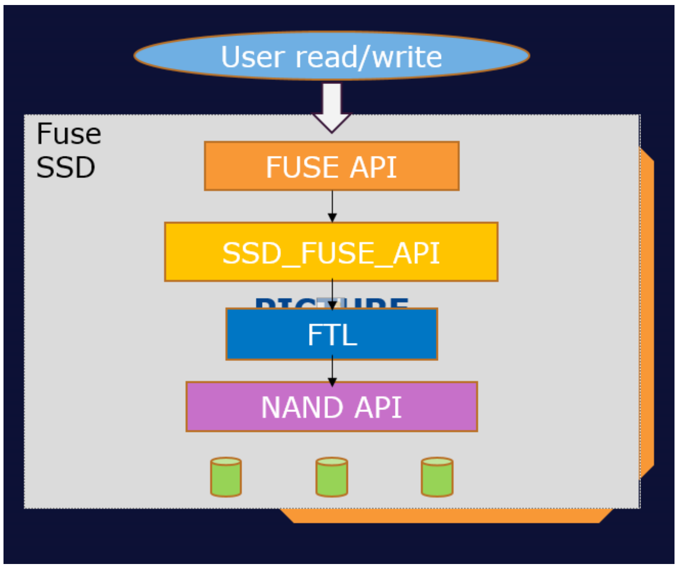
Lab: Simple SSD

Introduction:

SSD use FTL to manage data in NAND and the mapping between PCA and LBA
We already use FUSE to implement a simple file system for simulate SSD.
In this lab, you need to complete four functions to let SSD work.



Goals of this lab:

1. Understand FTL in SSD
2. Understand read modify write
3. Understand the use case of L2P table

Background:

1. **FUSE:**

FUSE is a userspace filesystem framework. It consists of a kernel module (fuse.ko), a userspace library (libfuse.*) and a mount utility (fusermount).
One of the most important features of FUSE is allowing secure, non-privileged mounts. This opens up new possibilities for the use of filesystems. A good example is sshfs: a secure network filesystem using the sftp protocol.
The userspace library and utilities are available from the FUSE homepage

2. LBA, PCA, L2P, P2L:

Because NAND flash's characteristic, PCA(physical cluster address) are more than LBA(logical block address). When user modify the data in NAND flash, data in NAND flash won't be erase immediately.
Instead, we'll find the new PCA to write the data and modify L2P and P2L table to record the mapping between LBA and new PCA.
So we need L2P table for record the mapping between LBA and PCA.

3. FTL:

FTL is responsible for:

1. manage L2P table
2. manage PCA state
    - valid
    - stale: data is useless but not erase yet
    - empty
3. manage P2L table
4. GC(garbage collection):
   Flash SSDs divide storage into blocks, which are further divided into pages. Data is read and written at the page level but must be erased at the block level. Before erase block, we have to move the valid page to another block.
   GC is not use in this lab, you can see Garbage_collection for more information.

4. Align:

In this lab, we use 512 bytes as a unit for easier implementation instead of 4KB which is real word operation unit.

Environment setup

In this chapter we will walk through how to set up this ssd fuse lab environment in Rpi and how to execute the ssd fuse.

Install Library

Install the fuse packages by cmd below:

1. apt-cache search fuse
2. sudo apt-get update
3. sudo apt-get install fuse3
4. sudo apt-get install libfuse3-dev
5. reboot

Execute ssd fuse

Once you install the packages and reboot, please copy the fuse ssd template provided from Phison to your local folder. Phison will provide following content as descript bellow:

1. Ssd_fuse_header.h: for ssd configuration
    a. **please modify the NAND_LOCATION to your**

current file path.

   i. You can use PWD cmd to get current file path

2. Ssd_fuse_dut.c:
   a. Verification code which will send read and write command to fuse ssd
   b. This file support functions below:

