시스템 프로그래밍 Term Project1 레포트

2015410119 배상근

2015410120 김용우

제출 일자: 10월 30일(수)

프리데이 사용일수: 0일

목차

1. 각자 맡은 부분 및 주요 기여점

2015410119 배상근

- 배경 지식 작성
- 결과 그래프 작성
- VFS 구조체 분석
- 실험 환경 설정

- 결과값을 통한 파일시스템 쓰기 동작 방식 비교
- 모듈 코드 작성

2015410120 김용우

- 작성한 코드에 대한 설명 작성
- 모듈 코드 작성
- blk-core 수정
- nilfs2 superblock 연결
- VFS 구조체 분석
- 결과 실행 및 결과 값 출력
- Circular queue 구현

2. 개발 환경

가상머신 : Oracle VM VirtualBox

운영체제 : Ubuntu 16.04 LTS

커널 : Linux-4.4.0+

파일시스템 : Ext4, Nilfs2

쓰기 시나리오에 사용한 프로그램 : iozone3_414

3. 배경지식

OS는 여러가지 파일시스템을 사용하기 위해서 Virtual File System을 사용하는데 이번 과제에서 다룬 Ext4와 Nilfs2도 여러 파일시스템 중 하나이다. 파일시스템은 파일의 내용을 담고 있는 data block이 저장되는 디스크상 실제 위치 정보를 결정하는 것에 대한 방법을 다룬 것이라고 볼 수 있다. 따라서 파일시스템의 종류와 이에 따른 디스크상 실제 위치 정보를 결정하는 알고리즘에 따라서 실제 디스크에 써지는 블록주소들이 다르게 결정되게 된다. 이번 텀 프로젝트의 목적은 Ext4와 Nilfs2 파일시스템의 쓰기방식 차이를 알아보는 것이다. 따라서 Ext4와 Nilfs2의 특징들과 쓰기방식들을 살펴보기로 하겠다.

먼저 Ext4에 대해서 살펴보자면 Linux에서 채택한 Ext파일시스템의 발전형으로써 1992년 Linux의 주 파일 시스템으로 ext가 채택된 이후 ext2, ext3를 거쳐서 불안정 버전 기준 2006년 10월 10일, 안정 버전 기준 2008년 10월 21일 Linux에 도입되게 되었다. Ext4의 특징으로는 64비트 기억 공간 제한을 없애고 Extent란 개념을 도입시켜서 ext2와 ext3에서 쓰이던 전통적인 block mapping 방식을 대체한다. 여기서 Extent란 인접한 물리적 블록의 묶음으로써 대용량 파일 접근 성능을 향상시키고 파일 단편화를 줄인다. 그리고 ext2와 ext3에 대한 하위 호환성이 있어서 ext2, ext3 파일 시스템을 ext4로 마운트 하는 것이 가능하다. 이를 통해서 ext4의 새로운 기능들을 이전 파일 시스템에도 사용할 수 있다는 장점이 있다.

Ext파일시스템의 공통적인 특징에 대해서 살펴보자면 Ext는 디스크 저장 매체에 사용되는 파일 시스템으로써 구성은 Superblock, Inode로 구성되며, Inode와 데이터 블록이 구분되어 저장된다.

Inode에 해당 파일에 대한 블록의 번호를 저장하고 추가적인 Write가 필요할 경우에는 블록을 새로 할당받아 데이터를 저장한다. 데이터의 수정이 필요한 경우에는 해당 블록의 내용을 수정하고 그 블록에 덮어쓰는 방식으로 이루어진다. 파일 삭제가 일어날 때는 Inode를 삭제하여 Inode가 가리키고 있는 블록에 다른 작업요청이 들어오면 그 위에 덮어쓰는 방식으로 진행되게 된다.

