TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-04-11 (35 회차)

강사: Innova Lee(이상훈)
gcccompil3r@gmail.com
학생: 정유경
ucong@naver.com

[Ch 5. 파일 시스템과 가상파일시스템]

CPU 의 추상화 → 태스크
디스크의 추상화 → 파일

1. 파일 시스템 일반
메모리관리기법: 주기억장치 를 관리하는 SW → 페이지프레임
파일시스템: 보조기억장치를 관리하는 SW → 디스크 블록
메모리관리 기법과 파일시스템 모두 내부/외부단편화를 최소화하기 위해 노력

파일시스템이 하드디스크에 저장하는 정보
메타 데이터(이름, 파일정보, 인덱싱정보), 사용자 데이터

2, 디스크 구조와 블록 관리 기법
파일시스템이 저장공간의 기본 접근단위인 디스크 블록을 관리하는 기법

하드디스크: 원판, 암, 헤드로 구성
원판에서 같은 위치를 갖는 트랙의 집합 → 실린더
트랙은 몇개의 섹터로 구분됨(일반적으로 512byte)

파일시스템은 디스크를 물리적인 구조로 보지 않고 논리적인 디스크블록들의 집합으로 본다.

디스크블록은 논리적인 번호를 갖는다. 0,1,2…
페이지 프레임의 크기와 같이 디스크 블록의 크기도 4KB
메모리 hierarchy 에서 디스크가 가장 느리다
따라서 디스크 블록의 크기를 더 크게 설정하여 한번의 디스크 입출력으로 더 많은 데이터를 메모리로 읽어들일수 있도록 한다.
디스크 블록의 크기가 클수록 속도는 증가하지만 공간효율성은 감소한다.

디스크블록의 크기가 4KB 이고, 섹터크기가 512byte 일때 하나의 디스크 블록에 8 개의 섹터가 대응된다. (섹터는 데이터를 디스크에서 읽고 쓰는 기본단위로 일반적으로 512byte 이다)
디스크 블록의 번호를 이에 대응하는 섹터들로 매핑시키는 일은 디바이스드라이버가 담당한다.
*. 디바이스 드라이버: 하드웨어, 장치를 제어하기위한 커널의 일부로 동작하는 프로그램

디스크블록의 할당과 회수방법
파일시스템은 자신이 관리하고 있는 공간(파티션)내의 free 상태의 블록을 관리하고 있다가 파일생성요청이 들어오면 이 파일을 위한 공간을 할당한다.
파일크기가 14KB 라면 4 개의 디스크블록이 필요하다.

디스크블록을 할당하는 방법은 불연속할당과 연속 할당의 두가지 방법이 있다
1) 1, 5, 6, 9
불연속할당→ 해당정보가 어디에 흩어져 있는지 위치정보를 따로 저장/관리, 파일의 크기가 커져도 할당이 쉽다. 블록체인기법, 인덱스블록기법, FAT(File Allocation Table)
2) 13, 14, 15, 16
연속할당→ 디스크 블록이 일렬로 인접한 공간에 배치되어 탐색시간이 감소, 파일 읽는 속도 빠르다
, 파일의 크기가 변하면 문제

불연속할당기법
1) 블록체인기법: 같은 파일에 속한 디스크 블록들을 체인(각 블록에 포인터를 둠)으로 다음 블록의 위치를 기록하여 linked list 처럼 연결해 놓는 방법

원하는 정보를 검색할때 (lseek())처음부터 읽어가야 한다
2) 인덱스 블록기법
위치정보를 인덱스 블록이라는 별도의 공간에 블록을 두고 관리
파일마다 인덱스 블록을 가짐
검색속도가 빠름
인덱스 블록을 잃어버리면 해당 파일을 사용할 수 없다
파일의 수가 늘어날수록 인덱스 블록을 저장하는 공간이 많이 소비됨

