Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com


CONTENT

파일시스템 과 가상 파일시스템

1. 파일시스템 일반

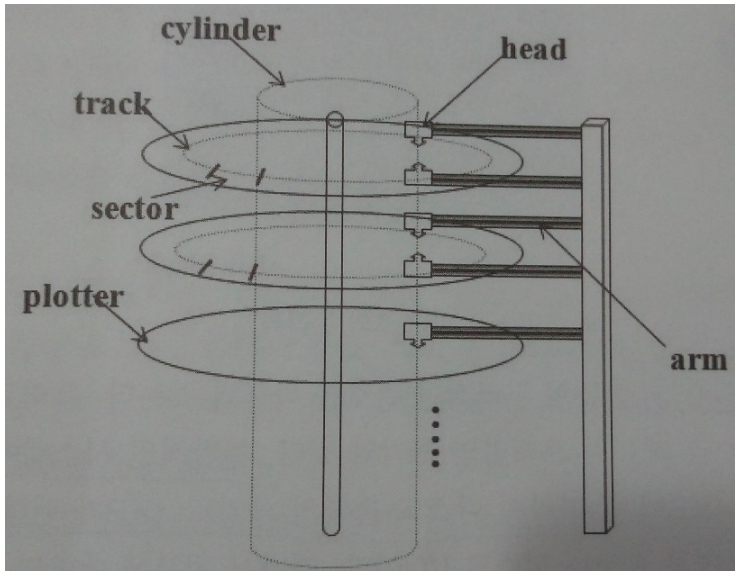
메모리관리에서 buddy 와 slab (외부 및 내부 단편화를 최소화시키는 알고리즘, 단편화를 없애기 위한 알고리즘)이 있었다.
마찬가지로 디스크도 파일이므로 메모리에 올라간다. 그러므로 페이지프레임 단위로 관리한다. 디스크도 할당하려다보면 외부 및 내부 단편화 문제가 발생한다. 이것을 효율적으로 관리할 수 있는 메커니즘이 필요하다.
→ 이는 가상파일시스템이 커널과 함께 관리한다. 우리가 할 것은 파일시스템에서는 어떠한 식으로 단편화들을 관리하는지 알아야한다. 추가적으로 메모리와 다른 점이 있다. 메모리는 그냥 메모리이다. 파일시스템이라는 것은 종류 자체가 여러개인데, 이것에 대한 해결책도 제시해야한다.

메모리 관리 기법과 **파일시스템** 간의 **차이점**은 무엇일까? 파일시스템은 사용자에게 이름이라는 속성으로 접근되는 추상적인 객체인 '파일'이라는 개념을 제공한다.
메모리에는 이름이 없었다. 파일시스템에서는 만약 파일이나 디렉토리 같은 것을 찾으려고 할 때, 파일 **이름**으로 찾는 것이 사용자에게 편하다. 추가적으로 가상 파일 시스템이라는 것이 붙어야 하는데, 이는 파일 포맷이 여러 개이기 때문이다.
이름을 입력 받아 해당 데이터를 리턴 해주는 소프트웨어가 바로 파일 시스템이다. 파일 시스템이 디스크에 저장하는 정보는 크게 메타 데이터와, 유저 데이터로 나뉜다.

메타데이터 – inode, super blocking (파일이름, 파일 생성 시간, 실제 데이터블록을 인덱싱 하기 위한 정보 등)
유저데이터 - 메타데이터에 실제로 들어있는 데이터

예) 사용자가 파일을 1개 생성하고 "12월 9일 토요일 오후 3시" 라는 문자열인 약속시간을 저장하려 한다고 가정한다. 기록을 한 후, 나중에 사용자가 약속이 언제였는지 확인 하려한다. 어떻게 할까?
→ 파일 시스템은 disk block 하나를 할당 받는다. (실제 하드디스크에서 4KB 를 할당 받는다 → 이는 **디바이스 드라이버**가 할당해준다)
→ '약속.txt' 로 불러온다. (**파일디스크립터**는 추상적인 자원인 '파일'로 제공해주기 위해 **부가적인 정보(메타데이터)** 을 기록해 두어야 한다.

