Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 전문가 과정

<리눅스 커널>

2018.04.11 - 35 일차

강사 - 이상훈

gcccompil3r@gmail.com

학생 - 안상재

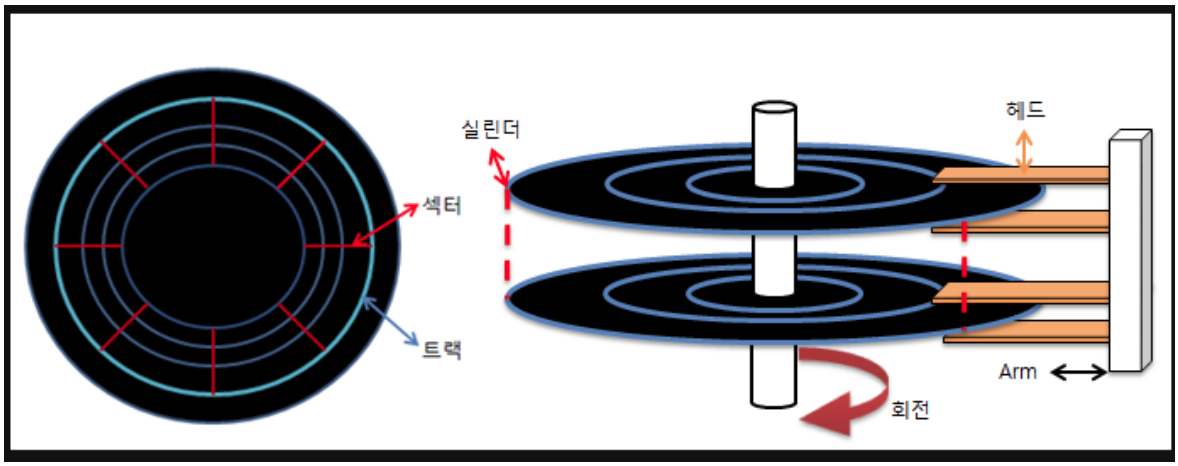
sangjae2015@naver.com

<Chapter 5 파일 시스템과 가상 파일 시스템>

1. 파일시스템 일반
1) 파일 시스템
- 보조 기억 장치라는 저장장치를 관리하는 소프트웨어
- '이름'을 입력으로 받아 해당 데이터를 리턴해 주는 소프트웨어

2) 파일시스템이 하드디스크에 저장하는 내용
- 메타 데이터 : 파일의 속성 정보, 데이터 블록 인덱스 정보. ex) inode, 수퍼블록
- 사용자 데이터 : 사용자가 실제 기록하려 했던 내용

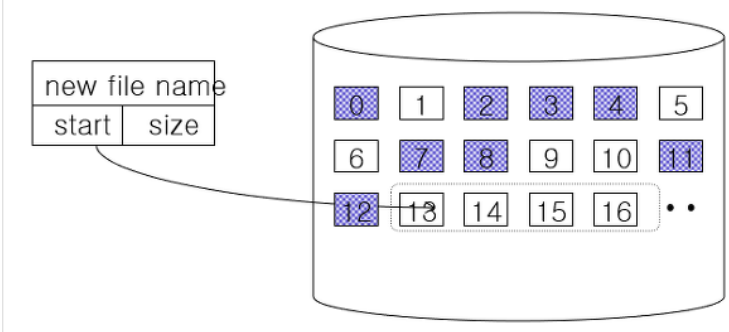
2. 디스크 구조와 블록 관리 기법



1) 디스크 구조
- 실린더 : 같은 위치를 갖는 트랙들의 집합
- 섹터 : 섹터는 디스크에서 데이터를 읽거나 기록할 때 기본 단위가 되며, 일반적으로 섹터의 크기는 512Byte 이다.
- 헤드 : 각 원판의 읽기/쓰기가 가능한 면마다 하나씩 존재함.

