2019F 시스템 프로그래밍

과제1 : Ext4 와 Nilfs2 의 동작방식 분석

Group 25

2014210042 한윤
2014210055 김민섭

제출일자 2019년 10월 31일
FreeDay 사용일수 0일

목차

1. 각자 맡은 부분, 주요 기여점

한 윤 : 수정해야 할 커널 소스 코드 확인, 커널 소스 코드 수정, 디버깅. 작성한 소스코드에 대한 보고서 내용 작성.

김 민섭 : 수정해야 할 커널 소스 코드 확인, LKM 코드 작성, 디버깅. 작성한 소스코드에 대한 보고서 내용 작성.

2. 개발 환경

Host OS : MAC OS X Catalina 10.15
Virtual Machine : Parallels Desktop 11
Guest OS : Ubuntu 14.04 LTS
C Compiler : gcc
Linux Kernel Version : 4.4.0
하드웨어 스펙: i7 9th gen, 16GB memory, 256GB SSD

3. 배경 지식

3.1 Virtual File System

Virtual File System(VFS)는 실제 파일시스템의 추상 계층이다. VFS는 사용자가 자신이 사용하는 파일시스템의 종류, 갯수의 상관없이 파일시스템을 일관된 방식으로 사용할수있도록 각 파일시스템들에 대한 같은 API를 제공한다. VFS는 디스크기반 파일시스템, 네트워크 파일 시스템, 특수 파일 시스템(proc)등을 지원한다. VFS는 지원하는 모든 파일 시스템을 표현하기위해 공통으로 쓰는 Object를 만들었는데 SuperBlock, Inode, File, Dentry 들이 그 Object에 해당한다. 우선 SuperBlock은 마운트 된 파일 시스템에 대한 정보를 저장하고 있다. Inode는 특정 파일, 디렉토리에 대한 정보를 저장하고 있다. File은 열린 파일에 대한 정보를 저장하고 있으며 각 프로세스가 열린 파일을 가지는 동안 커널 메모리에 존재를 한다. Dentry는 디렉토리 항목에 대한 정보를 가지고 있고 있다.

3.2 NILFS2

NILFS2는 일본의 NTT라는 회사에 개발한 Log Structured File System(LFS)이다. LFS는 기존의 파일 시스템의 seek time과 crash recovery의 비용에 대한 고민으로 나온 새로운 파일시스템이다. 원래는 디스크 기반 파일 시스템이 였으나 플래시 파일 시스템의 기반이 되었다. LFS는 기존의 파일 시스템과 다르게 모든 write request를 디스크에 로그형식으로 순차적으로 기록한다. 따라서 디스크 seek time이 줄어들었고 crash recovery시 파일 제일

뒤에 있는 부분만 확인하면 되어서 성능이 향상되었다. NILFS2는 LFS의 종류중 하나이고 리눅스 커널에 기본적으로 들어가있다.

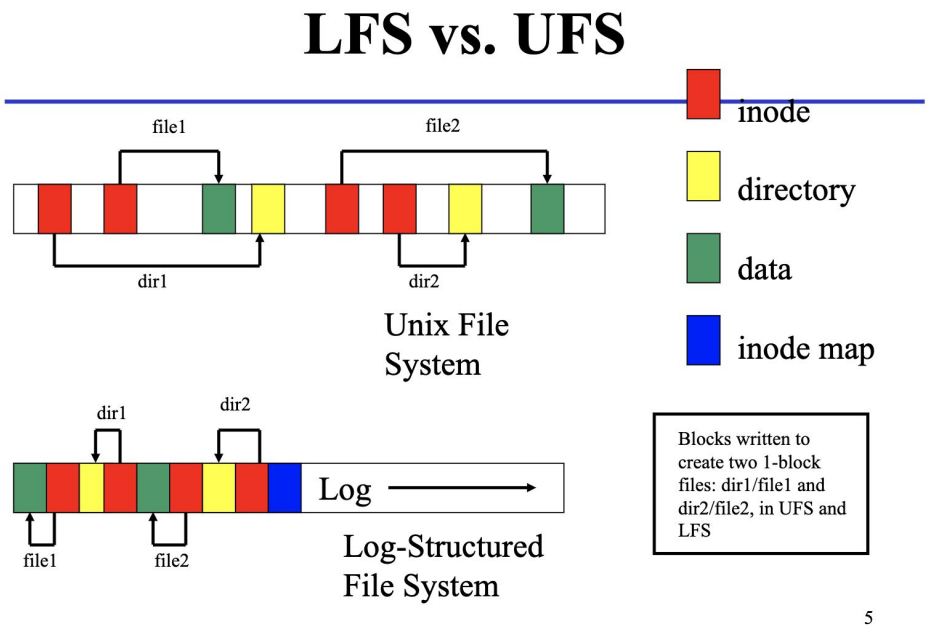

Fig 1 LFS, UFS 비교사진

3.3 EXT4

리눅스에 주로 쓰이는 파일 시스템중 하나이다. 대부분의 리눅스 배포판에서는 기본 파일 시스템으로 사용하고있다. 최대 1EB의 볼륨과 16TB의 파일을 지원한다.