2. 디스크 구조와 블록관리 기법 -(메타 데이터의 인덱싱과 관련된 내용)



디스크 구조- 원판 , 팔 , 헤드

plotter - 플래터(platter)는 접시라는 뜻. 데이터를 담는 둥근 판으로 하드디스크 속의 디스크이다.
Head - 헤드(head)는 플래터에 기록한 데이터를 읽는 장치, 전자석으로 되어있다. 미세한 자장을 읽고 씀

원판에 원모양의 **트랙**(track)이 존재하며, 모든 원판에서 같은 위치를 갖는 트랙들의 집합을 **실린더**(cylinder)라고 한다. 트랙은 다시 몇 개의 **섹터**(sector)로 구분된다. 섹터는 디스크에서 데이터를 읽거나 기록할때 기본 단위가 되며, 일반적으로 섹터의 크기는 512byte이다. 한편 헤드는 각 원판의 읽기/쓰기가 가능한 면마다 하나씩 존재한다. → 따라서 헤드, 트랙이(또는 실린더), 섹터 각각의 개수가 결정되면 디스크의 전체 용량 등 해당 디스크의 물리적 특성을 결정 할 수 있게 된다.
(디바이스 드라이버(dv)가 이 정보들을 가지고 있다. 하드디스크 dv는 이 정보를 가지고, 서보모터를 제어하여 arm을 돌리고 head를 찍는다. 이 뿐만아니라 디스크 헤드가 찍히는 순간 자기장 패턴을 읽는다. 이는 또 연산과정이 필요하다. 연산과정을 불러내는 것 또한 하드디스크 디바이스 드라이버의 역할이다.)

→ 디바이스 드라이버 개발자들이 만들어놓은 것처럼(디바이스 드라이버가 하드디스크 자체를 구동시키는 것을 제어하려면 계산을 해야함)
우리도 나중에 모터제어를 할 때, pwm신호만 넣는 것이 아니라 계산을 하여 속도와 위치 제어를 해야한다.

*2.1 디스크에서 데이터를 접근하는데 걸리는 시간

2.1.1. 탐색시간 - 헤드를 요청한 데이터가 존재하는 트랙 위치까지 이동하는데 걸리는 시간
2.1.2. 회전 시간 - 요청한 섹터가 헤드 아래로 위치될 때까지 디스크 원판을 회전시키는 데 걸리는 시간
2.1.3. 데이터 전송 시간 - 헤드가 섹터의 내용을 읽거나 또는 기록하는데 걸리는 시간이다. (전기적 신호기때문에 제일 빠름)

파일 시스템은 디스크를 물리적인 구조로 보지 않고, 논리적인 디스크 블록들의 집합으로 본다. 디스크 블록의 크기는 페이지 프레임의 크기(4KB)와 같다.
→ 디스크 블록의 크기와 성능은 비례하며, 공간효율성과는 반비례한다.

*2.2 디스크 블록의 할당 - 공간측면과 속도측면에서 다뤄야한다. 연속할당과 불연속할당으로

2.2.1. 블록체인 기법
같은 파일에 속한 디스크 블록들을 연결리스트로 연결함. 단점은 중간의 한 블록이 유실된 경우 나머지 데이터까지 모두 잃게 된다.

2.2.2. 인덱스블록 기법
블록들에 대한 위치 정보들을 기록한 인덱스 블록을 따로 사용한다. 단점은 인덱스 블록이 유실되면 파일의 데이터 전체가 소실된다.

2.2.3. fat 구조
파일 시스템이 관리하는 공간 내에 전역적으로 존재하는 FAT 구조를 사용하여 파일에 속해있는 데이터를 찾아가는 방법. 단점은 FAT 구조의 유실은 파일 시스템내의 모든파일의 소실을 의미한다.

3. Fat 파일시스템

- 파일 시스템이 관리하는 공간 내에 전역적으로 존재하는 **FAT구조**를 사용하여 파일에 속해있는 **포인터**를 찾아가는 방법

3.1 msdos 파일 시스템의 디렉토리 엔트리

msdos 파일 시스템은 각 파일마다 디렉토리 엔트리를 1개씩 가지고 있고, FAT시스템에서 readdir()함수를 통해 디렉토리 엔트리를 가져온다.

```
struct msdos_dir_entry {
        __u8    name[MSDOS_NAME];/* name and extension */
        __u8    attr;           /* attribute bits */
        __u8    lcase;          /* Case for base and extension */
        __u8    ctime_cs;       /* Creation time, centiseconds (0-199) */
        __le16  ctime;          /* Creation time */
        __le16  cdate;          /* Creation date */
        __le16  adate;          /* Last access date */
        __le16  starthi;        /* High 16 bits of cluster in FAT32 */
        __le16  time,date,start;/* time, date and first cluster */
        __le32  size;           /* file size (in bytes) */
};
```

3.2 파일 시스템에서 사용자가 요청한 파일 찾기

사용자가 요청한 파일의 이름을 가지고 디렉토리 엔트리를 찾는다. 수퍼블록에서 '/'을 찾은다. 디렉토리 엔트리 정보를 이용해 데이터 블록을 찾아서 사용자에게 제공한다.

* 파일 시스템이 디렉토리 엔트리 찾는 법
- 상대경로 : 현재 디렉토리위치가 기준
- 절대경로 : '/' 에서부터 시작된다 .
- 상대, 절대 경로 모두 디렉토리엔트리를 가지고 찾게 된다.