l: get logical size

p: get physical size

r SIZE [OFF]: read SIZE bytes @ OFF (default 0) and output to stdout

w SIZE [OFF]: write SIZE bytes @ OFF (default 0) and from default data

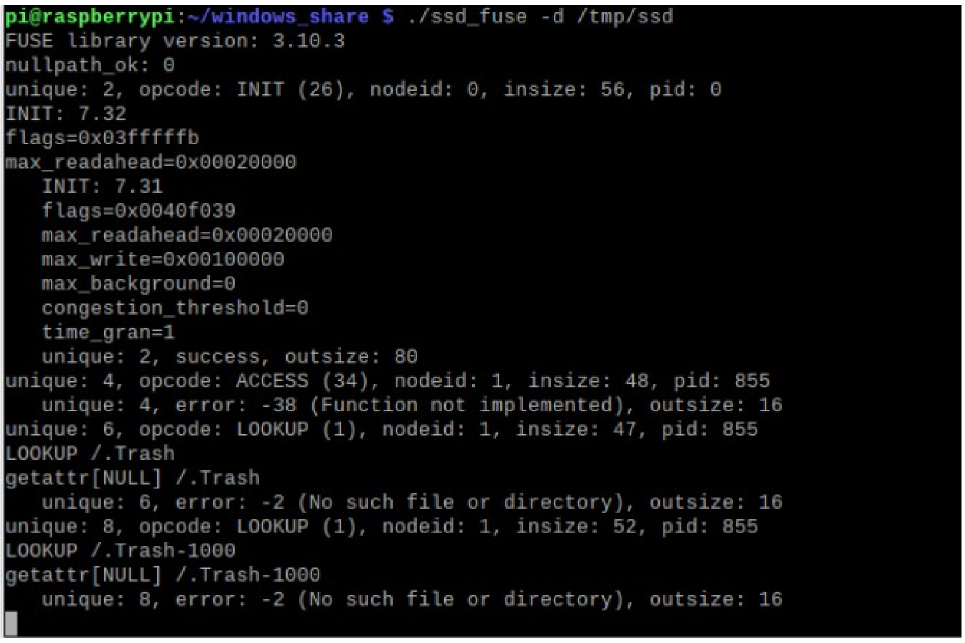
W: get write amplification factor

3. Ssd_fuse.c
   a. SSD implementation code
   b. **This file is what you have to modify to complete this Lab**.
4. Makefile
   a. Command that how to build ssd_fuse_dut.c and ssd_fuse.c

The following steps are how to execute fuse ssd

1. open 2 terminals in your Rpi (we call it Ta and Tb separately)
   a. Ta terminal will run fuse ssd code
   b. Tb terminal will run verification code
2. Compile ssd_fuse.c by following cmd
   a. Ta#gcc –Wall ssd_fuse.c `pkg-config fuse3 --cflags --libs` -D_FILE_OFFSET_BITS=64 –o ssd_fuse
      i. Please make sure you use ` (grave accent) not ‘ (Apostrophe)
3. Compile ssd_fuse_dut.c by following cmd
   a. Tb#gcc –Wall ssd_fuse_dut.c –o ssd_fuse_dut
4. Create folder for fuse library to mount at /tmp in your system
   a. Ta#mkdir /tmp/ssd
5. Execute fuse ssd at debug mode and mount at /tmp/ssd at Ta terminal

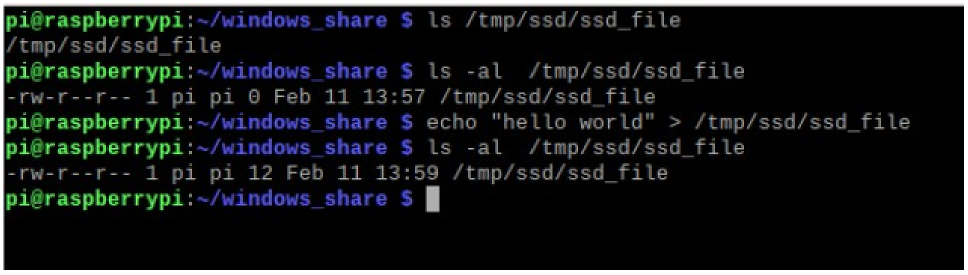
a. Ta#./ssd_fuse –d /tmp/ssd
b. You will see Ta terminal show FUSE library execute as bellow



```
pi@raspberrypi:~/windows_share $ ./ssd_fuse -d /tmp/ssd
FUSE library version: 3.10.3
nullpath_ok: 0
unique: 2, opcode: INIT (26), nodeid: 0, insize: 56, pid: 0
INIT: 7.32
flags=0x03fffffb
max_readahead=0x00020000
   INIT: 7.31
   flags=0x0040f039
   max_readahead=0x00020000
   max_write=0x00100000
   max_background=0
   congestion_threshold=0
   time_gran=1
   unique: 2, success, outsize: 80
unique: 4, opcode: ACCESS (34), nodeid: 1, insize: 48, pid: 855
   unique: 4, error: -38 (Function not implemented), outsize: 16
unique: 6, opcode: LOOKUP (1), nodeid: 1, insize: 47, pid: 855
LOOKUP /.Trash
getattr[NULL] /.Trash
   unique: 6, error: -2 (No such file or directory), outsize: 16
unique: 8, opcode: LOOKUP (1), nodeid: 1, insize: 52, pid: 855
LOOKUP /.Trash-1000
getattr[NULL] /.Trash-1000
   unique: 8, error: -2 (No such file or directory), outsize: 16
```

c. Ssd_fuse will create a file at /tmp/ssd/ssd_file which you can operate it.