Nilfs2에 대해서 살펴보자면 Nilfs2는 1988년 John K. Ousterhout와 Fred Douglis에 의해서 처음 고안되었고 1992년 Unix-like Sprite distributed operating system에 적용된 Log structured file system(LFS)에 기반을 두고 있다.

LFS는 기존 파일시스템들이 가지고 있는 read, write, crash recovery등을 개선하기 위해 고안되었다. 가장 획기적이라고 할 수 있는 부분은 write에 관한 것인데 기존 파일시스템은 데이터가 memory에 caching 되었다가 Inode에 할당한 블록주소에 가서 쓰이게 되는데 LFS는 disk write를 disk에 연속되게 처리함으로써 기존 파일시스템의 단점인 seek time을 없앨 수 있게 했다. 연속되게 데이터를 쓰고 맨 뒤에 Inode를 써주면서 이곳저곳 seek 하지않고 맨뒤만 보면 데이터를 찾을 수 있게 했다. 그리고 이 Inode의 위치를 찾기 힘들다는 단점은 Inode Map이란 새로운 구조를 도입해서 Inode들의 위치를 찾을 수 있게 해서 극복했다.

그리고 기존 파일시스템들은 block단위를 write의 단위로 사용했는데 LFS는 block보다 큰 segment를 write의 단위로 사용함으로써 파일 단편화를 개선했다. 여기서 중요한 게 segment cleaning인데 segment cleaning은 파일을 쓸 때 해당 segment에 live data가 있으면 그 live data들을 정리해서 전보다 적은 수의 segment로 만드는 과정이다.

Nilfs2는 플래쉬 저장 매체에 사용되는 파일 시스템으로써 구성은 superblock, Inode, Inode map으로 구성되며 Inode와 데이터 블록을 구분하지 않고 순차적으로 쓰기를 하여 Inode와 데이터가 함께 저장된다.

이번 프로젝트에서는 Ext4와 Nilfs2의 write방식에 주목을 해야 한다. Ext4는 기존의 inode방식을 사용해서 inode가 가리키는 블록 주소에 write를 하기 때문에 순차적이 아닌 임의의 블록위치에 저장을 한다. 반면 NIlfs2는 LFS의 write방식을 사용해서 순차적인 블록위치에 적은 뒤에 마지막에 Inode를 추가할 것이다. 우리가 사용한 시스템은 플래쉬 저장장치인 SSD를 사용하기 때문에 플래쉬 파일시스템인 Nilfs2는 바로 실제 block number에 write하겠지만 Ext4는 바로 write하지 못하고 중간에 FTL의 도움을 받아서 FTL이 실제 block number와 매칭 시킨 뒤 write할 것이다.

Linux커널에서는 Ext4나 Nilfs2와 같은 파일시스템이 블록 넘버를 결정해서 blk-core.c에 존재하는 submit_bio에 전달하게 되면 submit_bio가 실제로 디스크에 적는 역할을 하게 된다. 우리는 Nilfs2와 Ext4가 실제로 동작하는 방식을 확인하기 위해서 submit_bio에서 Nilfs2의 super block을 참조할 수 있게 수정하고 circular queue를 구현해서 submit_bio에 들어가는 block number를 저장한 뒤 Loadable Kernel Module(LKM)을 통해서 그 정보를 빼 오는 방식으로 이번 프로젝트를 구성하였다.

4. 작성한 코드에 대한 설명

1) include/linux/bio.h

```
 */
#ifndef __LINUX_BIO_H
#define __LINUX_BIO_H

#define MY_QUEUE_SIZE 100
struct my_data{
        char *name;
        long long time;
        unsigned long long block_number;
};
```

큐 크기를 정하는 MY_QUEUE_SIZE
큐에 넣을 데이터의 구조체를 정의해주었다.