3.4 Loadable kernel Module(LKM)

Loadable Kernel Module(LKM)이란 시스템 실행중에 Loadable한 커널 모듈이라는 뜻이다. 실행중에 load하거나 unload할수있다. 따라서 커널모듈을 수정하거나 해도 시스템을 재부팅할 필요가 없어진다.
이번 과제에서는 LKM으로 proc파일 시스템을 통해서 리눅스 커널 안에 있는 로그 데이터를 읽어올수있는 코드를 작성할 예정이다.

3.5 Mount

리눅스는 usb, cd-drive 등 외부 물리적인 장치 파일 시스템을 디렉토리에 붙이는 작업이다. nilfs2와 같이 다른 파일 시스템안에 있는 파일들도 mount를 통해 local file 인것 처럼 사용할수 있다는 것이 장점이다.
명령어는 mount -o 옵션 디바이스명 마운트할 디렉토리 이다.
이번 과제에서는 nilfs2 파일시스템을 mount하여 사용할 예정이다.

4. 작성한 부분에 대한 설명

커널에서 변경한 부분은 **/iinux/block/blk-core.c** 와 **/linux/fs/nilfs2/segbuf.c** 두 부분이고, proc FS 를 다루는 LKM 코드를 하나를 새로 작성하였다.(**mymodule.c)**
blk-core.c 파일에서는 다음과 같은 로직으로 코드를 작성하였다.

1) 매 block i/o 가 일어날 때의 time, block no, filesystem name 을 기록해놓기 위한 전역 변수를 선언하고 EXPORT_SYMBOL 매크로 함수로 LKM 에서 참조 할 수 있도록 만든다.
2) submit_bio() 함수 내에서 block i/o operation 이 write 인 if 문 안에서 포인터 예외처리를 먼저 해준 후에 i/o 가 일어난 파일 시스템이 ext4인지 nilfs2인지에 따라 각자 다른 전역 변수에 로그를 기록한다.

```
if (rw & WRITE) {                                                    submit_bio  Aa Abl .*  1 of 4
    count_vm_events(PGPGOUT, count);
    /* This code below is to log information into kernel mesg. It includes
     * name of file system,
     * block number,
     * time,
     * for every write op.
     * AND store all this information into kernel memory(global variables)
     */
    if (bio && bio->bi_bdev && bio->bi_bdev->bd_super && bio->bi_bdev->bd_super->s_type) { // exception handling
        if (strcmp(bio->bi_bdev->bd_super->s_type->name, "ext4") == 0) {
            // store write time in seconds and milliseconds
            do_gettimeofday(&time);
            write_t_s[pos] = time.tv_sec;
            write_t_us[pos] = time.tv_usec;
            // store block number
            block_n[pos] = (unsigned long long)bio->bi_iter.bi_sector;
            // store file system name
            fs[pos] = bio->bi_bdev->bd_super->s_type->name;
            // if pos is less than (BUFFER_MAX_LEN - 1), increase position by 1
            // or reset to 0
            if (pos < BUFFER_MAX_LEN - 1) pos += 1;
            else pos = 0;
        } else if (strcmp(bio->bi_bdev->bd_super->s_type->name, "nilfs2") == 0) {
            // store write time in seconds and milliseconds
            do_gettimeofday(&time);
            write_t_s_nilfs2[pos_nilfs2] = time.tv_sec;
            write_t_us_nilfs2[pos_nilfs2] = time.tv_usec;
            // store block number
            block_n_nilfs2[pos_nilfs2] = (unsigned long long)bio->bi_iter.bi_sector;
            // store file system name
            fs_nilfs2[pos_nilfs2] = bio->bi_bdev->bd_super->s_type->name;
            // if pos is less than (BUFFER_MAX_LEN - 1), increase position by 1
            // or reset to 0
            if (pos_nilfs2 < BUFFER_MAX_LEN - 1) pos_nilfs2 += 1;
            else pos_nilfs2 = 0;
        }
```

