Lab: Simple SSD

310551091 林柏均

1. Source Code

https://github.com/b4435242/osdi-nycu2022/tree/main/final

2. Fuse

FUSE is a userspace filesystem framework. It consists of a kernel module (fuse.ko), a userspace library (libfuse.*) and a mount utility (fusermount). Fuse filesystem in which data and metadata are provided by an ordinary userspace process. The filesystem can be accessed normally through the kernel interface. A userspace filesystem mounted by a non-privileged (non-root) user. The filesystem daemon is running with the privileges of the mounting user. A connection between the filesystem daemon and the kernel. The connection exists until either the daemon dies, or the filesystem is umounted. Note that detaching (or lazy umounting) the filesystem does *not* break the connection, in this case it will exist until the last reference to the filesystem is released.

3. Implementation

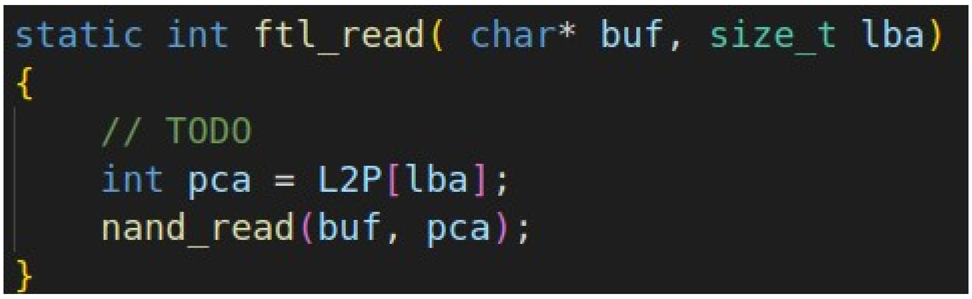
The control flow is like specify below:

a. ssd_read -> ssd_do_read -> ftl_read -> nand_read

b. ssd_write -> ssd_do_write -> ftl_write (garbage collection) -> nand_write

As the functions except ftl layer are implemented already, I will only explain the ftl functions and garbage collection.

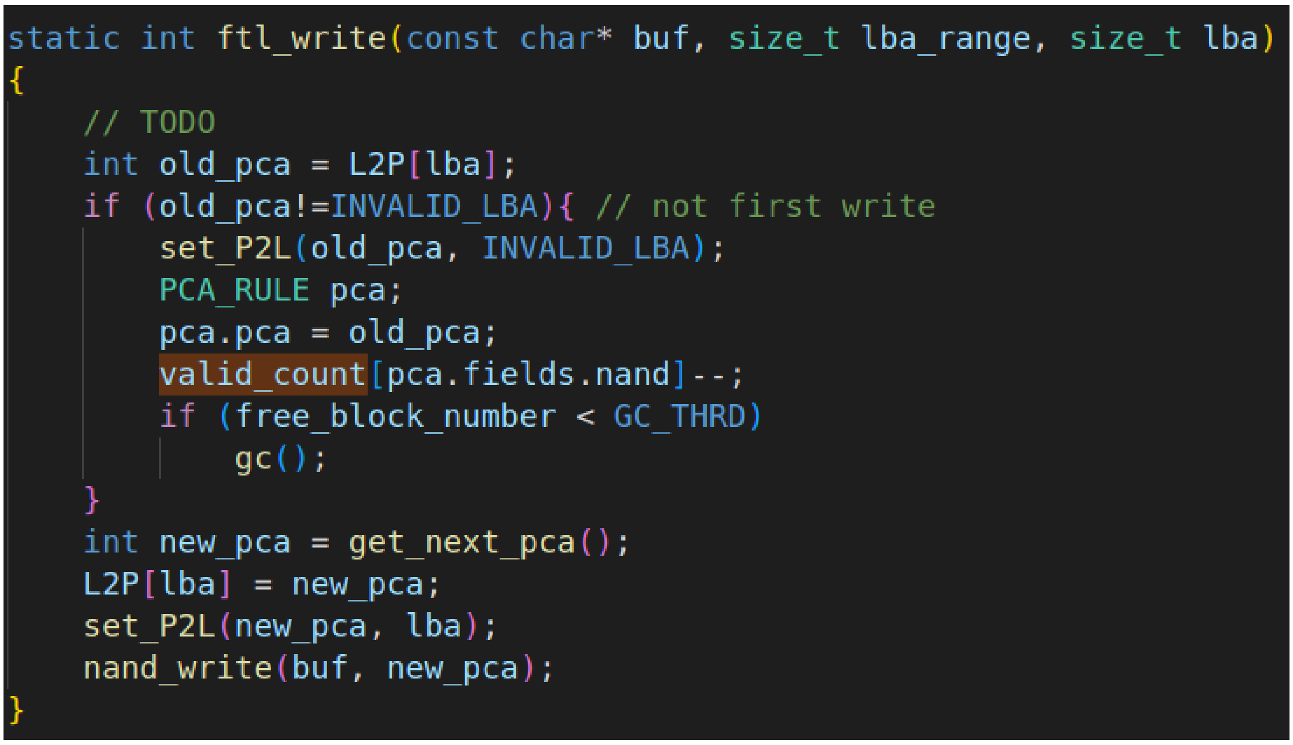
3.1 ftl_read



```
static int ftl_read( char* buf, size_t lba)
{
    // TODO
    int pca = L2P[lba];
    nand_read(buf, pca);
}
```

