

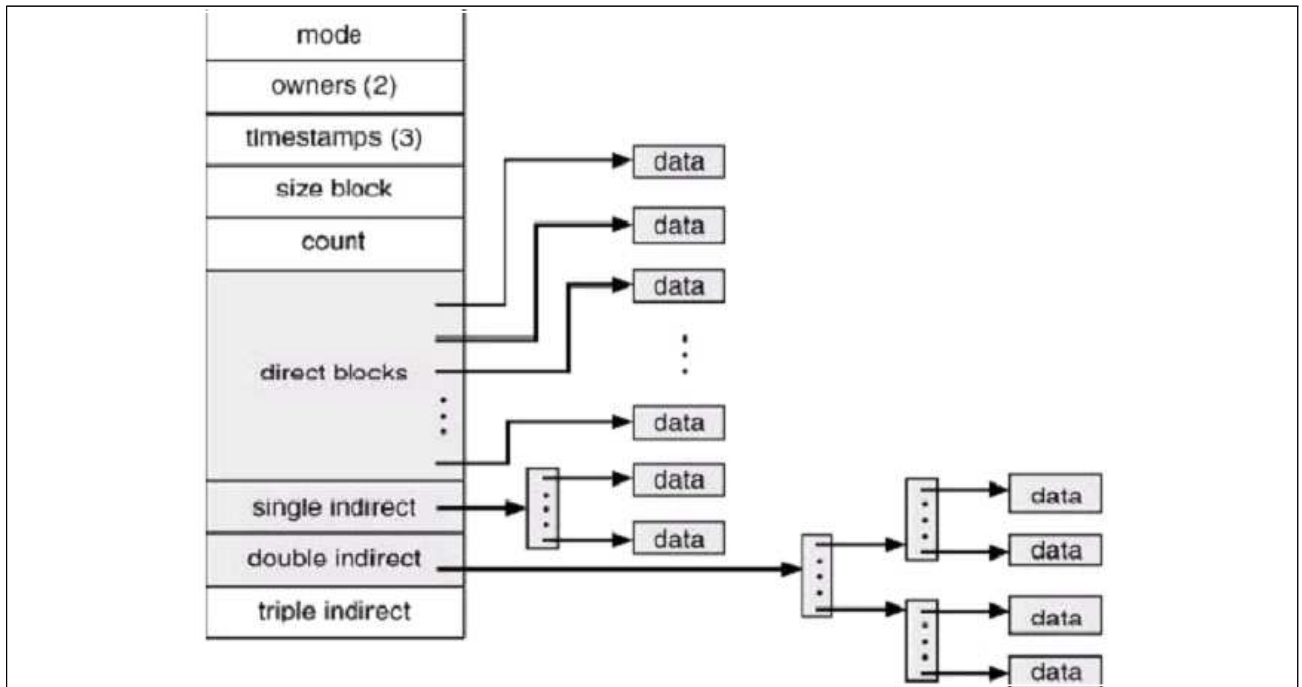
과제 #8 : Indirect Block

○ 과제 목표

- File System의 Indirect Block 이해 및 구현

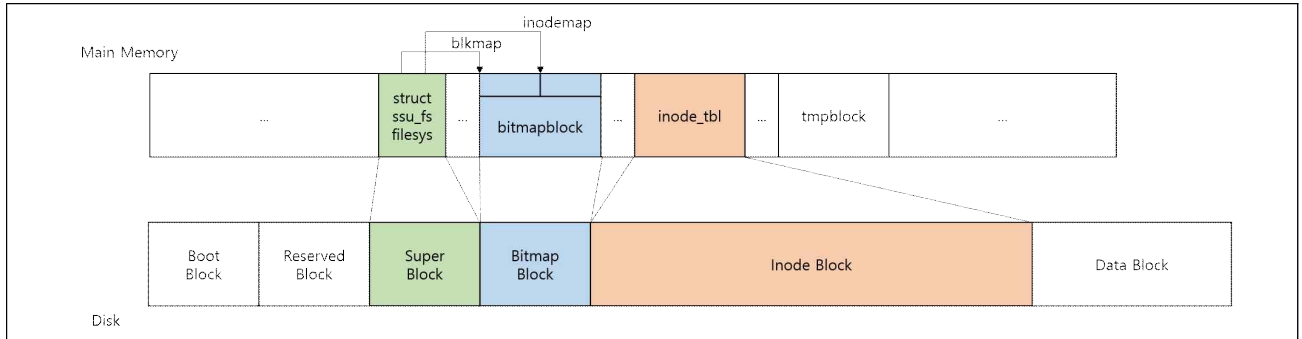
○ 기본 배경 지식

- inode 정의



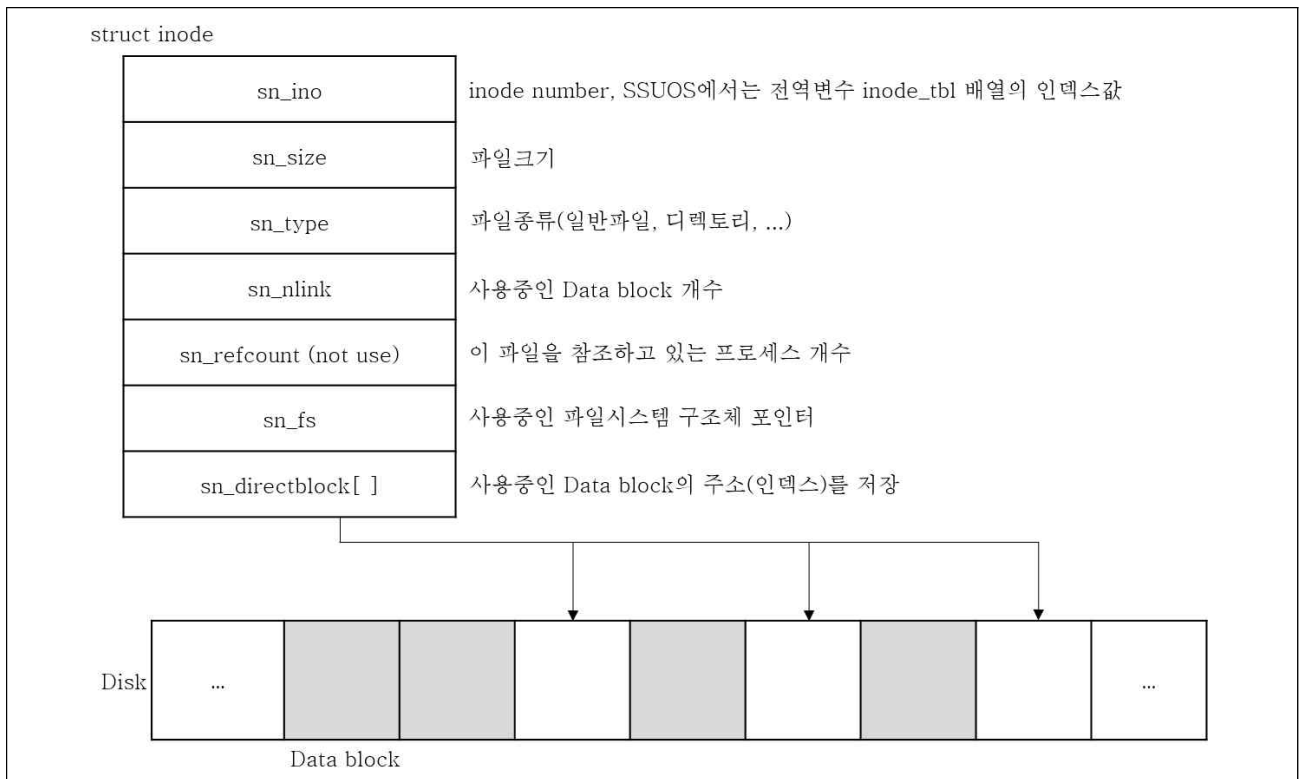
- ✓ Linux에서 파일을 관리하기 위한 객체로, 파일이 새로 생성되면 만들어짐
- ✓ 파일의 모든 정보를 관리함
 - 파일에 속한 블록 위치 (index block 방법과 유사)
 - 파일 소유자 및 접근 권한
 - 파일 시간 정보
 - 파일 유형 : 커널은 정규 파일 뿐만 아니라 디렉토리, 디바이스, 파이프, 소켓 등도 파일이라는 추상화 객체로 관리
- ✓ Disk에 정적으로 존재

- SSUFS 디스크 블록 구조



- ✓ SSUFS에서는 디스크를 4096B(4KB) 단위로 나누어서 관리
- ✓ 처음 0,1번 블록은 사용하지 않고 2번 블록에 디바이스와 파일시스템에 관한 정보를 담은 super_block구조체가 저장됨
- ✓ 3번블록은 반으로 나눠 앞 2KB는 Block Bitmap, 뒤 2KB는 Inode Bitmap으로 사용
- ✓ 4~7번 블록은 Inode Block, 64B인 inode를 256개까지 저장할 수 있음
- ✓ 8번 블록부터는 Data Block으로 파일의 내용을 저장하는 데에 사용됨
- ✓ Data Block은 Block Bitmap을 참조하여 할당되지 않은 Data Block중 가장 적은 index의 Data Block을 할당함