2) fs/nilfs2/segbuf.c

```
static int nilfs_segbuf_submit_bio(struct nilfs_segment_buffer *segbuf,
                                   struct nilfs_write_info *wi, int mode)
{
        struct bio *bio = wi->bio;
        int err;

        if (segbuf->sb_nbio > 0 &&
            bdi_write_congested(segbuf->sb_super->s_bdi)) {
                wait_for_completion(&segbuf->sb_bio_event);
                segbuf->sb_nbio--;
                if (unlikely(atomic_read(&segbuf->sb_err))) {
                        bio_put(bio);
                        err = -EIO;
                        goto failed;
                }
        }

        bio->bi_end_io = nilfs_end_bio_write;
        bio->bi_private = segbuf;
        bio->bi_bdev->bd_super = segbuf->sb_super; // added
        submit_bio(mode, bio);
```

Nilfs2의 super block을 bio에 연결해 주었다.

3) block/blk-core.c

```
// add structure, function
#include <linux/ktime.h>

struct my_data my_queue[MY_QUEUE_SIZE];
int my_idx = 0;
int my_number = 0;
bool my_queue_enable = false;

void my_enqueue(struct my_data queue[], struct my_data data){
        if(my_number >= MY_QUEUE_SIZE) return;

        my_queue[(my_idx+my_number) % MY_QUEUE_SIZE] = data;
        my_number++;
}

struct my_data my_dequeue(struct my_data queue[]){
        if(my_number <= 0) return;
        struct my_data data = my_queue[my_idx];
        my_idx = (my_idx+1) % MY_QUEUE_SIZE;
        my_number--;
        return data;
}

EXPORT_SYMBOL(my_queue);
EXPORT_SYMBOL(my_idx);
EXPORT_SYMBOL(my_number)
EXPORT_SYMBOL(my_queue_enable);
EXPORT_SYMBOL(my_enqueue);
EXPORT_SYMBOL(my_dequeue);
// add end
blk_qc_t submit_bio(int rw, struct bio *bio)
{
        // add start
        if(my_queue_enable){
                struct my_data data;
                data.name = bio->bi_bdev->bd_super->s_type->name;
                data.time = (long long)(ktime_get().tv64);
                data.block_number = (unsigned long long)bio->bi_iter.bi_sector;
                my_enqueue(my_queue, data);
        }
        // add end
        bio->bi_rw |= rw;

        /*
```

큐 관련 변수, enqueue, dequeue함수를 선언해 주었다.

my_queue_enable이 켜져 있으면 파일 시스템 이름, 시간, 블록 넘버를 받아서 큐에 넣어준다.

4) basic.c (module code)

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/bio.h>

extern struct my_data my_queue[MY_QUEUE_SIZE];
extern int my_idx;
extern int my_number;
extern bool my_queue_enable;
extern void my_enqueue(struct my_data queue[], struct my_data data);
extern struct my_data my_dequeue(struct my_data queue[]);

static int hello_init(void)
{
        while(my_number != 0)
                my_dequeue(my_queue);
        printk(KERN_ALERT "Module init\n");
        my_queue_enable = true;
        return 0;
}
static void hello_exit(void)
{
        printk(KERN_INFO "%d data in queue", my_number);
        while(my_number != 0)
        {
                struct my_data data;
                data = my_dequeue(my_queue);
                printk(KERN_INFO "%-7s %-12lld %-8llu", data.name, data.time ,data.block_number);
        }
        printk(KERN_ALERT "Module exit\n");
        my_queue_enable = false;
}
module_init(hello_init);
module_exit(hello_exit);

