# OSC Final Report

310605004 李頤

[github]
https://github.com/Sciencethebird/osc2022/tree/310605004/Final_Project/ssd_fuse_lab

1) FUSE原理與概述

FUSE 主要是提供一個簡單的API實現與現有檔案系統的掛接，好處是讓user不需要在複雜且容易出現錯誤的 kernel space 重新自行實現檔案系統。使用的方法如下：

- 不管是自定義的檔案或裝置，只要user定義好檔案系統的基本API，就可以與FUSE進行對接，以此lab的code為例，下圖為FUSE所需的函式：

```
static const struct fuse_operations ssd_oper =
{
    .getattr        = ssd_getattr,
    .readdir        = ssd_readdir,
    .truncate       = ssd_truncate,
    .open           = ssd_open,
    .read           = ssd_read,
    .write          = ssd_write,
    .ioctl          = ssd_ioctl,
};
```

- API設定完成之後，只需要呼叫 fuse_main()即可完成與FUSE的對接。

```
fuse_main(argc, argv, &ssd_oper, NULL);
```

- 對接完成後，使用者的裝置或檔案系統就可以在原本的作業系統上正常運作，並可使用一般檔案操作以及 open() wirte() 等等標準函式進行讀寫。

2) 實作方法

**ftl_read():**

需要先將透過 API 傳近來的 offset 與 size 轉換成 lba，在透過 L2P table查找目前此 logical page 所對應的 physical page，之後對該page進行 nand_read 即可。

```
static int ftl_read( char* buf, size_t lba)
{
    // TODO
    // lba here means i-th logical page
    unsigned int pca = L2P[lba];
    if(pca != INVALID_PCA)
    {
        nand_read(buf, pca);
    }
    else
    {
        fprintf(debug,"[ftl_read] invalid PCA, ignoring nand_read\n");
    }
}
```

**ftl_write():**

主要的運作流程為:

1. 使用 get_next_pca() 來取得新的 physical page
2. 將資料寫入其中並更新L2P table。

這邊需要處理兩個額外功能:

1. 當寫入起始的位置並沒有align 512的時候, 需要先將原本的 page 讀出來, 透過指標的操作將新資料寫入原來 page 的指定資料區段, 在將更動後的 page 資料寫入透過 get_next_pca()取得的新page。

```
tmp_lba = offset / 512; // index of your logical block fro
tmp_offset = offset % 512; // offset in the page
tmp_lba_range = (offset + size - 1) / 512 - (tmp_lba) + 1;
tmp_buf = calloc(tmp_lba_range * 512, sizeof(char));

//fprintf(debug,"\n\n[offset ]: %d\n", offset);
//fprintf(debug,"[tmp_lba_range]: %d\n", tmp_lba_range);
//fprintf(debug,"[tmp_lba]: %d\n\n", tmp_lba);

for (idx = 0; idx < tmp_lba_range; idx++)
{
    // TODO
    ftl_read(tmp_buf+(idx*512), tmp_lba+idx);
}       You, a week ago • [SSD] add source code

memcpy(tmp_buf+tmp_offset, buf, size);

ftl_write(tmp_buf, tmp_lba_range, tmp_lba);
return size;
```

2. 複寫的時候, 也就是同一個 logical address 再次被寫到時, 需要將原本對應的的 physical page 標記成 stale, 以便之後 GC 時回收。

```
if(L2P[lba+idx] != INVALID_PCA)
{
    // mark overwritten P2L as stale for later GC
    fprintf(debug,"[ftl_write] overwite to existing logical address (lba)...\n");

    PCA_RULE stale_pca;
    stale_pca.pca = L2P[lba+idx];

    fprintf(debug,"[ftl_write] marking stale at stale_pca: [%d, %d]\n",
            stale_pca.fields.nand, stale_pca.fields.lba);

    P2L[stale_pca.fields.nand*10 + stale_pca.fields.lba] = STALE_LBA;       You,
}
```

**gc():**

當free_block_number <= 0得時候, 找出最少 valid page 的 dirty block, 把這些 valid page 搬移至其他的 free pages, 就可以使用 nand_erase()清出一個完整的 block, 完成 garbage collection 的動作。

3) 修改了哪些地方

降低 WA的的主要有兩個方向，一為降低GC的頻率，二為降低每次GC的讀寫次數，我實做的方法主要是降低每次GC的讀寫次數，所以在GC開始的時候，我會去掃描每個block，確認每個 block 的 stale page 數量，找出 stale page最多的block去做GC。這樣就可以搬動最少的 valid page 的情況下清理出一個 free block，同時也可以避免在GC的過程去回收全部都是valid page的block，造成無意義的搬動。

第二個比較明顯的提昇點為進行 GC 的時機點，原本認為在剩下一個block的時候進行 GC 可以達到最好的效果，但是根據 get_next_pca() 的運作規則，當free_block_number 還剩下一個的時候，curr_pca 其實才走到剩下唯二 block 的第一個 page，所以等是在還有19個page的情況下就做了GC。我把做 GC的時間點下修成剩下0個free block的時候才做 GC，這樣一來就是剩下9 個 free pages得時候做GC，可以明顯降低讀寫的頻率。雖然沒有完整的 10 個 free page，但是因為最多會有 99 個 valid lba，也就是至少會有 21 個 stale分佈於其他12 個 block，所以計算下來其餘的任一 block 最少會有 2 個 stale pages，所以不用擔心剩餘只有9個page會不夠放置的問題。

最後一個提昇點是我又進一步限制開始GC的時機，因為最少 valid page 的 block 並不是每次都為 9，大多數可能只有 1-2 個 valid pages，如果沒有特別限制就相當是提早了GC的時間點，也就增加GC的頻率。所以我的作法是，當剩餘的 free pages 恰巧足夠容納最少 valid pages 的 dirty blocks 時，才會進行 GC的搬移。這樣可以確保最後一刻才進行GC。增加此條件後，可以再降低大約 0.2 的 WA 值。

4) 結果分析

在同一筆測試資料下

```
"test3")
    # multiple overwrite test
    cat /dev/urandom | tr -dc '[:alpha:][:digit:]' | head -c 51200 | tee ${SSD_FILE} > ${GOLDEN} 2> /dev/null
    cat /dev/urandom | tr -dc '[:alpha:][:digit:]' | head -c 11264 > ${TEMP}
    for i in $(seq 0 1000)
    do
        dd if=${TEMP} skip=1024 of=${GOLDEN} iflag=skip_bytes oflag=seek_bytes seek=6789 bs=5000 count=1 conv=notrunc 2> /dev/null
        dd if=${TEMP} skip=1024 of=${SSD_FILE} iflag=skip_bytes oflag=seek_bytes seek=6789 bs=5000 count=1 conv=notrunc 2> /dev/null
        dd if=${TEMP} skip=2024 of=${GOLDEN} iflag=skip_bytes oflag=seek_bytes seek=123 bs=777 count=1 conv=notrunc 2> /dev/null
        dd if=${TEMP} skip=2024 of=${SSD_FILE} iflag=skip_bytes oflag=seek_bytes seek=123 bs=777 count=1 conv=notrunc 2> /dev/null
    done
```

上：未加上上述條件
下：加上上述優化條件

```
~/De/osc2022/Final_Project/ssd_fuse_lab
> sh ./test.sh test3
success!
WA:
1.323272

~/De/osc2022/Final_Project/ssd_fuse_lab
> sh ./test.sh test3
success!
WA:
1.150820
```