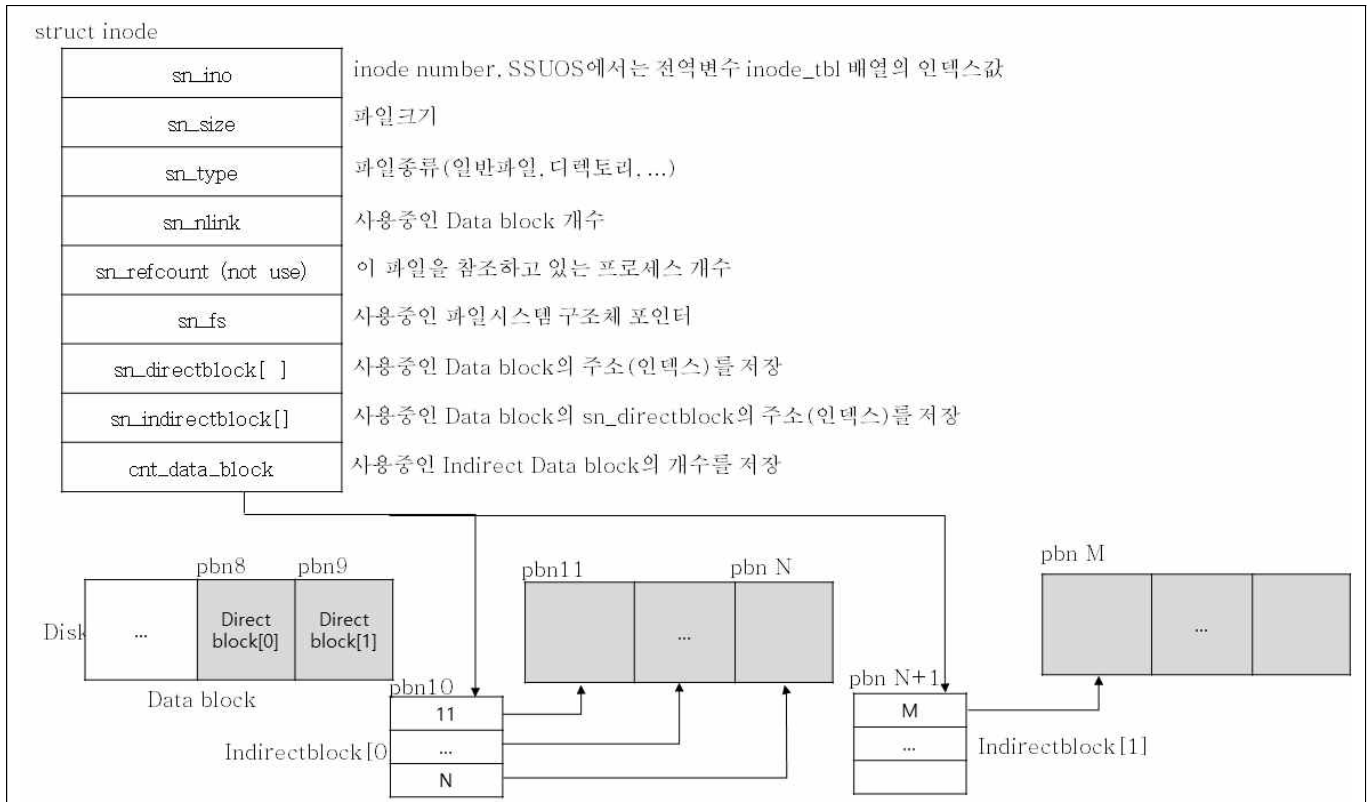
- 기존 SSUFS Inode 구조(Direct Block만을 사용)



- ✓ SSUOS는 간단한 Inode 구조를 제공함(64B)
- ✓ 파일을 생성할때마다 inode가 새로 생성되며, inode bitmap을 통해 사용가능한 inode number를 찾아 할당받음
- ✓ 기존 SSUOS는 inode 구조체는 10개의 Direct block로 구성

○ 과제 내용

- 새로운 SSUFS Inode 구조(Indirect Block 사용)