Basically, the function lookups to L2P table to get pca, and parse pca to nand layer to read data.

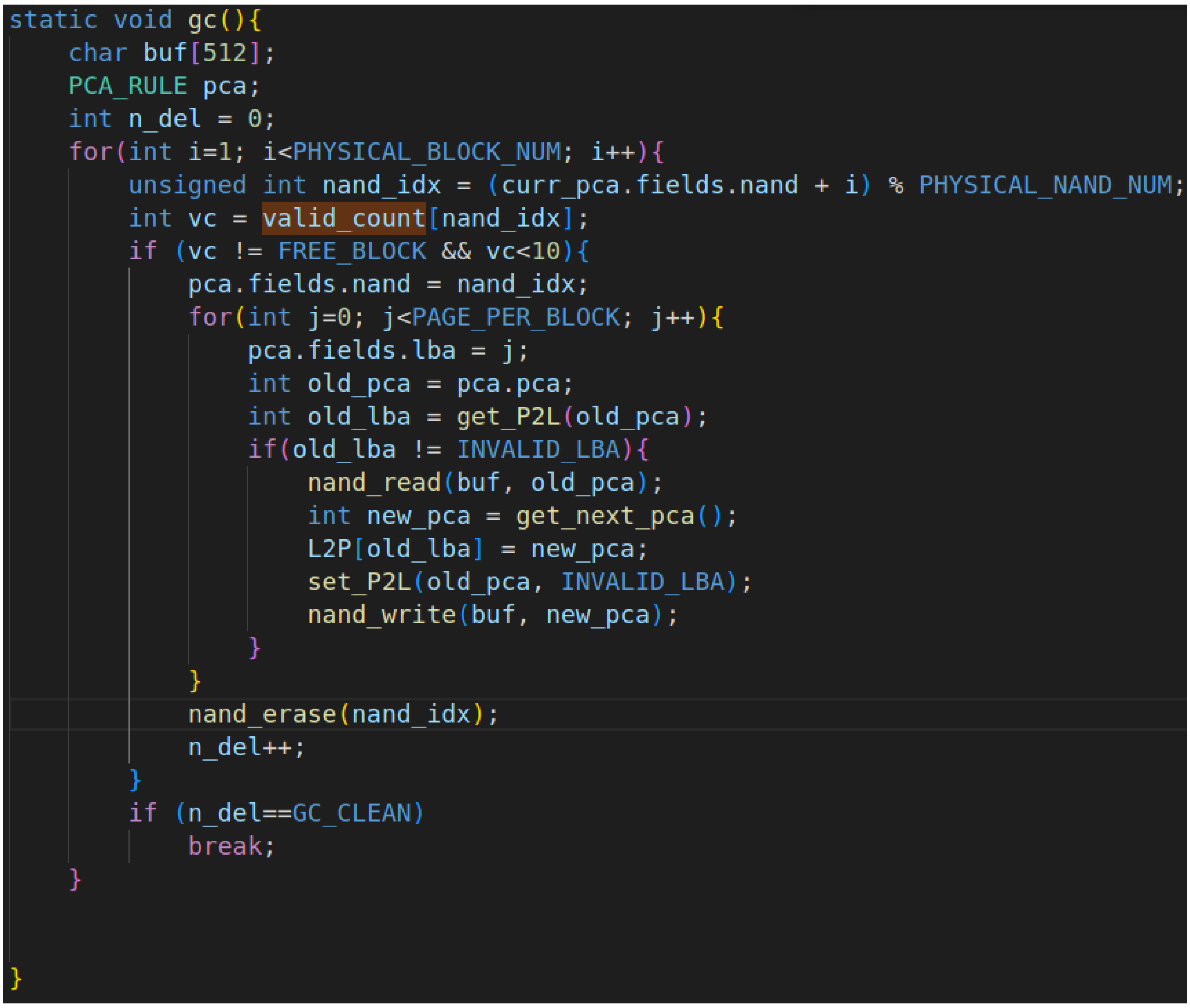
3.2 ftl_write



```c
static int ftl_write(const char* buf, size_t lba_range, size_t lba)
{
    // TODO
    int old_pca = L2P[lba];
    if (old_pca!=INVALID_LBA){ // not first write
        set_P2L(old_pca, INVALID_LBA);
        PCA_RULE pca;
        pca.pca = old_pca;
        valid_count[pca.fields.nand]--;
        if (free_block_number < GC_THRD)
            gc();
    }
    int new_pca = get_next_pca();
    L2P[lba] = new_pca;
    set_P2L(new_pca, lba);
    nand_write(buf, new_pca);
}
```