Fig 2 Submit_bio() 함수 로직

segbuf.c 파일에서는 nilfs_segbuf_submit_bio() 함수 안에서 submit_bio() 를 호출하기 전에 block i/o 의 superblock 의 값을 nilfs의 superblock 으로 치환해주는 코드를 간단히 작성하였다.

```
/*
 * Related to modifying super block before executing submit_bio()
 */
if (bio && bio->bi_bdev) {
    bio->bi_bdev->bd_super = segbuf->sb_super;
}
```

Fig 3 Superblock 치환 코드

mymodule.c 에서는 cat 명령어를 통해, blk-core.c 에서 각 파일시스템별 전역변수에 기록해 두었던 로그를 읽어올 수 있도록 코드를 작성하였다. 이 때 ext4 로그와 nilfs2 로그를 각각 while loop 문을 사용하여 출력하도록 하였다.

5. 실행 방법에 대한 간략한 설명, 실행 결과 캡처 화면

1) linux-4.4 버전의 커널 소스코드에 변경 사항을 적용 후 make && make install 로
커널 재설치를 진행한다. 이후 변경된 커널로 재부팅 한다.

2) 부팅 후 nilfs2 파일 시스템 타입으로 포맷된 가상의 diskfile 을 만들고 loop device (/dev/loop0)에 mount 한다. 이후 해당 loop device 를 홈 디렉토리에 존재하는 적절한 디렉토리에 mount 한다.

3) 만들어둔 LKM 을 컴파일하고 insmod 로 모듈을 적재한다.

4) lsmod 로 모듈이 잘 적재되었는지 확인한 뒤, mount해둔 nilfs2 device 에 iozone 으로 write operation 을 수행한다.

5) **sudo sh -c 'cat /proc/myproc/myproc'** 명령어를 이용해서 모듈이 proc FS의 read callback 함수를 통해 write 테스트가 진행되는 동안 기록된 로그를 콘솔 상에 출력하게 한다.

6) 콘솔 상에 출력된 로그들을 redirection pipe (>) 를 통해 txt 형식의 파일로 내보낸다.