- ✓ 본 과제에서 SSUOS는 Single indirect block 구조를 제공함(64B)
- ✓ 새로운 inode 구조체는 2개의 Direct block, 8개의 Indirect block으로 구성
- ✓ 파일의 크기가 커서 Direct block만으로 데이터를 저장하지 못할 때, Indirect block을 사용함
- ✓ Indirect block은 1024개의 Data block의 index를 저장하며, 각 index는 Indirect block을 사용해 할당받은 Data block의 index가 저장됨

- int lbn_to_pbn() 함수 구현

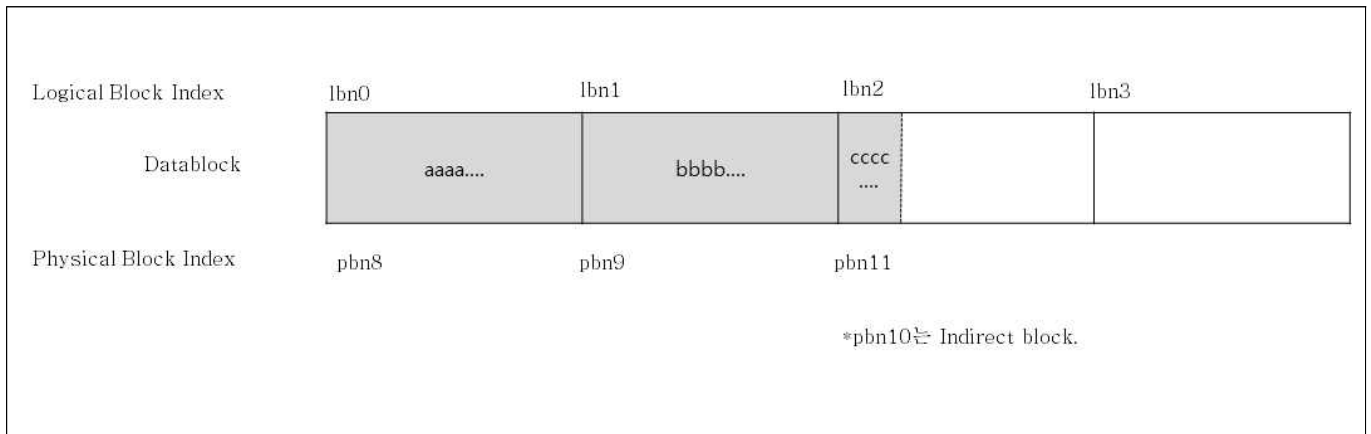
- ✓ 함수의 인자로 inode 구조체와 Logical Block Number(lbn)가 들어감
- ✓ Direct block에 해당하는 1 이하의 lbn이 인자로 주어진 경우, sn_directblock변수에 저장된 Data block index를 반환함
- ✓ Indirect block에 해당하는 2 이상의 lbn이 인자로 주어진 경우, sn_indirectblock변수가 가리키는 Indirect block에 저장된 Data block index를 반환함
- ✓ 만약 접근하려는 index가 2 이상이며, Indirect Data Block이 할당 안 된 경우, Indirect block을 할당하고 실제 데이터가 저장될 Data block을 할당해야 함
- ✓ Data block이 추가되는 경우는 항상 bitmap 구조체 및 관련 함수를 통해 sn_fs의 bitmap 정보를 갱신

- int inode_write() 함수 수정

- ✓ 주어진 과제코드에서 inode_write() 함수는 Direct block만을 사용

- ✓ 주어진 과제 코드를 바탕으로 Indirect block이 추가된 inode_write() 함수로 수정
- ✓ 함수의 인자로 파일의 offset, 쓰여질 데이터의 길이인 len과 쓰여질 내용인 buf가 주어짐
 - offset은 파일의 크기를 넘지 말아야 함
 - len은 buf의 크기를 넘는 경우 buf의 끝까지만 쓰기작업을 수행함
- ✓ [offset : offset+len]의 범위가 여러 블록을 접근하는 경우도 쓰기작업을 수행할 수 있어야함
- ✓ inode_write()를 통해 파일의 크기가 수정될 때마다 관련 inode의 내용이 갱신되어야 함
- ✓ 성공시 0, 실패시 -1 반환

