

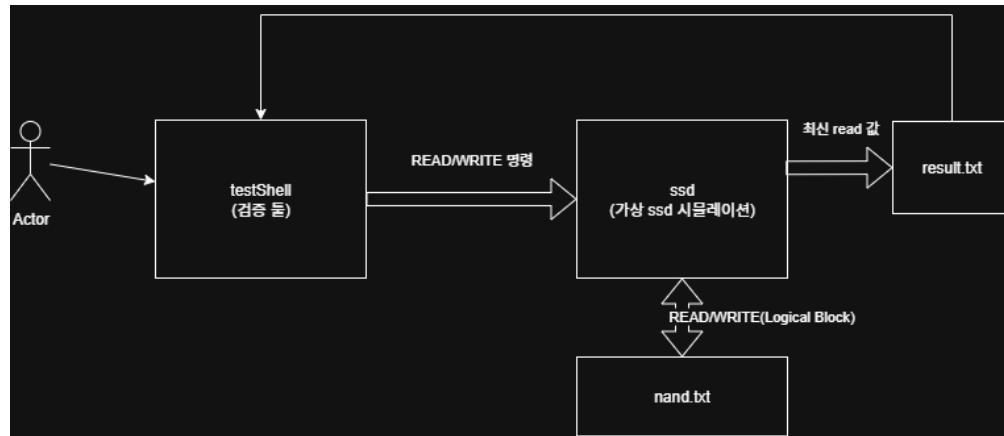
SSD Simulator & Test Shell

개요

이 프로젝트는 간단한 NAND 플래시 시뮬레이터(ssd)와 이를 제어하는 셸 프로그램(testShell)로 구성되어 있습니다.

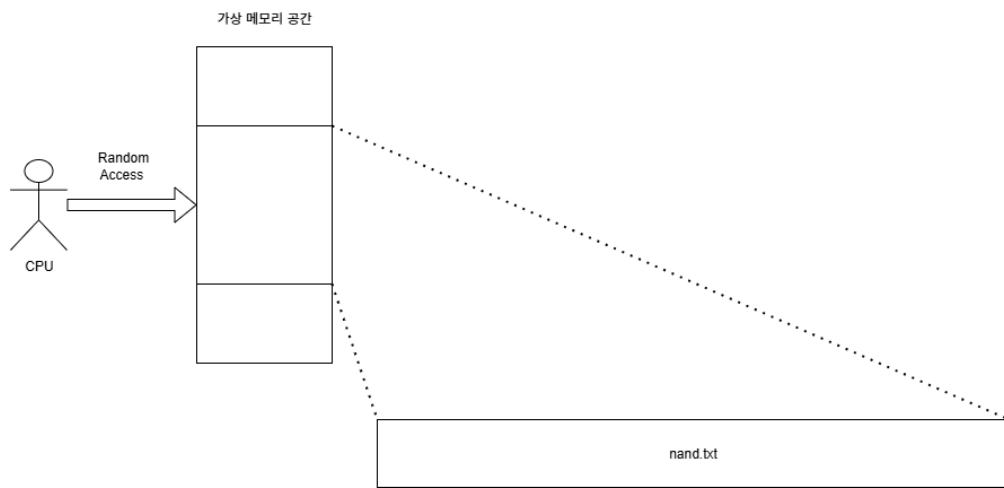
ssd는 nand.txt 파일을 메모리 매핑하여 논리 블록(LBA) 단위로 데이터를 읽고 쓰며,
testShell은 이를 명령어 기반으로 제어할 수 있도록 제공합니다.

구조도



차별점

nand.txt 접근 시, mmap으로 파일을 특정 가상주소공간에 매핑하여 nand에 read/write하는 것을 가상으로 구현



장점 :

1. read/write 시스템 콜을 통해 파일에 접근 X -> 가상주소에 random access하므로 overhead 감소.
2. 바이너리 형태로 데이터를 저장함에도 불구하고 가상주소로 접근하므로, 특정 logical block에 접근하기 용이하다. ex) mappedAddress[logical block 주소] = 0x12345678;

mmap은 가상주소일부분과 물리메모리의 페이지를 매핑한다. 하지만 처음부터 모든 페이지가 메모리에 로딩 되진 않는다. 접근하려는 페이지가 메모리에 없을 때, page fault 발생한다. 이때 일반 페이지는 swap 영역에서

페이지를 가져오는데 반해, mmap 페이지는 파일에서 직접 페이지를 가져온다.