To elaborate the functionality of ftl_write, I simply divide the function to 2 parts. The first part is about marking the pca to stale in the case of data overwritten. While, the second part is to map lba to the pca that we just allocate. Furthermore, garbage collection is enabled when free block number is lower than the threshold defined manually.

3.3 garbage collection



```c
static void gc(){
    char buf[512];
    PCA_RULE pca;
    int n_del = 0;
    for(int i=1; i<PHYSICAL_BLOCK_NUM; i++){
        unsigned int nand_idx = (curr_pca.fields.nand + i) % PHYSICAL_NAND_NUM;
        int vc = valid_count[nand_idx];
        if (vc != FREE_BLOCK && vc<10){
            pca.fields.nand = nand_idx;
            for(int j=0; j<PAGE_PER_BLOCK; j++){
                pca.fields.lba = j;
                int old_pca = pca.pca;
                int old_lba = get_P2L(old_pca);
                if(old_lba != INVALID_LBA){
                    nand_read(buf, old_pca);
                    int new_pca = get_next_pca();
                    L2P[old_lba] = new_pca;
                    set_P2L(old_pca, INVALID_LBA);
                    nand_write(buf, new_pca);
                }
            }
            nand_erase(nand_idx);
            n_del++;
        }
        if (n_del==GC_CLEAN)
            break;
    }


}
```

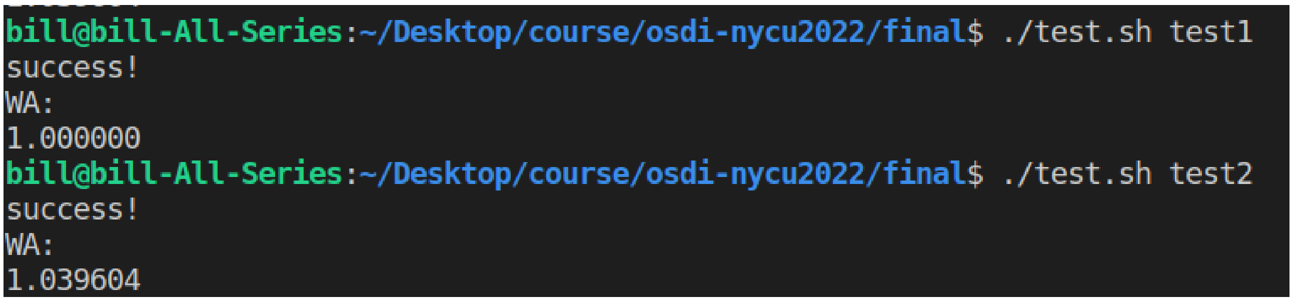
gc loops through all nands to choose the block that does not have full valid pages except free block to erase, and moves the valid page to the new allocated page. When gc erases enough number of nands defined manually or loops through all the nands, it stops and falls back to ftl_write.