4. inode 구조

```
struct ext2_inode {
        __le16  i_mode;         /* File mode */
        __le16  i_uid;          /* Low 16 bits of Owner Uid */
        __le32  i_size;         /* Size in bytes */
        __le32  i_atime;        /* Access time */
        __le32  i_ctime;        /* Creation time */
        __le32  i_mtime;        /* Modification time */
        __le32  i_dtime;        /* Deletion Time */
        __le16  i_gid;          /* Low 16 bits of Group Id */
        __le16  i_links_count;  /* Links count */
        __le32  i_blocks;       /* Blocks count */
        __le32  i_flags;        /* File flags */
        union {
```

“생략 ‘’

: inode는 12개의 **directblock**, 3개의 **indirect block**을 i_block[15]에 저장한다.

4.1 directblock - 실제 파일의 내용을 담고 있는 디스크의 데이터 블록을 가리키는 포인터.
(디스크 블록의 크기가 4KB이므로, 직접 블록으로 지원할 수 있는 파일의 크기는 48KB임)
→ 디스크에서 direct라는 것은 1:1 맵핑이다. 이렇게 맵핑하면 속도가 많이 느려지게 된다.

4.2 indirect block - 인덱스 블록(디스크 블록을 가리키는 포인터들을 갖는 블록)을 가르키는 포인터.
단일 간접블록: 1024 *4KB = 4MB, 이중 : 1024 * 1024* 4KB = 4GB, 삼중: 4TB (이 정도의 데이터 크기를 가르킬수 있다)

→ 디스크 블록의 크기가 4KB이고, 인덱스의 각 포인터 크기가 4Byte인 32bit인 시스템 환경에서 inode
조가 지원할 수 있는 파일의 최대 크기는 48KB+4MB+4GB+4TB (ext2) 이다.

Quiz. 문제 100기가 바이트이다 이녀석의 마지막 블록 위치는??

vi -t ext2_inodb
vi -t ext4_inode
vi -t msdos_dir_entry

파일 시스템은 디렉터리 엔트리를 어떻게 찾을까?
→ 슈퍼블록 - 디렉터리 하위에 존재하는 다른 모든 파일을 찾을 수 있게된다. (루트파일 시스템이 파일찾기)

Vi -t ext2_sb_info

저널링기법

5. Ext2 파일시스템

IDE 방식의 디스크 : 각 디스크는 /dev 디렉토리에 “hd” 라는 이름의 블록 장치 파일로 접근됨.
SCSI 방식의 디스크: “sd”라는 이름으로 접근됨
→ 못찾겠음..

df-h
mount

p 145 루트파일시스템 -3,start

루트파일시스템은 무조건 2번에 물린다

6. Ext3 파일 시스템과 Ext4 파일시스템 - 구글땨매만들어짐 한번읽어보도록

7. 가상파일시스템 - 파일시스템에서 가장중요 p149~153

수퍼블록을 읽으면 메타 데이터가 읽히기 때문에 어떤 파일 시스템인지 알 수 있다.
- 각 파일 시스템마다 구현된 함수들이 다르기 때문에, 어느 파일 시스템이건 사용자 태스크에서 일관된 함수를 사용해서 파일 연산을 할 수 있도록 해야한다.

일관된 사용하기 때문에 VFS(가상 파일 시스템) 은 유저에서 호출하는 함수의 인자에 담겨있는 파일이름을 보고, 해당 파일 시스템이 어떤것인지 판단하고, 사용자가 원하는 일을 해 줄 수 있는 파일 시스템 고유의 함수를 호출해준다.

File_opration

7.1 리눅스의 가상파일 시스템이 사용자 태스크에게 제공할 일관된 인터페이스

7.1.1 수퍼블록 객체: 파일 시스템 당 하나씩 주어지고 파일 시스템의 종류 및 고유한 정보를 저장함.
→
7.1.2 inode 객체: 특정 파일과 관련된 정보를 담기 위한 구조체임.
→ 파일을 읽으려면 디렉토리를 찾아야한다. 디렉토리를 찾으려면 디렉토리 포인터를 선언하고, 디렉토리 엔트리 읽으면서 찾아야한다.
7.1.3 파일 객체: 태스크가 open한 파일과 연관되어있는 정보를 관리함.
→ 파일디스크립터 (파일구조체포인터배열에 인덱스) 0번,1,2, 표준입력출력에러 , 우린 파일을 3번부터 시작했다.파일위치는 태스크마다 다르다. 시스템프로그래밍 첫번째인날 오픈을 두개 서로 다르게 해놓고 같은 것을 읽으면 각각 별개로 나왔었다.
7.1.4 디엔트리 객체 : 어떤파일찾으려고할때 내가 어디있는지 상대경로, 루트의 위치를 기준으로는 절대경로