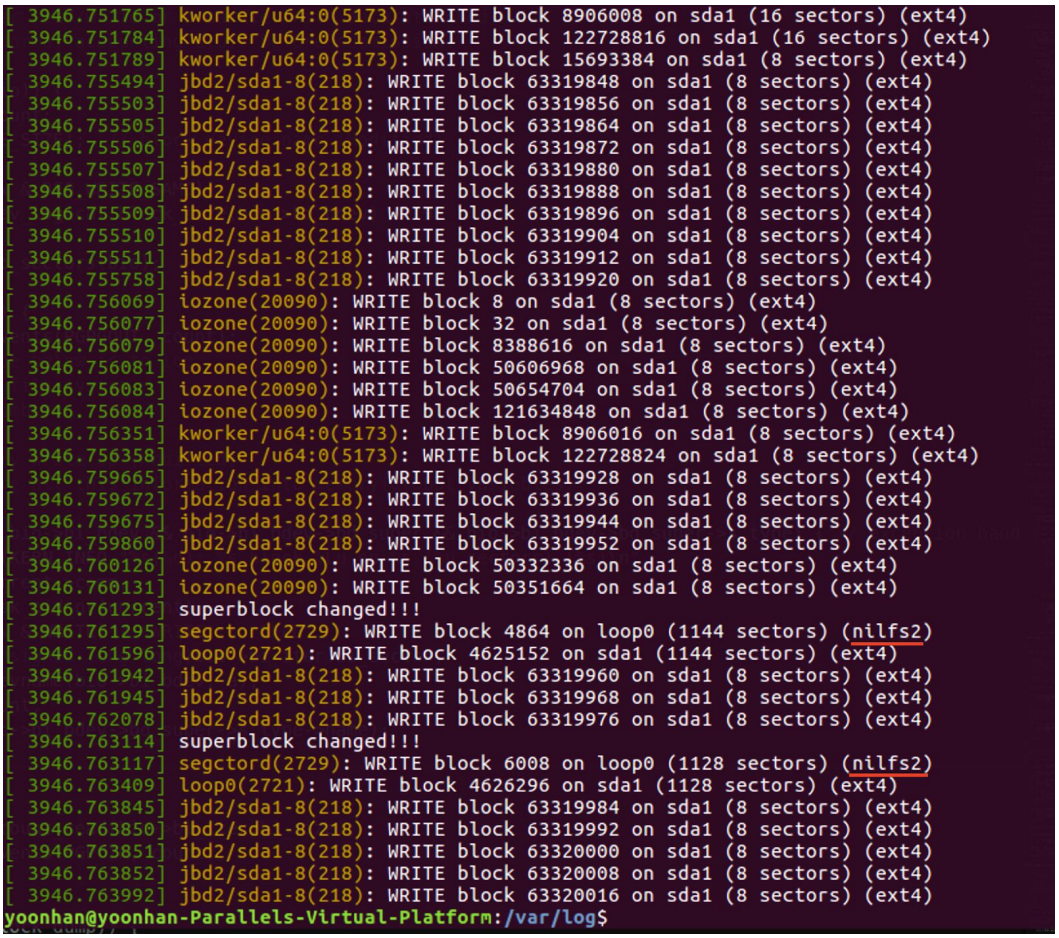

Fig 4 blk-core.c 소스코드를 수정하여 매 block i/o 마다 커널 로그 기록

6. 결과 그래프 및 그에 대한 설명

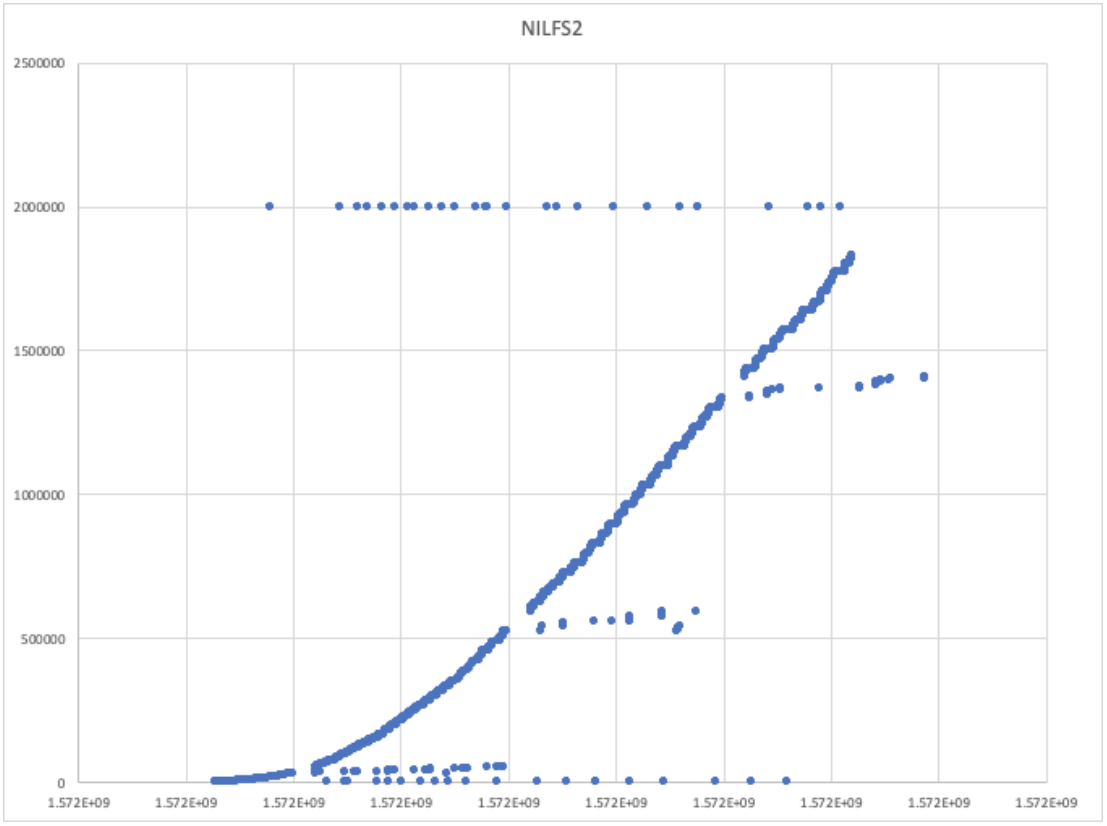

Fig 5 NILFS2 테스트 결과 그래프(x축 time y축 block number)

그래프를 보면 시간이 지날수록 블록넘버도 순처적으로 증가하고 있는거를 볼수있다. 그 이유는 NILFS2가 Log Structured File System(LFS)의 종류중 하나이기 때문이다. 따라서 계속 마지막에 쓰인 블록뒤로 append를 수행하고 있기 때문에 순차적으로 증가하는 그래프가 나왔다. 또 중간중간에 위로 튀거나 옆으로 튄 값들은 LFS가 garbage collecting을 하고 있어서 나타난것으로 보인다.