4. Analysis

I set GC_threshold=3 at the end to satisfy the purposes of these two test scripts. As you can see below the experiment snapshot about how GC_threshold decides WA.

4.1 GC_threshold = 3

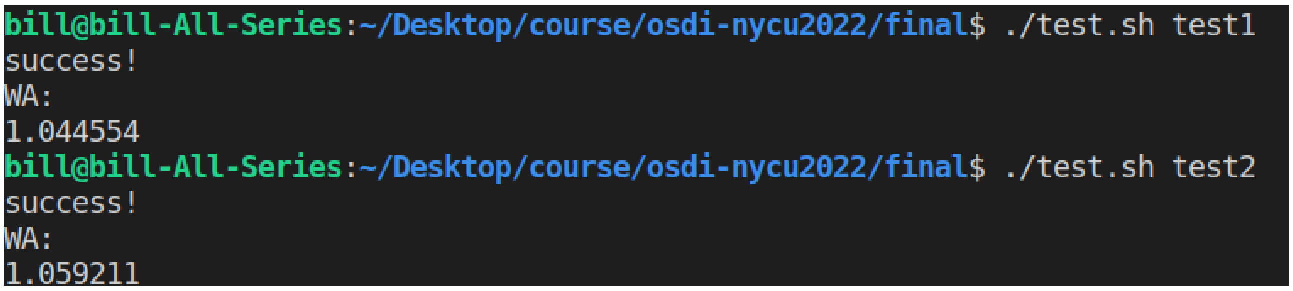
GC is successfully enabled in test2 and disable in test1, which is perfect in the way test script sees.



```
bill@bill-All-Series:~/Desktop/course/osdi-nycu2022/final$ ./test.sh test1
success!
WA:
1.000000
bill@bill-All-Series:~/Desktop/course/osdi-nycu2022/final$ ./test.sh test2
success!
WA:
1.039604
```

4.2 GC_threshold = 5

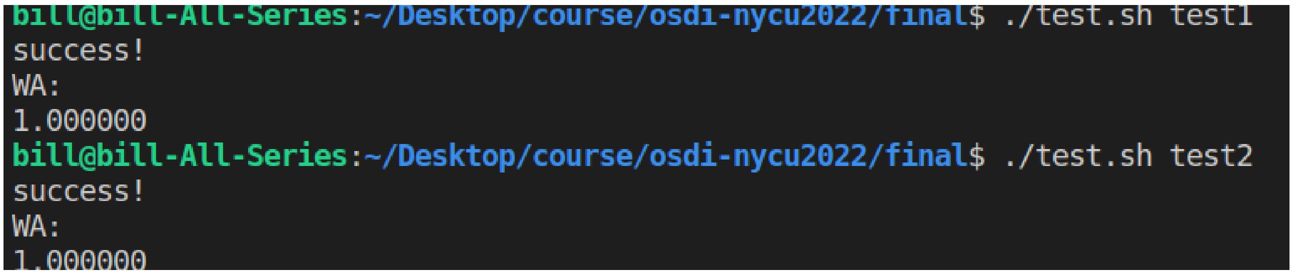
GC is both enabled in test2 and test1, which indicates threshold may be too high. Consequently, gc is enable too frequent, and WA gets bigger.



```
bill@bill-All-Series:~/Desktop/course/osdi-nycu2022/final$ ./test.sh test1
success!
WA:
1.044554
bill@bill-All-Series:~/Desktop/course/osdi-nycu2022/final$ ./test.sh test2
success!
WA:
1.059211
```

4.3 GC_threshold = 2

GC is both disabled in test2 and test1, which indicates threshold may be too low for these two test scenarios.



```
bill@bill-All-Series:~/Desktop/course/osdi-nycu2022/final$ ./test.sh test1
success!
WA:
1.000000
bill@bill-All-Series:~/Desktop/course/osdi-nycu2022/final$ ./test.sh test2
success!
WA:
1.000000
```