중요한 점은 mmap으로 매핑된 가상주소에 write를 한다고 해서, 즉시 파일에 반영되는 것은 아니라는 것이다. write를 하는건 물리메모리의 페이지에 write하는 것이다. 이후 커널이 적절한 시간에 물리메모리 페이지와 디스크의 파일을 동기화한다.(즉시 동기화 하고 싶다면 msync() 사용)

⚙️ SSD Simulator (ssd)

❖ 실행 형식

```
./ssd <command> <arguments...>
```

❖ 개요

nand.txt 파일을 메모리 맵핑(mmap)하여 100개의 논리 블록(0~99)을 시뮬레이션합니다.

각 블록은 32비트(4바이트) 크기를 가지며, write 결과는 nand.txt, read 결과는 result.txt에 기록됩니다.

① write

- 설명:** 지정한 논리 블록 주소(LBA)에 32비트 16진수 데이터를 기록합니다.
- 사용법:** `./ssd W <LBA> <hex32>`
- 예시:** `./ssd W 10 0x1234ABCD`
- 동작:**
 - `nand.txt`를 mmap()으로 열어 해당 LBA 위치에 데이터를 기록
 - `msync()`를 호출하여 파일에 즉시 반영

```
M] 1440660 > result.txt
1 0xABCD1234

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● ssafy@ssafy-VirtualBox:~/ssd-project/1440660$ ./ssd W 27 0xABCD1234
● ssafy@ssafy-VirtualBox:~/ssd-project/1440660$ ./ssd R 27
○ ssafy@ssafy-VirtualBox:~/ssd-project/1440660$
```

② read

- 설명:** 지정한 LBA의 데이터를 읽어 result.txt에 출력합니다.
- 사용법:** `./ssd R <LBA>`
- 예시:** `./ssd R 10`
- 출력(result.txt):**

0x1234ABCD

The screenshot shows a terminal window with several tabs at the top: 'testShell.c', 'Makefile', 'result.txt' (which is the active tab), and 'ssd.c'. Below the tabs, the terminal displays the command '1440660 > result.txt' followed by the output '1 0xABCD1234'. At the bottom of the terminal, there is a scroll history with three entries:

- ssafy@ssafy-VirtualBox:~/ssd-project/1440660\$./ssd W 27 0xABCD1234
- ssafy@ssafy-VirtualBox:~/ssd-project/1440660\$./ssd R 27
- ssafy@ssafy-VirtualBox:~/ssd-project/1440660\$

Below the terminal, there is a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS.

💻 Test Shell (`testShell`)

❖ 실행 형식

`./testShell`

❖ 개요

`testShell`은 명령 기반으로 `ssd` 프로그램을 호출하는 셸입니다.

각 명령은 `fork()` + `exec()`를 통해 `ssd`를 실행하며, 결과는 `result.txt`로부터 읽어옵니다.

입력 명령을 통해 블록 단위 읽기/쓰기, 전체 테스트 수행이 가능합니다.

① write

- **설명:** 지정한 LBA에 32비트 16진수 값을 기록합니다.
- **사용법:** `write <LBA> <hex32>`
- **예시:** `write 5 0xAABBCCDD`

The screenshot shows a terminal window with the following interface elements:

- Top bar: C testShell.c, M Makefile, ≡ result.txt, C ssd.c
- Text area:

```
1440660 > ≡ result.txt
1 0x0102ABCD
```
- Bottom navigation bar: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), PORTS
- Terminal log:

```
ssafy@ssafy-VirtualBox:~/ssd-project/1440660$ ./testShell
>> write 1 0x0102ABCD
>> read 1
0x0102ABCD
>> []
```

② read

- **설명:** 지정한 LBA의 값을 읽어 콘솔에 출력합니다.
- **사용법:** `read <LBA>`

The screenshot shows a terminal window with the following interface elements:

- Top bar: C testShell.c, M Makefile, ≡ result.txt, C ssd.c
- Text area:

```
1440660 > ≡ result.txt
1 0x0102ABCD
```
- Bottom navigation bar: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), PORTS
- Terminal log:

```
ssafy@ssafy-VirtualBox:~/ssd-project/1440660$ ./testShell
>> write 1 0x0102ABCD
>> read 1
0x0102ABCD
>> []
```

③ fullwrite

- **설명:** 전체 100개 LBA(0~99)에 랜덤 데이터를 씁니다.
- **사용법:** `fullwrite`

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

○ ssafy@ssafy-VirtualBox:~/ssd-project/1440660$ ./testShell
>> fullwrite 0xAABB11FF
>> []
```

4 fullread

- 설명: 전체 LBA의 데이터를 읽어 콘솔에 출력합니다.
- 사용법: `fullread`