- inode_write() 예시

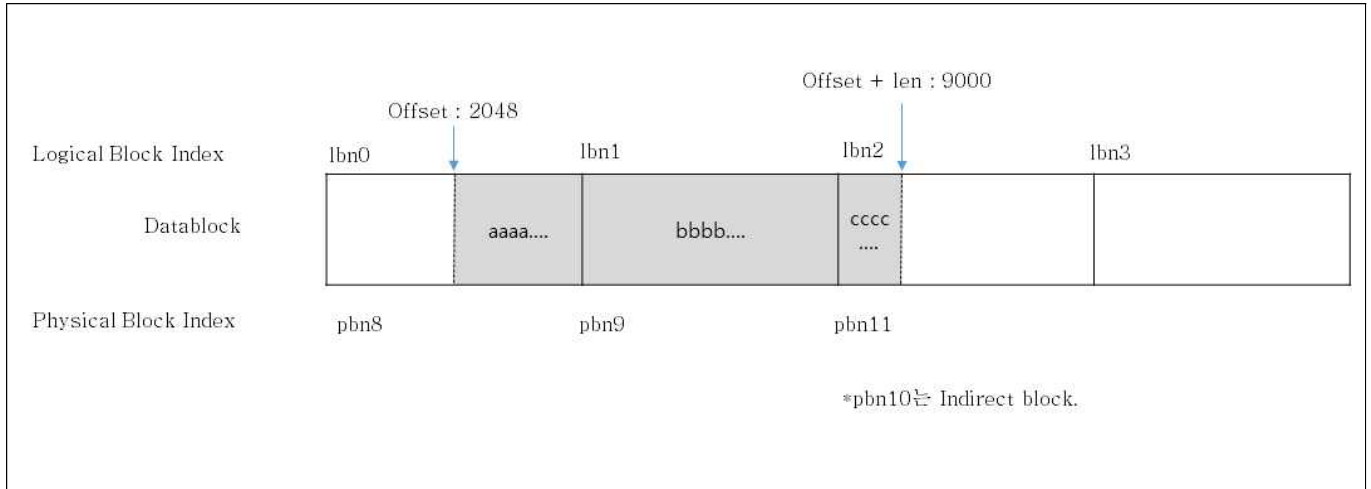


- ✓ 위 예시는 offset이 0, len이 9000 인자로 들어왔을 때, inode_write() 함수의 Data Block에 쓰기 과정임
- ✓ 총 3개의 Data Block(lbn0 , lbn1 , lbn2)에 buf의 내용을 저장
- ✓ lbn0 과 lbn1은 sn_directblock[0], sn_directblock[1]를 참조하여 pbn8, pbn9에 쓰기를 수행
- ✓ pbn10은 Indirect block으로 쓰임
- ✓ lbn2는 pbn10의 첫 번째 index를 참조하여 pbn11에 lbn2에 해당하는 내용을 저장
- ✓ lbn0의 [0 : 4095]번지에 buf의 내용을 저장
- ✓ lbn1의 [0 : 4095]번지에 buf+4096에 저장
- ✓ lbn2의 [0: 807]번지에 buf+8192에 저장

- int inode_read() 함수 수정

- ✓ 주어진 과제코드에서 inode_read() 함수는 Direct block만을 사용
- ✓ 주어진 과제 코드를 바탕으로 Indirect Data Block이 추가된 inode_read() 함수로 수정
- ✓ 함수의 인자로 파일의 offset, 파일의 길이인 len과 읽을 내용을 저장할 buf가 주어짐
 - offset은 파일의 크기를 넘지 말아야 함
 - len은 buf의 크기를 넘는 경우 buf의 끝까지만 읽기작업을 수행함
 - offset+len가 파일의 크기를 넘을 경우, offset부터 파일의 Data의 끝까지 read 과정이 이루어짐
- ✓ [offset : offset+len]의 범위가 여러 블록을 접근하는 경우도 읽기작업을 수행할 수 있어야함
- ✓ 성공시 0, 실패시 -1 반환

- inode_read() 예시



- ✓ 위 예시는 offset이 2048 , len이 6952 인자로 들어왔을 때, inode_read() 함수의 Data Block으로 부터 읽기 과정임
- ✓ 총 3개의 Data Block(lbn0, lbn1, lbn2)에서 읽어들이 범위[offset : offset+len]을 buf에 저장
- ✓ lbn0과 lbn1은 sn_directblock[0], sn_directblock[1]를 참조하여 pbn8, pbn9에 읽기를 수행
- ✓ Physical Datablock[10]은 Indirect data block으로 쓰임
- ✓ lbn2는 pbn10의 첫 번째 index를 참조하여 pbn11에서 읽을 내용을 buf에 저장
- ✓ lbn0은 [2048 : 4095]번지를 buf에 저장
- ✓ lbn1은 [0 : 4095]번지를 buf+2048에 저장
- ✓ lbn2은 [0 : 807]번지를 buf+6144에 저장