MODULE_AUTHOR("KYW");
MODULE_DESCRIPTION("It's Simple");
MODULE_LICENSE("GPL");
MODULE_VERSION("NEW");
```

my_data, MY_QUEUE_SIZE를 정의해둔 bio.h를 include했다.

모듈을 올리면 큐를 비워주고, my_queue_enable을 켜줘서 큐에 데이터를 받을 수 있게 했다.

모듈을 내릴 때, 큐를 비우면서 갖고 있는 정보를 출력하게 하고, my_queue_enable을 꺼줘서 큐를 비활성화 시켰다.

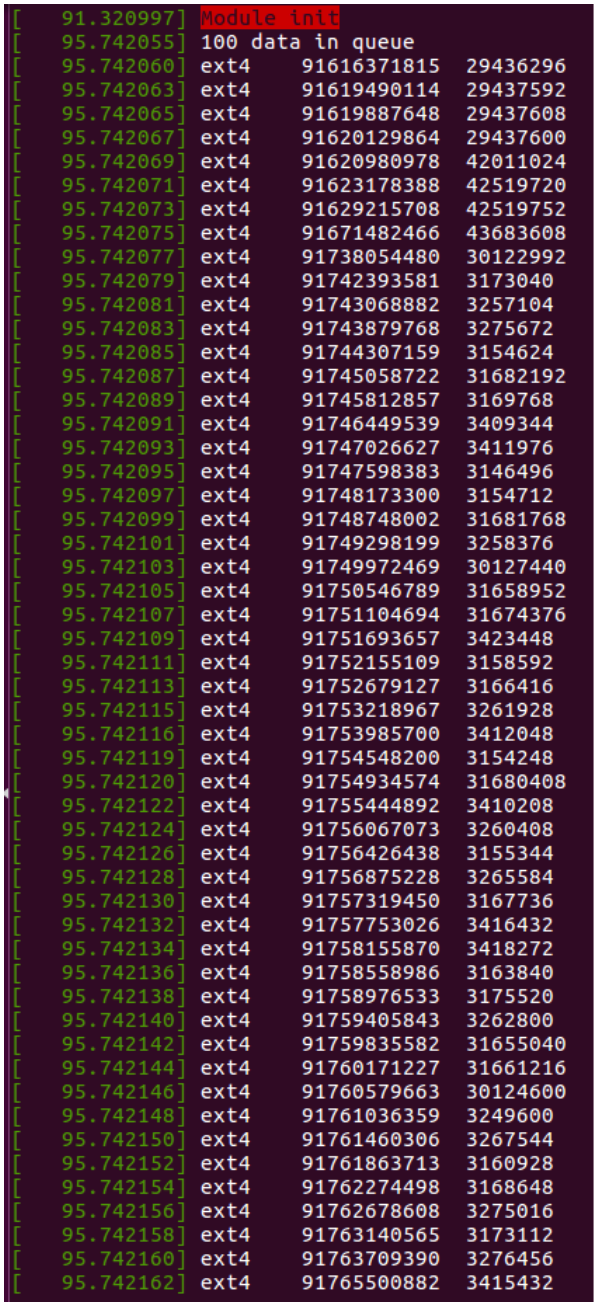
5. 실행 방법에 대한 설명과 결과 캡쳐 화면