6. You can read/write to the /tmp/ssd file/ssd_file by Tb terminal to see Ta terminal show fuse log content like following example.
   a. Tb#ls -al /tmp/ssd/ssd_file
   b. Tb#echo “hello world” > /tmp/ssd/ssd_file
   c. Tb#ls -al /tmp/ssd/ssd_file
      i. You can observe that the /tmp/ssd/ssd_file size is increase



```
pi@raspberrypi:~/windows_share $ ls /tmp/ssd/ssd_file
/tmp/ssd/ssd_file
pi@raspberrypi:~/windows_share $ ls -al  /tmp/ssd/ssd_file
-rw-r--r-- 1 pi pi 0 Feb 11 13:57 /tmp/ssd/ssd_file
pi@raspberrypi:~/windows_share $ echo "hello world" > /tmp/ssd/ssd_file
pi@raspberrypi:~/windows_share $ ls -al  /tmp/ssd/ssd_file
-rw-r--r-- 1 pi pi 12 Feb 11 13:59 /tmp/ssd/ssd_file
pi@raspberrypi:~/windows_share $
```

7. Execute verification code at Tb
   a. Tb#./ssd_fuse_dut /tmp/ssd/ssd_file l
      i. This show current logical size in ssd_file
   b. Tb#./ssd_fuse_dut /tmp/ssd/ssd_file p
      i. This show current physical size in ssd_file

    c. Tb#./ssd_fuse_dut /tmp/ssd/ssd_file r 100 0
        i. This will read 100 Byte at 0 offset
    d. Tb#./ssd_fuse_dut /tmp/ssd/ssd_file w 200 10
        i. This will write 200 Byte at 10 offset

Close ssd fuse program:

When you close ./ssd_fuse program or face a segment fault, remember to unmount the /tmp/ssd and rm the folder. Create a new /tmp/ssd every time when you run ./ssd_fuse program

- Fusermount –u /tmp/ssd
    - Unmount /tmp/ssd
- Rm –rf /tmp/ssd
- Mkdir /tmp/ssd

Requirement

We provide a step-by-step tutorial to guide you to make your original kernel works with virtual memory by fuse.

ssd_fuse.c: This file need to implement the functions as above.

1. User read/write will forward into fuse ssd by fuse library
2. SSD_FUSE_API
    a. Change data offset into LBA
    b. LBA is 512B unite
3. FTL
    a. Handle LBA, PCA mapping
    b. PCA is 512B unite
    c. GC
4. NAND API
    a. Handle storage read/write

b. Storage is 512B align

There are five functions have to be complete.

- **ssd_do_read: We already implement**
  1. Change offset into LBA
  2. Divide read cmd into 512B package by size
  3. Copy data into src_buffer

  Please implement: call ftl_read function and handle result

- **ssd_do_write: We already implement**
  1. Change offset into LBA
  2. Divide write cmd into 512B package by size

  Please implement: call ftl_write function and handle result(because
  the data may not align 512B, so must implement read modify write)

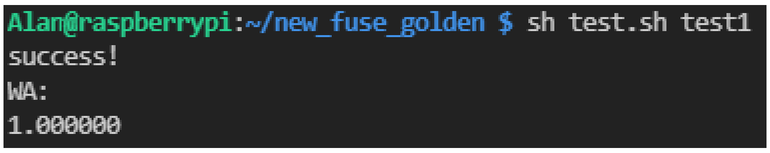
- **ftl_read: Please implement functions as below**
  1. Check L2P to get PCA
  2. Send read data into tmp_buffer
- **ftl_write: Please implement functions as below**
  1. Allocate a new PCA address
  2. Send NAND-write cmd
  3. Update L2P table
- **GC: Please implement functions as below**
  1. Decide the target block to be erase
  2. Move all the valid data that in target block to another block
  3. Erase the target block when all the data in target block are stale
  4. Mark the target block as available
  5. Continue until the number of blocks that you erased reach your goal

  Also, you have to decide when to start and end GC.

Test

We provide two tests to verify your implementation.

- sh test.sh test1
- sh test.sh test2



```
Alan@raspberrypi:~/new_fuse_golden $ sh test.sh test1
success!
WA:
1.000000
```

If you pass the test stdout will print success

We also have 5 undisclosed tests to test your implementation and use your WA for ranking.