2) 디스크에서 데이터를 접근하는 데 걸리는 시간
- 탐색 시간 : 헤드를 요청한 데이터가 존재하는 트랙 위치까지 이동하는 데 걸리는 시간
- 회전 지연 시간 : 디스크의 원판 구동 시간
- 데이터 전송 시간

3) 파일 시스템은 디스크를 물리적인 구조로 보지 않고 논리적인 디스크 블록들의 집합으로 본다. 디스크 블록의 크기는 일반적으로 페이지 프레임의 크기와 같다.(4KB)



4) 파일 시스템 성능의 최대 병목 요소는 디스크 I/O 이다.
- 디스크 블록의 크기와 성능은 비례, 공간효율성과는 반비례 한다.

5) 디스크 블록의 할당과 회수 방법
- 연속 할당
- 불연속 할당

* 불연속 할당
<1> 블록 체인 기법
- 같은 파일에 속한 디스크 블록들을 체인(연결리스트) 으로 연결함.
- 중간의 한 블록이 유실된 경우 나머지 데이터까지 모두 잃게 된다는 단점이 존재.

<2> 인덱스 블록 기법
- 블록들에 대한 위치 정보들을 기록한 인덱스 블록을 따로 사용함.
- 인덱스 블록이 유실되면 파일의 데이터 전체가 소실되는 문제가 존재.

<3> FAT 기법
- 파일 시스템이 관리하는 공간 내에 전역적으로 존재하는 FAT 구조를 사용하여 파일에 속해있는 데이터를 찾아가는 방법.
- 파일 시스템 전체적으로 하나의 FAT 가 존재함.
- FAT 구조의 유실은 파일 시스템 내의 모든 파일의 소실을 의미함.

3. FAT 파일 시스템
* 파일 시스템이 관리하는 데이터는 크게 메타 데이터와 유저 데이터로 구분됨.
- 메타 데이터 : 데이터에 대한 구조화된 데이터 (FAT 테이블, 디렉토리 엔트리, 수퍼 블록)
- 유저 데이터 : 파일의 내용

1) msdos 파일시스템의 디렉토리 엔트리
- msdos 파일시스템에서는 각 파일마다 디렉토리 엔트리를 하나 씩 가지고 있음.
- FAT 시스템에서 readdir() 함수를 통해 디렉토리 엔트리를 가져옴.

```
struct msdos_dir_entry{
__u8 name[MSDOS_NAME];
__u8 attr;
__u8 lcase;
__u8 ctime_cs;
__le16 ctime;
__le16 cdate;
__le16 adate;
__le16 starthi;
__le16 time, date, start;
__le32 size;
}
```

2) 파일 시스템에서 사용자가 요청한 파일 찾기
→ 사용자가 요청한 파일의 이름을 가지고 디렉토리 엔트리를 찾아냄.
→ 수퍼블록에서 '/' 을 찾음
→ 디렉토리 엔트리 정보를 이용해 데이터 블록을 찾아서 사용자에게 제공해줌.

* 파일 시스템이 디렉토리 엔트리 찾는 법
- 상대 경로 : 현재 디렉토리 위치를 기준으로 시작됨.
- 절대 경로 : '/' 에서부터 시작됨.
- 상대, 절대 경로 모두 디렉토리 엔트리를 가지고 찾게됨.

4. inode 구조

```
struct ext2_inode{
__le16  i_mode;   // 해당 inode가 관리하는 파일의 속성 및 접근 제어 정보
__le16 i_links_count // inode를 가리키고 있는 파일 수
……
}
```

* inode 는 12 개의 direct block 과 3 개의 indirect block 을 i_block[15] 배열을 통해 저장하고 있다.
- direct block : 실제 파일의 내용을 담고 있는 디스크의 데이터 블록을 가리키는 포인터.
디스크 블록의 크기가 4KB 이므로, 직접 블록으로 지원할 수 있는 파일의 크기는 48KB 이다.