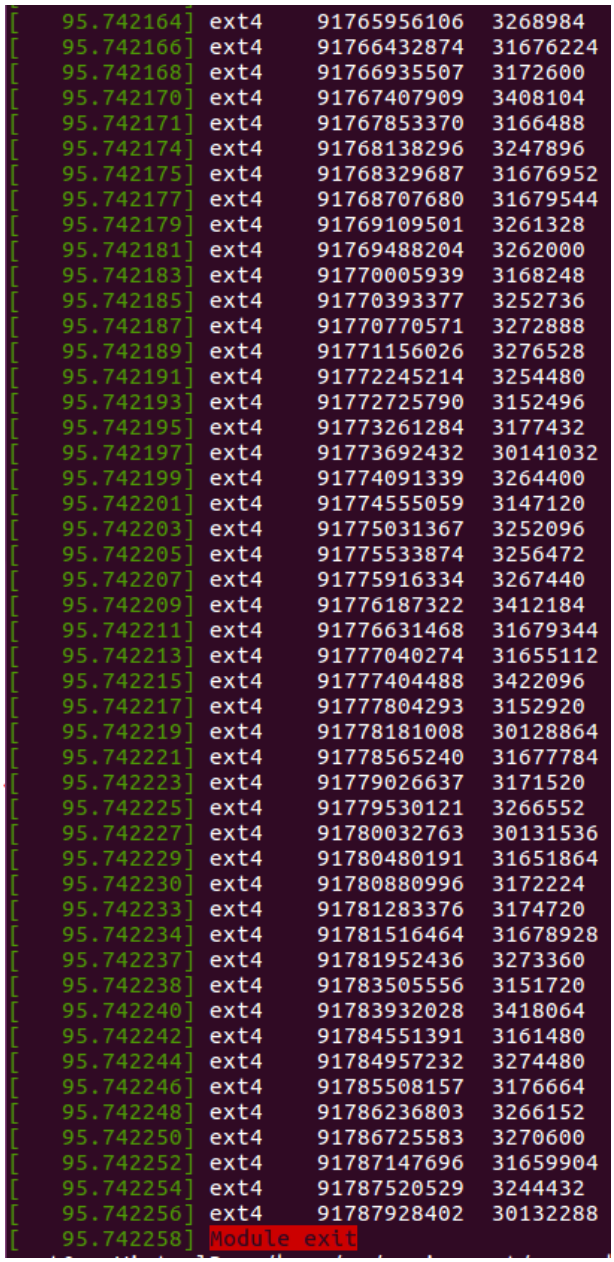
먼저 basic 모듈을 올려서 write작업이 발생할 경우 우리가 만든 circular queue에 fs_name, time, block_number가 들어가게 한다.

그리고 write작업을 발생시키기 위해서 iozone을 실행하는데 iozone -s 16384 -r 4 이 옵션을 줘서 실행했다.

그 다음 우리가 짠 basic 모듈을 내리면 dmesg에 circular queue의 내용이 비워지면서 출력되게 되는데 이 출력된 값들을 dmesg를 통해서 확인했다.

1) Ext4



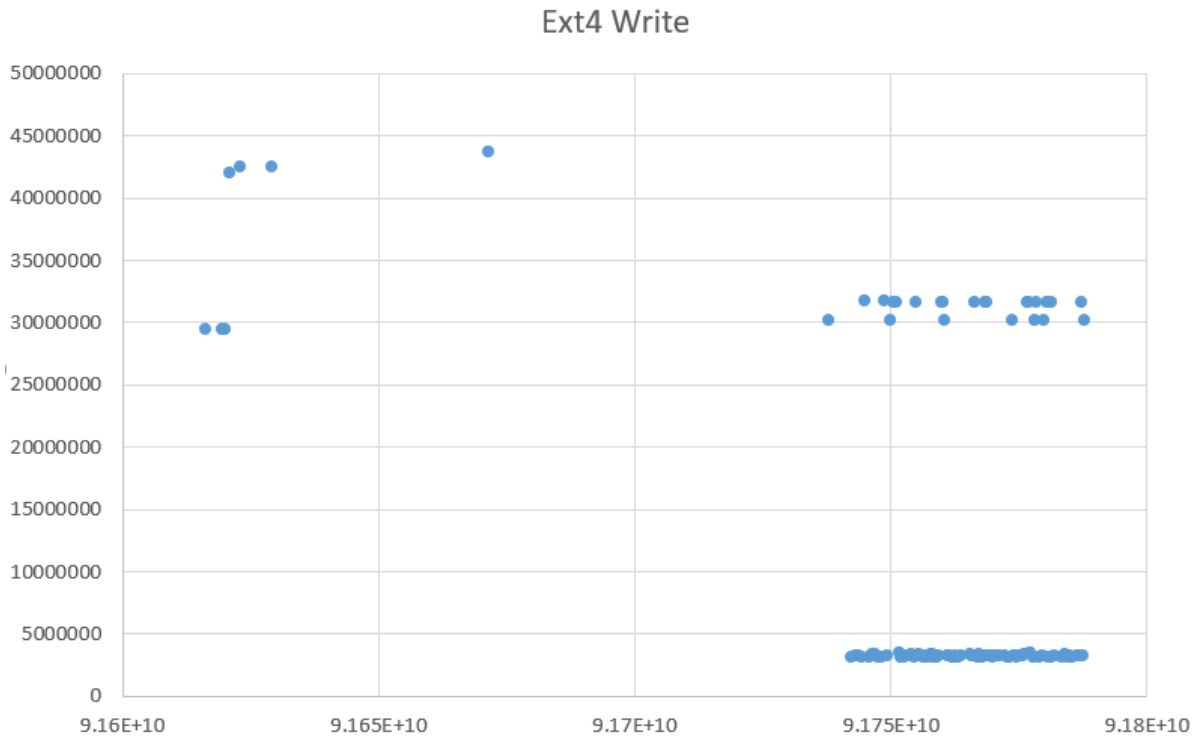


2) Nilfs2