○ 주의 사항

- ✓ 불필요한 변수 및 라이브러리를 추가하는 경우 감점
- ✓ 과제 구현을 위한 함수들의 불필요한 프로토타입 변경을 하는 경우 감점
- ✓ 과제 수행결과 및 과제 내용의 예시는 하나의 파일만이 Data를 read/write 한다고 가정하여 설명
- ✓ 프로세스의 가상메모리 크기가 한정되어 있음 -> 함수 내에 tmp[4096] 같이 큰 배열을 잡을 경우 스택오버플로우가 발생할 수 있음
 - 단, 허용 가능한 배열의 크기는 프로세스의 코드 및 데이터 크기에 종속적임
 - 본 과제에서는 tmpblock[4096]/ tmpblock_indirect[4096]을 전역변수로 선언함

○ 과제 수행 방법

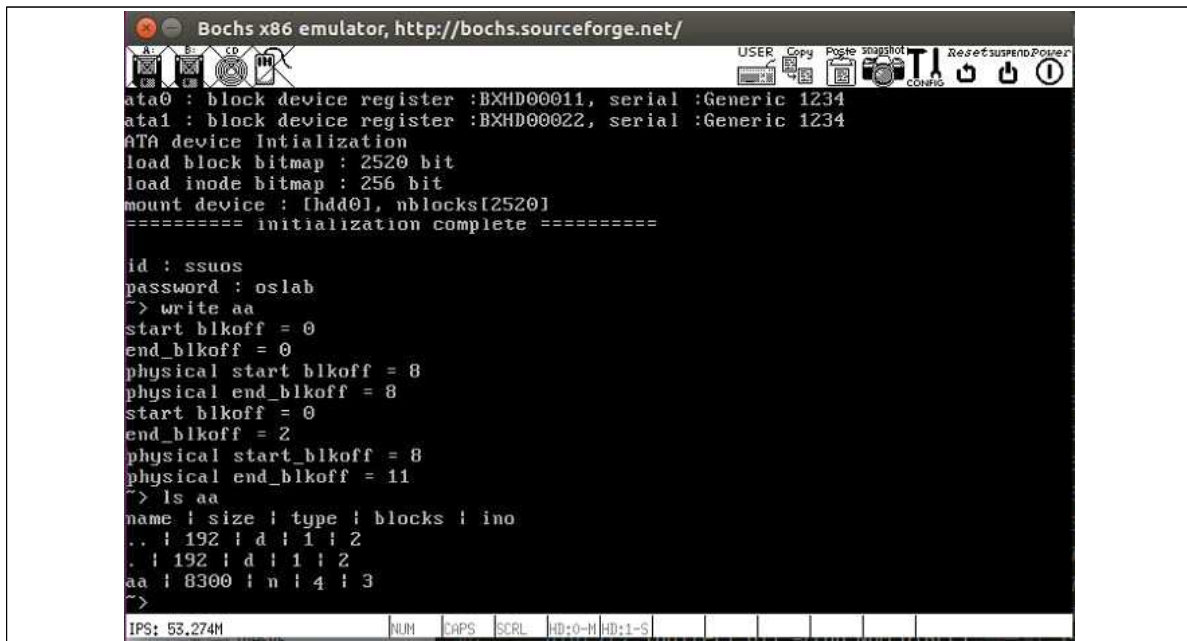
- (1) lbn_to_pbn0 함수 구현

- ✓ ssu_fs 구조체, inode 구조체와 bitmap 구조체 및 관련 함수를 분석하여 비트맵 갱신 및 Block을 할당하는 과정을 이해
- ✓ 인자로 주어진 lbn이 1이하인 경우 Direct datablock을 참조하여 pbn에 바로 접근
- ✓ 인자로 주어진 lbn이 2이상인 경우 Indirect Datablock를 참조하여 lbn의 pbn을 알아냄
- ✓ Indirect datablock에 대한 내용은 tmpblock_indirect 에 저장하여 접근함
- ✓ lbn에 해당하는 Datablock에 접근하기 위해 Indirect Datablock이 할당이 안 된 경우, 할당을 하고 Indirect Datablock의 index를 inode 구조체에 저장 및 inode 구조체의 관련 내용을 갱신해야함
- ✓ lbn에 해당하는 Datablock에 접근하기 위해 Direct Datablock이 할당이 안 된 경우, 할당을 하고