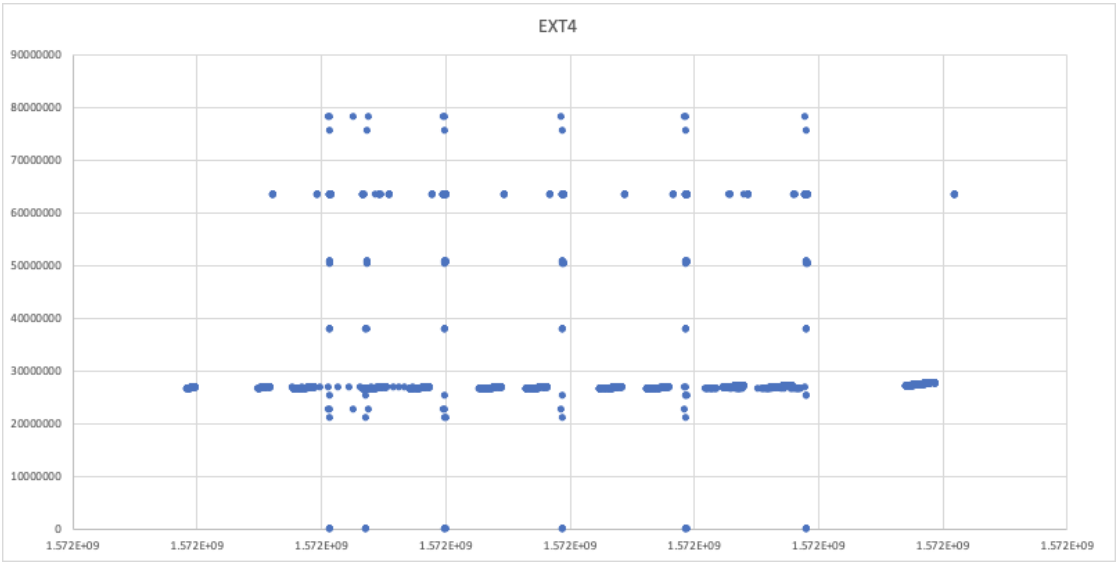

Fig 6 EXT4 테스트 결과 그래프(x축 time y축 block number)

Fast File System(FFS)는 Cylinder라는 그룹을 만들어 그 안에 superblock, inode, file등을 배치한다. 또 같은 디렉토리에 속하는 파일과 inode를 같은 Cylinder그룹에 할당한다. 그리고 LFS와 다르게 super block이 있는 블록과 inode가 있는 블록들이 연속적으로 있지 않고 따로 떨어져있다. 이런 FFS의 특성을 EXT2부터 채택하였고 EXT4는 EXT2를 발전시킨것이기 때문에 EXT4에는 FFS의 특성이 남아있다. 그래서 위 그래프에서 보면 LFS와 다르게 연속적으로 블록을 접근하지않고 따로 떨어진 블록을 계속 접근하는 것을 볼수있다.

7. 과제 수행 시 어려웠던 부분과 해결 방법

- proc file system 에서 proc 파일을 생성하는 예제를 실행하던 도중에,
  sudo echo bbb > /proc/myproc
  를 실행했을 때 permission denied 오류가 발생하였다. 구글링을 해보니 sudo 명령으로 echo 를 실행해도 redirection 명령 자체는 로그인한 계정(사용자 계정)의 권한으로 실행이 된다는 내용을 확인하고
  sudo sh -c "echo bbb > /proc/myproc
  명령어를 통해 해결하였다.

- 코드 디버깅시에 LKM을 올리고 내리는 과정에서 명령어들을 full length 로 타이핑 하는게 매우 번거로웠다. 이 부분은 ~/.bash_profile 파일에 명령어 alias를 등록하여 해결하였다.

- kernel의 printk함수에 나타나는 timestamp는 CPU의 TSC(Time Stamp Counter)를 보고 생성하는데 이 TSC는 CPU가 저전력모드라 클럭이 낮아지거나 ,오버클럭이 되어버리면 신뢰하기 어렵다는 단점이 있었다. 따라서

do_gettimeofday 라는 linux 표준함수를 사용하는 것이 바람직하다.
출처 : https://access.redhat.com/solutions/125273

- 커널을 recompile 하고 다시 install 하는 과정에서 매번 diskfile 을 생성하고 마운트 하기가 번거로워서 alias 에 **mkmtdf** 라는 명령어 하나로 지루한 타이핑 과정을 없애버렸다. iozone으로 쓰기 성능 테스트 하는 명령어도 길기에 등록해 놓았다.

- dmesg 명령어가 출력할 수 있는 최대 로그의 수가 한정되어 있었다(대략 한 번에 4천개 정도가 출력되는듯 하다). 그래서 ext4 block i/o 와 nilfs2 block i/o 로그데이터를 한 번에 뽑아내지 않고 각자 뽑아내었다.