```
[  142.601621] Module init
[  152.270259] 100 data in queue
[  152.270276] ext4     143997088270 31768680
[  152.270280] ext4     144036354713 1329472
[  152.270283] ext4     144036376783 748744
[  152.270287] ext4     144036384326 79840
[  152.270290] nilfs2   144039092491 0
[  152.270293] ext4     144039178535 29629488
[  152.270296] ext4     144039194459 29629496
[  152.270300] ext4     144039198650 29629504
[  152.270303] ext4     144039201164 29629512
[  152.270306] ext4     144039471869 29629520
[  152.270309] ext4     144043866003 0
[  152.270312] ext4     144044767514 16779792
[  152.270315] ext4     144044791539 16784944
[  152.270318] ext4     144044798244 58728424
[  152.270321] ext4     144051621178 0
[  152.270325] nilfs2   144093087901 2480
[  152.270328] nilfs2   144093803355 4528
[  152.270331] nilfs2   144094470758 6576
[  152.270334] nilfs2   144096260651 8624
[  152.270337] nilfs2   144098937248 10672
[  152.270340] nilfs2   144099459102 12720
[  152.270344] nilfs2   144099945476 14768
[  152.270347] nilfs2   144110769198 16384
[  152.270350] nilfs2   144111381846 18432
[  152.270353] nilfs2   144112375268 20480
[  152.270356] nilfs2   144114741491 22528
[  152.270359] nilfs2   144115326202 24576
[  152.270363] nilfs2   144115944158 26624
[  152.270366] nilfs2   144116590329 28672
[  152.270369] nilfs2   144117305225 30720
[  152.270372] nilfs2   144121079447 31536
[  152.270375] nilfs2   144123774483 32768
[  152.270378] nilfs2   144126077569 34816
[  152.270382] nilfs2   144127066521 0
[  152.270384] ext4     144127614077 1329584
[  152.270388] ext4     144127769963 1331632
[  152.270391] ext4     144127921378 1333680
[  152.270394] ext4     144128251588 1335728
[  152.270397] ext4     144128419207 1337776
[  152.270400] ext4     144128564198 1339824
[  152.270403] ext4     144128675944 1341872