```
○ ssafy@ssafy-VirtualBox:~/ssd-project/1440660$ ./testShell
>> fullwrite 0xAABB11FF
>> fullread
[0] : 0xAABB11FF
[1] : 0xAABB11FF
[2] : 0xAABB11FF
[3] : 0xAABB11FF
[4] : 0xAABB11FF
[5] : 0xAABB11FF
[6] : 0xAABB11FF
[7] : 0xAABB11FF
[8] : 0xAABB11FF
[9] : 0xAABB11FF
[10] : 0xAABB11FF
[11] : 0xAABB11FF
[12] : 0xAABB11FF
[13] : 0xAABB11FF
[14] : 0xAABB11FF
[15] : 0xAABB11FF
[16] : 0xAABB11FF
[17] : 0xAABB11FF
[18] : 0xAABB11FF
[19] : 0xAABB11FF
[20] : 0xAABB11FF
[21] : 0xAABB11FF
[22] : 0xAABB11FF
[23] : 0xAABB11FF
[24] : 0xAABB11FF
[25] : 0xAABB11FF
[26] : 0xAABB11FF
```

5 testapp1

- 설명:
모든 LBA(0~99)에 랜덤 값을 기록한 뒤 다시 읽어와 일치 여부를 검증합니다.
- 사용법: `testapp1`

LBA	Written Value	Read Value	Result
0	0xCB96F53	0xCB96F53	Success
1	0xC66841E7	0xC66841E7	Success
2	0xB14C689D	0xB14C689D	Success
3	0x0D6ADE17	0x0D6ADE17	Success
4	0x7ADA4757	0x7ADA4757	Success
5	0x2696E0D7	0x2696E0D7	Success
6	0xFAED17B5	0xFAED17B5	Success
7	0xA3D3764F	0xA3D3764F	Success
8	0x577F6A5B	0x577F6A5B	Success
9	0xC88638BF	0xC88638BF	Success
10	0x57B76919	0x57B76919	Success
11	0x5CA31729	0x5CA31729	Success
12	0x63D46D91	0x63D46D91	Success
13	0xC45C7EEF	0xC45C7EEF	Success
14	0xECF0EC77	0xECF0EC77	Success
15	0xE1FBE872	0xE1FBE872	Success
16	0xFF515248	0xFF515248	Success
17	0x465CCB00	0x465CCB00	Success
18	0x6D10BD3D	0x6D10BD3D	Success
19	0xD757EABB	0xD757EABB	Success
20	0x5530C774	0x5530C774	Success
21	0x0A42CAF0	0x0A42CAF0	Success
22	0xF298841D	0xF298841D	Success
23	0xA6729437	0xA6729437	Success
24	0x4ADEFE92	0x4ADEFE92	Success
25	0x82AEFFCD	0x82AEFFCD	Success

6 testapp2

- 설명: LBA 0~5에 고정된 값을 기록한 후, 다른 값으로 덮어써서 데이터가 정상적으로 갱신되는지 확인합니다.
- 사용법: `testapp2`

LBA	Prev Value	Written Value	Read Value	Result
0	0xAAAABBAA	0x12345678	0x12345678	SUCCESS
1	0xAAAABBAA	0x12345678	0x12345678	SUCCESS
2	0xAAAABBAA	0x12345678	0x12345678	SUCCESS
3	0xAAAABBAA	0x12345678	0x12345678	SUCCESS
4	0xAAAABBAA	0x12345678	0x12345678	SUCCESS
5	0xAAAABBAA	0x12345678	0x12345678	SUCCESS

total result : SUCCESS
 >> []

7 help

- 설명: 사용 가능한 명령 목록을 표시합니다.

- 사용법: `help`

```
>> help
Available commands:
  write <logical_block_address(0 - 99)> <value(hex32)>      : Write 32-bit value to given logical block.
  read <logical_block_address(0 - 99)>                          : Read 32-bit value from given logical block.
  fullwrite <value(hex32)>                                     : Write the same 32-bit value to all logical blocks.
  fullread                                         : Read and print all logical blocks' value.
  help                                           : Show this help message.
  exit                                           : Exit the shell.
>> []
```

8 exit

- 설명: 셸 프로그램을 종료합니다.
- 사용법: `exit`

```
>> exit
Bye!
ssafy@ssafy-VirtualBox:~/ssd-project/1440660$ []
```