- indirect block : 인덱스 블록(디스크 블록을 가리키는 포인터들을 갖는 블록)을 가리키는 포인터
단일 간접 블록 : 1024 * 4KB = 4MB
이중 간접 블록 : 1024 * 1024 * 4KB = 4GB
삼중 간접 블록 : 1024 * 1024 * 1024 * 4KB = 4TB

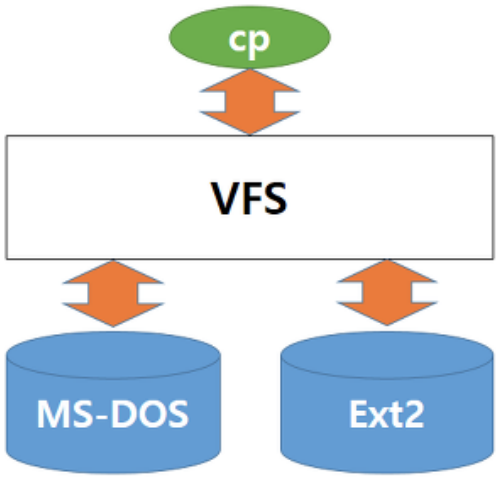
=> 디스크 블록의 크기가 4KB 이고 인덱스의 각 포인터 크기가 4byte(32bit cpu) 인 시스템 환경에서 inode 구조가 지원할 수 있는 파일의 최대 크기는 48KB+4MB+4GB+4TB 가 된다.

- Ext2 파일 시스템의 디렉토리 엔트리

```
struct ext2_dir_entry {
__le32  inode;          /* Inode number */
__le16  rec_len;        /* Directory entry length */
__le16  name_len;       /* Name length */
char    name[];         /* File name, up to EXT2_NAME_LEN */
};
```

5. Ext2 파일 시스템
- IDE 방식의 디스크 : 각 디스크는 /dev 디렉토리에 "hd"라는 이름의 블록 장치 파일로 접근됨.
- SCSI 방식의 디스크 : "sd" 라는 이름을 접근됨.

7. 가상 파일 시스템
* 수퍼 블록을 읽으면 메타 데이터가 읽혀서 어떤 파일 시스템인지 알게됨.
- 각 파일 시스템마다 구현된 함수들이 다르기 때문에, 어느 파일 시스템이건 사용자 태스크에서 일관된 함수를 사용해서 파일 연산을 할 수 있도록 해야됨.



- 위의 그림처럼 한 디스크의 여러 파티션에 따라 파일 시스템이 다르더라도 사용자 태스크 입장에서는 일관된 파일 연산 함수를 사용한다. 그렇게 되면 VFS(가상 파일 시스템) 에서 유저에서 호출하는 함수의 인자에 담겨있는 파일이름을 보고 해당 파일 시스템이 어떤 것인지 판단하고 사용자가 원하는 일을 해줄 수 있는 파일 시스템 고유의 함수를 호출해준다.

1) 리눅스의 가상 파일 시스템이 사용자 태스크에게 제공할 일관된 인터페이스
<1> 수퍼 블록 객체 : 파일 시스템 당 하나씩 주어지고 파일 시스템의 종류 및 고유한 정보를 저장함.
<2> inode 객체 : 특정 파일과 관련된 정보를 담기 위한 구조체
<3> 파일 객체 : 태스크가 open 한 파일과 연관되어 있는 정보를 관리함.
<4> 디엔트리 객체 : 태스크가 파일에 접근하려면 해당 파일의 inode 객체를 자신의 태스크와 연관된 객체인 파일 객체에 연결 시켜야함. 이 관계를 조근 더 빠르게 연결하기 위한 일종의 캐시 역할을 하는 것이 디엔트리 객체임.

* 가상 파일 시스템이 각기 다른 파일 시스템을 동일한 함수로 사용자 태스크에게 지원할 수 있는 이유는 각 파일 시스템 마다 적절한 함수 포인터들을 정의해 놓았기 때문이다.