[  152.270407] ext4     144128804172 1392640
[  152.270410] ext4     144128941899 1394688
[  152.270413] ext4     144129083258 1396736
[  152.270416] ext4     144129221823 1398784
[  152.270419] ext4     144129398382 1400832
[  152.270422] ext4     144129531360 1402880
[  152.270425] ext4     144129668248 1404928
[  152.270428] ext4     144129795080 1406976
[  152.270432] ext4     144129926941 10321920
[  152.270435] ext4     144130006001 10323968
[  152.270438] ext4     144155471807 12928544
[  152.270441] ext4     144167907466 29629528
[  152.270444] ext4     144167916964 29629536
[  152.270447] ext4     144168105815 29629544
[  152.270451] nilfs2   144253092874 35912
[  152.270454] nilfs2   144253806652 37960
[  152.270457] nilfs2   144254499756 40008
[  152.270460] nilfs2   144255179452 42056
[  152.270463] nilfs2   144255795452 44104
[  152.270466] nilfs2   144256392455 46152
[  152.270470] nilfs2   144258229001 48200
[  152.270472] nilfs2   144271410844 49152
[  152.270476] nilfs2   144272357333 51200
[  152.270479] nilfs2   144272872762 53248
[  152.270482] nilfs2   144273434286 55296
[  152.270485] nilfs2   144273915073 57344
[  152.270488] nilfs2   144274425753 59392
[  152.270491] nilfs2   144277148166 61440
[  152.270495] nilfs2   144277869207 63488
[  152.270498] nilfs2   144278328204 65536
[  152.270501] nilfs2   144279379455 399992
[  152.270504] ext4     144279593448 10325064
[  152.270507] ext4     144279792077 10327112
[  152.270510] ext4     144279915556 10329160
[  152.270513] ext4     144280143798 10331208
[  152.270516] ext4     144280272585 10333256
[  152.270520] ext4     144280391595 10335304
[  152.270523] ext4     144280457804 10337352
[  152.270526] ext4     144280591341 10452992
[  152.270529] ext4     144280713982 10455040
[  152.270532] ext4     144280834109 10457088
[  152.270535] ext4     144280955074 10459136