- 관련 해당 lbn을 관리하는 Indirect datablock의 index 및 inode 구조체의 내용을 갱신해야함
- ✓ Datablock을 할당하는 경우, 비트맵 갱신을 해야함
- (2) inode_write() 함수 수정
- ✓ 인자로 주어진 offset 과 len을 통해 lbn의 시작과 끝의 인덱스(lbn_start, lbn_end)와 Datablock의 개수 (lbn_end - lbn_start)를 저장
 - ✓ 내용을 저장할 Datablock의 개수만큼 반복문을 돌며 Datablock을 갱신
 - ✓ 반복문의 내용은 다음과 같음
 - 1) lbn_to_pbn()함수를 통하여 pbn을 통해 tmpblock에 한 block씩 저장
 - 2) tmpblock에 buf의 해당 블록의 내용을 갱신
 - 3) 갱신된 tmpblock을 다시 pbn에 저장
 - ✓ ls 명령어를 통하여 정상적으로 Datablock이 저장되었는지 확인
- (3) inode_read() 함수 수정
- ✓ 인자로 주어진 offset 과 len을 통해 Logical Block의 시작과 끝의 인덱스(lbn_start, lbn_end)와 Datablock의 개수 (lbn_end - lbn_start)를 저장
 - ✓ 읽을 내용을 가진 Datablock의 개수만큼 반복문을 돌며 buf를 채워나감
 - ✓ 반복문의 내용은 다음과 같음
 - 1) lbn_to_pbn()함수를 통하여 pbn을 tmpblock에 한 block씩 저장
 - 2) tmpblock 의 읽을 부분을 buf의 해당 위치에 memcpy()
- (4) lbn_to_pbn()함수 , inode_write() 함수의 검증을 위한 write 명령어
- ✓ ‘write [파일명]’ 을 입력하여 a로 구성된 임의의 배열 char tmp[8300]을 생성
 - ✓ tmp의 8200번지부터 “Hello , ssuos world” 의 문자열을 변경
 - ✓ 과정의 검증을 위해 write(fd, tmp, 8300)을 수행하여 참조되는 모든 블록들의 pbn을 모두 printk() 함수를 통해 출력
 - ✓ shell 로그인 후 “write [파일명]” 명령어를 입력하여 aa 파일을 만들고 위의 과정을 보여 검증
 - ✓ ls명령어를 통하여 “[파일명]” 파일 저장 확인
- (5) lbn_to_pbn()함수 , inode_read() 함수의 검증을 위한 cat 명령어
- ✓ ‘cat [파일명]’ 을 입력하여 (4)를 통해 저장하였던 임의의 배열을 buf에 읽어들이
 - ✓ 과정의 검증을 위해 read(fd, tmp, 8300)을 수행하여 참조되는 모든 블록들의 pbn의 인덱스를 모두 printk() 함수를 통해 출력
 - ✓ buf의 8200번지부터 printk()를 통해 출력하여 “Hello, ssuos world” 의 문자열 확인
 - ✓ (4) 의 write 명령어 호출 후 cat 명령어를 통하여 위의 과정을 보여 검증

○ 과제 수행 결과

- 과제수행 방법 (4)의 실행 예시



The screenshot shows the Bochs x86 emulator interface. The title bar reads "Bochs x86 emulator, http://bochs.sourceforge.net/". The menu bar includes "USER", "Copy", "Paste", "Snapshot", "Reset/suspend/Power", and "CONFIG". The main window displays the following text:

