nand}$ 的執行時間比較表。

實驗中 SmartMedia Card 一個區塊是 16K Bytes 大小，如依照實驗的數據而言，原 JFFS2 完成我們所指定的存取動作後，一共使用了 6880×16K Bytes = 110.08M Bytes；$NFFS_{Nand}$ 則使用了 3141×16K Bytes = 50.256M Bytes；以存取同樣的資料來看，JFFS2 比 NJFFS2 多使用了 59.824M Bytes；換言之，JFFS2 比 $NFFS_{Nand}$ 浪費了 59.824M Bytes 的使用空間。因此，若儲存的資料越大時，$NFFS_{Nand}$ 使用效率亦能更為彰顯。

表六、 JFFS2 與 $NFFS_{Nand}$ 執行時間比

Order	shell script	JFFS2	$NFFS_{Nand}$
1	File1.sh	00:14:52	00:20:32
2	File2.sh	01:35:50	00:29:48
3	File3.sh	01:18:25	00:28:55
4	File4.sh	00:34:28	00:11:03
5	File4.sh	00:20:33	00:15:15
6	File3.sh	01:11:25	00:41:34
7	File3.sh	00:50:14	00:29:30
8	File2.sh	01:50:52	01:13:58
Total		07:56:39	04:10:35

五、 結 論

在一般的嵌入式系統中，使用快閃記憶體當做大量儲存體是普遍的趨勢，其中 NandFlash 管理機制不像硬碟管理方式的簡易，主要是因 NandFlash 本身不能覆寫原儲存資料之限制，另外在讀取、寫入及清除時有一個固定基本單位的要求，使得整個 NandFlash 使用時，所需考量點甚多。而嵌入式系統中所採用的作業系統大多採用 Linux Kernel。而在 Linux 的檔案管理系統中，針對快閃記憶體所設計的管理系統，稱之為 JFFS，至目前為止此系統已更新至第二版(JFFS2)。而此 JFFS2 中，在管理機制上而言，並非相當完善。

因此，本文針對 JFFS2 不足之處，我們以區塊作為 NandFlash 管理單元提出改良式架構，提出合理的理論及模擬數據，並以實作的方式加以驗證。在清除策略中以 KimLee 演算法為基礎，我們改善 *CleaningIndex* 提供妥善清除索引指標，選擇合乎回收效益的區塊。在資料配置方面，以一合理的配置指標，將冷、熱資料分別置於不同屬性的區塊中，改善冷熱資料再度混合的機率，以達到降低清除次數的功效，進而可提高系統壽命，與清除策略互相輝映，以達到區塊均勻使用之目的。

致 謝

本論文研究承蒙國家科學委員會 Linux 推動小組學門贊助“高可靠的 JFFS”，計畫編:NSC 92-2218-E-027-018。

參考文獻

[1] R. Bez; E. Camerlenghi; A. Modelli, A. Visconti, "Introduction to flash memory," *Proc. of the IEEE* , Volume: 91 , Issue: 4 , pp.489-502, Apr. 2003.

[2] C. C. Cheng, W. T. Huang, C. T. Chen, C. H. Chen, “The Design and Implementation of Flash File System Management,” *Jour. of NTUT*, Vol. 36-2, pp. 43 - 62, Mar. 2003.

[3] M. L. Chiang, *A Study on the Design of Memory Based Embedded Storage Systems*, Ph.D. Thesis, Department of Computer and Information Science, National Chiao Tung University, Jan. 1999.

[4] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, "Using data clustering to improve cleaning performance for flash memory," *Software Practice and Experience*, Vol.29, No.3, pp.267–290, 1999.

[5] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, "Managing Flash Memory in Personal Communication Devices," *Proc. of the International Symposium on Consumer Electronics*, 1997.

[6] B. Dipert and M. Levy, *A Flash memory file systems - in Designing with FLASH MEMORY*, San Diego: Annabooks, pp.227–271, 1994.

[7] F. Douglis, R. C’eres, F. Kaashoek, K. Li, B. Marsh, and J.A. Tauber, "Storage alternatives for mobile computers," *Proc. OSDI94*, pp.25–37, 1994.

[8] R. Dan and J. Williams, "A TrueFFS and Flite Technical Overview of M-Systems Flash File Systems", 80-SR-002-00-6L Rev. 1.30. http://www.m-sys.com/tech1.htm, March 1997.

[9] *Flash Memory Databook*, Intel,1995.

[10] W. T. Huang, Y. S. Chen, C. T. Chen, C. C. Cheng, “The Flash Memory Management Model of the Higher Reliability and Lower Cost-Benefit,” *Jour. of NTUT*, Vol. 37-1, pp 61-75, Oct. 2003.

[11] H. S. Huang, J. H. Huang, and P. Y. Chang, “Performance Improvement of Embedded Flash Memory,” *Jour. of National Taipei University of Technology*, Vol. 35-1, pp. 9-21, Mar. 2002.

[12] H. S. Huang, J. H. Huang, and P. Y. Chuang, "Performance Improvement of Embedded Flash Memory," *J. of National Taipei University of Technology*, Vol. 35-1, pp.9-21, Mar. 2002.

[13] Intel, What is Flash Memory ? http://www.intel.com/design/flash/articles/what.htm

[14] H. J. Kim and S. G. Lee, "An Effective Flash Memory Manager for Reliable Flash Memory Space Management," *IEICE Trans. Information & System*, Vol. E85-D, No. 6, pp. 951-964, June 2002.

[15] H. J. Kim, *and* S. G. Lee, "A New Flash Memory Management for Flash Storage System," *IEEE COMPSAC Computer Software and Applications Conference*, pp.284-289, 1999.

[16] Kawaguchi, S. Nishioka, and H. Motoda, "Flash memory based file system," *Proc. USENIX95*, pp.155–164, 1995.

[17] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, and T. E. Anderson, "Improving the performance of log-structured file systems with adaptive methods," *Proc. Sixteenth ACM Symposium on Operating Systems Principles*, pp.238–251, 1997.

[18] J. T. Robinson, "Analysis of steady-state segment storage utilization in a log-structured file system with least-utilized segment cleaning," *OS Review*, Vol.30, No.4, pp.29–32, 1996.

[19] Samsung Corp., "K9W8G08U1M Flash Memory," Data Sheet, 2003. http://www.samsung.com

[20] Samsung Corp.,"K9F1208U0M-YCB0, K9F1208U0M-YIB0,K9F5608U0M-YCB0 Flash Memory, " Data Sheet, 2001. http://www.samsung.com

[21] Samsung Corp., Flash/SmartMedia/ File System, Memory Data Sheet, 2000.

[22] D. See, C. Thurlo, "Managing data in an embedded system utilizing flash memory," Intel Technical Note, 1995.

http://www.intel.com/design/.comp/papers/esc .sh.htm

[23] C. H. Wu, L. P. Chang, and T. W. Kuo, "An Efficient B-Tree Layer for Flash Memory Storage Systems," *The 9th International Conference on Real-Time Computing Systems and Applications (RTCSA 2003)*, Taiwan, 2003.

[24] M. Wu and W. Zwaenepoel, "eNVy: A non-volatile main memory storage system," *Proc. of the 6th Symposium on Architectural Support for Programming Languages and Operating Systems*, pp.86–97.

[25] David Woodhouse, "JFFS The Journaling Flash File System," *Ottawa Linux Symposium*, 2001.