[  152.270538] ext4     144281075480 10461184
[  152.270541] ext4     144281194490 10463232
[  152.270545] ext4     144281316014 10465280
[  152.270548] ext4     144281438096 10467328
[  152.270551] ext4     144281465195 10289152
[  152.270554] ext4     144370879352 29629552
[  152.270557] ext4     144370893600 29629560
[  152.270560] ext4     144372101016 29629568
[  152.270564] ext4     144397231863 10869368
[  152.270566] ext4     144397414568 29629576
[  152.270570] ext4     144397420155 29629584
[  152.270573] ext4     144397548384 29629592
[  152.270576] nilfs2   144402461299 65776
[  152.270579] nilfs2   144403167255 67824
[  152.270582] nilfs2   144408133528 0
[  152.270585] ext4     144408346404 10289392
[  152.270589] ext4     144408466531 10291440
[  152.270591] Module exit
```

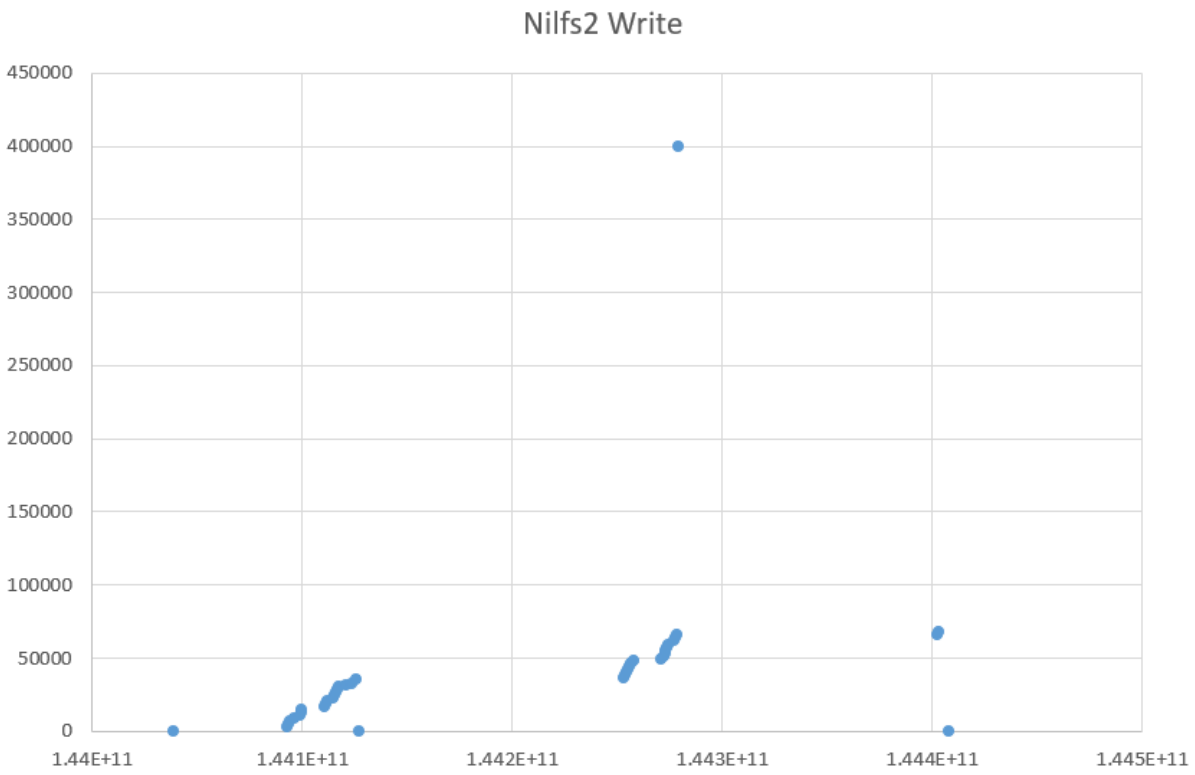
6. 결과 그래프 및 그에 대한 설명

1) Ext4



가로축은 시간이고 세로축은 write하는 Block number이다. Ext4는 write를 할 때 Inode에 블록주소를 할당 받아서 쓰게 되는데 이 때 할당 받는 블록주소는 일정한 규칙성이 없다. 그래프에서도 일정한 규칙성이 없이 이곳저곳에 쓰이는 모습을 확인할 수 있다.

2) Nilfs2



Nilfs2는 LFS의 쓰기방식을 가져와서 write를 할 때 연속된 블록에 write하고 마지막에 Inode를 쓰는 방식으로 진행된다. 그래프에서도 이와 같이 연속적인 블록에 write하는 모습을 확인할 수 있다.

7. 과제 수행 시 어려웠던 부분과 해결 방법

1) 커널코드를 처음 보다보니 circular queue에 들어갈 블록 넘버, 시간, 파일시스템 이름을 찾기가 힘들었다. 우리가 찾는 것 같은 부분들을 다 들어가보고 출력해보면서 해결했다.

2) blk-core.c안에서 우리가 짠 circular queue를 추가하고 수정을 해서 EXPORT_SYMBOL을 했는데 EXPORT가 되지 않았다. 처음 리눅스 설치 ppt를 보니 make만 하면 되는게 아니고 module install과 install을 다 한 뒤에 재부팅 해야 EXPORT된다는 사실을 알았다.

3) 구조체를 어디다 정의해야 할지 몰라서 약간 헤맸다. 새 헤더파일을 만들까 고민해 봤지만, makefile을 수정해야 할 것 같아 못했다. 조교님께 물어봤더니 있는 헤더 파일 중 아무거나 골라서 하라고 하셔서 해결했다.

4) nilfs 테스트에서 첫 번째 테스트는 돌아가는데, 두 번째부터는 디스크 용량이 부족하다고 해서

고생했다. 여전히 왜인지 모르는 상태이다. 그냥 디스크를 지우고 새로 만들고 반복해서 실험을 진행했다.

5) 모듈을 만들 때, 실습에서 한 것처럼, 파일을 만들어서 cat을 통해 queue의 데이터를 읽어오는 방식을 취하고 싶었으나, 만든 파일을 읽어올 때 터미널이 멈추는 현상이 반복해서 일어나서, 그냥 init, exit을 통해 queue의 데이터를 받아오는 방법으로 수정했다.