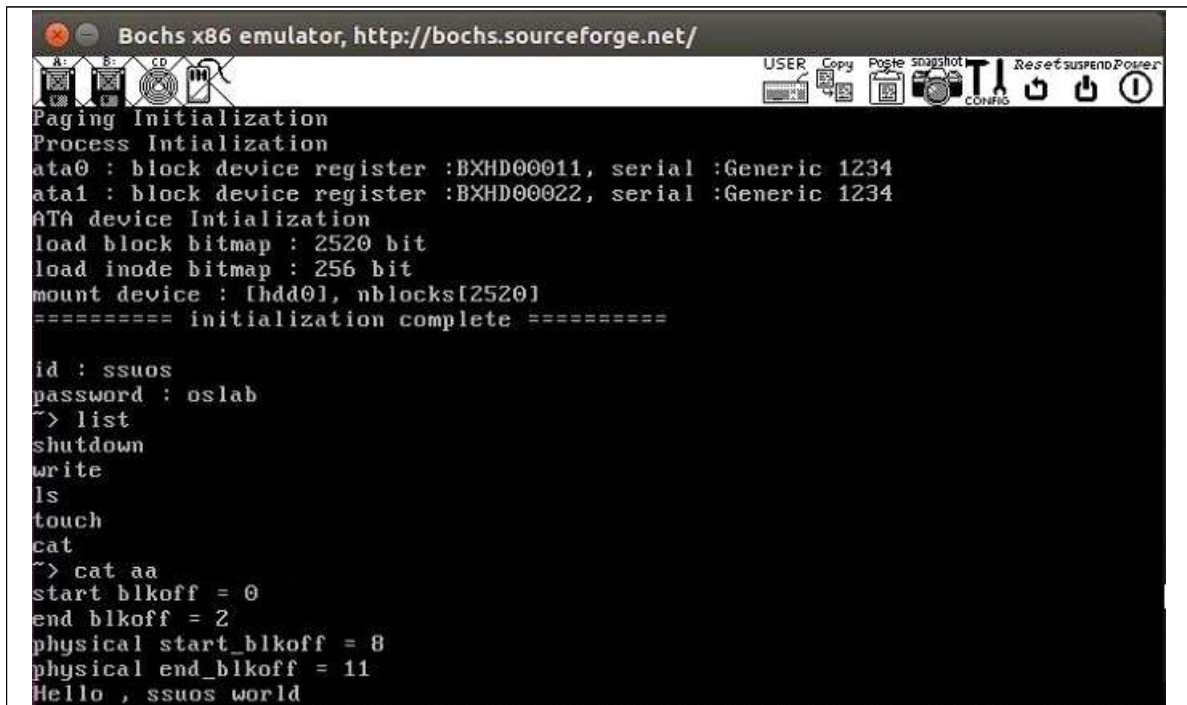
```
ata0 : block device register :BXHD00011, serial :Generic 1234
ata1 : block device register :BXHD00022, serial :Generic 1234
ATA device Initialization
load block bitmap : 2520 bit
load inode bitmap : 256 bit
mount device : [hdd0], nblocks[2520]
===== initialization complete =====

id : ssuos
password : oslab
~> write aa
start blkoff = 0
end_blkoff = 0
physical start_blkoff = 8
physical end_blkoff = 8
start blkoff = 0
end_blkoff = 2
physical start_blkoff = 8
physical end_blkoff = 11
~> ls aa
name | size | type | blocks | ino
.. | 192 | d | 1 | 2
. | 192 | d | 1 | 2
aa | 8300 | n | 4 | 3
~>
```

The status bar at the bottom shows "IPS: 53.274M" and several checkboxes for "NUM", "CAPS", "SCRL", "HD:0-M", and "HD:1-S".

✓ write 이후 4줄은 file open으로 인한 Datablock의 index 출력임

- 과제수행 방법 (5)의 실행 예시



The screenshot shows the Bochs x86 emulator interface. The title bar reads "Bochs x86 emulator, http://bochs.sourceforge.net/". The menu bar includes "USER", "Copy", "Paste", "Snapshot", "Reset/suspend/Power", and "CONFIG". The main window displays the following text:

```
Paging Initialization
Process Initialization
ata0 : block device register :BXHD00011, serial :Generic 1234
ata1 : block device register :BXHD00022, serial :Generic 1234
ATA device Initialization
load block bitmap : 2520 bit
load inode bitmap : 256 bit
mount device : [hdd0], nblocks[2520]
===== initialization complete =====

id : ssuos
password : oslab
~> list
shutdown
write
ls
touch
cat
~> cat aa
start blkoff = 0
end blkoff = 2
physical start_blkoff = 8
physical end_blkoff = 11
Hello , ssuos world
```

○ 과제 제출

- 2018년 12월 13일 (목) 23시 59분 59초까지 과제 게시판으로 제출

○ 배점 기준

- 보고서 15%
 - ✓ 개요 2%
 - ✓ 상세 설계 명세(기능 명세 포함) 10%
 - ✓ 실행 결과 3%
- 소스코드 85%
 - ✓ 컴파일 여부 5%(설계 요구에 따르지 않고 설계된 경우 0점 부여)
 - ✓ 실행 여부 80% ((1)구현 20% , (2)구현 20% (3)구현 20% (4)실행 10% (5)실행 10%)

○ 최소 기능 구현

- (1) lbn_to_pbn() 함수 구현
- (2) inode_write() 함수 수정