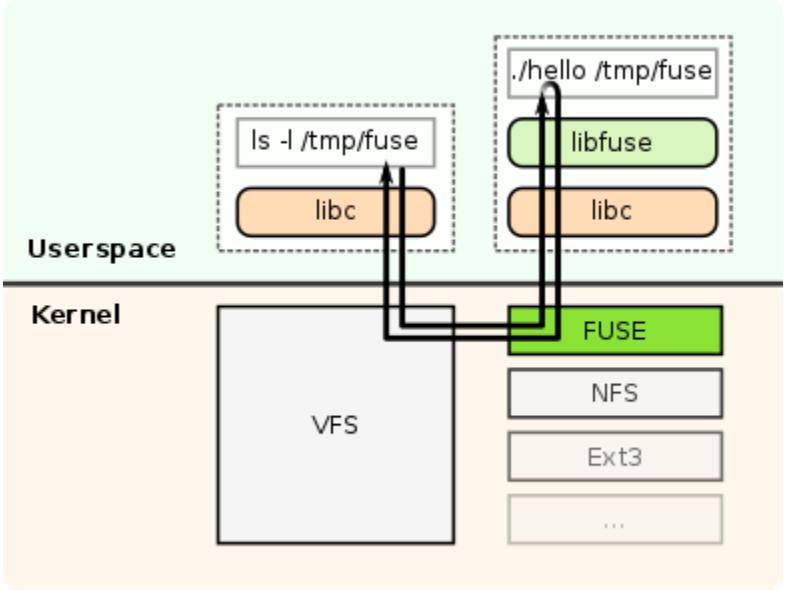
1. FUSE 的目的，原理。
2. 修改了哪些地方
3. 實作的方法
4. 結果分析

1. FUSE 的目的，原理

FUSE，Filesystem in USErspace，這個開源的 library 能夠讓我們能在 user level 的環境下架構自己的檔案系統，由下圖可以看到我們可以藉由 kernel 中的 FUSE，來在/tmp/的地方掛載我們的檔案並操作，而在這個 final 中，簡單的 SSD 操作便是在 FUSE 上面所建立的，因為我們想要模擬寫入 block 及 page，FUSE 可以來幫我們完成這件事。



2. 修改了哪些地方

因為傳統機械硬碟與固態硬碟的背後原理不同，因此在操作上的邏輯也會不同，但是為了讓 OS 端的介面可以用最低的成本來做機械硬碟到固態硬碟的轉換，因此 SSD 的 driver 上出現了 FTL layer，來協助處理相關的問題。再來因為 NAND 的物理特性，SSD 每次寫入的單位為 Page，每次擦除的單位為 Block，且一旦 page 被寫入後，便無法對該 page 做修改，必須重新寫入至新的 page 中。但 page 總會有被寫滿的時候，這時候我們就需要 garbage collection 來，回收有用的資料，清空原有的 block，這些便是 SSD 中的 FTL 主要在做的事情。

而我們主要就是要在原文件中，修改 TODO 的地方，基本上就是要套一層 FTL 要做的事情，例如 FTL_read、FTL_write、Garbage collection、以及自己所需的資料結構。

3.實作的方法

實作上，Garbage collection 會是我們的主要重點，以下我會分段的講解我的實作方式。

首先，我會走訪每一個 Block，找出 stale 數量最多的 block，並對他選為 erase 的對象。

```
for(int i = 0; i < PHYSICAL_NAND_NUM; i++){
        if(check_op(i))continue;
        if(max <= stale_count[i]){
                max = stale_count[i];
                victim_block_idx = i;
        }
}
```

接著，走訪該 block 中的每一個 page，檢查是否該 page 的資料仍為使用中的，如果是的話，便要找到新的 page 來抄寫過去。最後，在維護我們 L2P、P2L 的兩張 table 即可。

```
// traversal the victim block, and find the using page that need to be copied.
for(int i = 0; i < PAGE_PER_BLOCK; i++){
        victim_pca.fields.lba = i;
        victim_pca.fields.nand = victim_block_idx;
        if(page_state[victim_block_idx * PAGE_PER_BLOCK + i] == use){ //use, need to copy
                // find the vaild page, and copy the using page to it.
                new_free_page = find_valid_page(); // get free page
                memset(buf,0,512); //init buf

                nand_read(buf, victim_pca.pca); // read vaild page on victim block
                nand_write(buf, new_free_page.pca); // write vaild page to new free page
                page_state[get_pca_address(new_free_page)] = use;

                P2L[get_pca_address(new_free_page)] = P2L[get_pca_address(victim_pca)];
                L2P[P2L[get_pca_address(victim_pca)]] = new_free_page.pca;
                P2L[get_pca_address(victim_pca)] = INVALID_LBA;
        }else{
                P2L[get_pca_address(victim_pca)] = INVALID_LBA;
        }
}
```

4. 結果分析

我在我的 final 中，Physical block 數量為 13，Logical block 數量為 10。意思是我們會有 3 個多餘的 block 來做操控的空間。Over-provisioning (op)就是我們 SSD driver 自己保留，來做操控的，不同的 op 數量會對整體的表現。而我有測試不同數量的，分別是 3、2、1。而以下是測試的結果。

```
tzuheng@tzuheng-VirtualBox:~/Desktop/OSDI/ssd_fuse_lab$ ./test_case_x86_64
AC
write amplification: 28.764817
```

```
tzuheng@tzuheng-VirtualBox:~/Desktop/OSDI/ssd_fuse_lab$ ./test_case_x86_64
AC
write amplification: 12.330449
```

```
tzuheng@tzuheng-VirtualBox:~/Desktop/OSDI/ssd_fuse_lab$ ./test_case_x86_64
AC
write amplification: 8.025345
```

這樣的實驗結果表明，如果我們 Physical block 數量大於 Logical block 數量，我們這樣就可以減少作 garbage collection 的次數，放低 WA 的結果。

此外，如果要進一步降低 WA，我的猜想是，我們每次分配 page 的時候，應該要隨機的、平均的，分配在不同的 block 中，而不是照原本文件中的序列給。