3) FAT = 블록체인+인덱스블록
파일시스템마다 위치정보를 FAT 라는 자료구조에 저장
FAT 정보를 유실하면 손해가 크므로 중복으로 관리
인덱스 블록 기법과 마찬가지로 처음부터 데이터를 읽어갈 필요가 없어 속도가 빠름

3. FAT 파일시스템

FAT 파일 시스템 메타 데이터: FAT 테이블, 디렉토리 엔트리, 수퍼블록

파일시스템의 디렉터리 엔트리 개념은 모두 비슷하다
→ 자신만의 디렉터리 구조체 선언하고 생성될 때, 구조체에 정보를 적어서 디스크에 저장
→ 사용자가 요청한 파일 이름을 가지고 디렉터리 엔트리를 찾아서, 디렉터리 엔트리가 가지고 있는 정보를 통해 데이터 블록을 찾아서 사용자에게 제공

디렉터리 엔트리 찾기
→ 파일의 이름을 통해 데이터 접근
→ 파일의 상대 경로를 절대 경로로 변환
→ '/'의 디렉터리 엔트리 찾아서 디렉터리 블록에서 다음 경로 이름을 검색

root 의 디렉터리 엔트리 찾기
→ 수퍼 블록(super block)은 파일시스템의 전체적인 정보와 ‘/’의 위치를 기억

4. inode 구조 ← 리눅스 디폴트 파일 시스템 ext4 가 채택

inode 의 필드
1) i_mode: 파일의 속성 및 접근 제어 정보, 16 비트로 구성
2) i_blocks[15]: 파일에 속한 디스크 블록 위치를 관리
12 개 direct block → 디스크의 데이터 블록을 가리키는 포인터
3 개 indirect block → 인덱스 블록(디스크 블록을 가리키는 포인터를 갖는 블록)을 가리키는 포인터
, 간접블록은 단일 간접 블록, 이중 간접 블록, 삼중 간접 블록 세가지로 구성

한개의 블록이 4KB(4096byte) / 4byte → 1024 개 포인터
한블록이 1024 개의 포인터를 표현할 수 있다
한개의 간접블록으로 표현할 수 있는 한 파일의 최대 크기는 4KB(데이터 블록 한개)*1024 개(의포인터) = 4096KB(4MB)

최종적으로 inode 구조에서 지원할 수 있는 최대 파일의 사이즈는
직접 + 간접+이중간접+삼중간접 = 48KB+4MB+4GB+4TB

Ex2 의 디렉터리 엔트리
inode 를 접근하면 파일의 속성정보와 파일에 속한 디스크 블록 위치 파악
파일의 이름과 inode 는 디렉터리 엔트리를 이용해 연결

inode 번호는 파일 시스템이 관리하는 공간의 앞부분에 위치하고 있는 inode 테이블에서의 번호

실제 리눅스 지원 파일 크기는 4GB
file 자료 구조에 파일의 현재 쓰거나 읽을 위치를 나타내는 파일 오프셋(f_pos) 변수는 32 비트
파일을 쓰거나 읽을 수 있는 위치가 가질 수 있는 값은 2 의 32 승, 4GB
최근에 나온 Ext4 는 제한을 해결하여 더 큰 크기의 파일을 지원

5. Ext2 파일시스템

IDE 방식 디스크 /dev 디렉터리에 'hd' 이름의 블록 장치 파일로 접근
SCSI 방식 디스크는 'sd' 이름의 블록 장치 파일로 접근
현재 리눅스에서 디스크마다 최대 64 개까지 파티션 분할 가능
파일시스템은 파티션 당 하나
파티션은 여러개의 블록 그룹(Block Group)으로 나뉨

Ext2 파일 시스템 구조
1) 부트 블록 : 부트스트랩 코드
2) 여러개의 블록 그룹 : 수퍼 블록, 블록 그룹 디스크립터, 데이터 블록 bitmap, inode bitmap, inode table, data blocks
inode table 과 데이터 블록 내의 빈 공간을 관리하기 위해 bitmap 사용
수퍼 블록과 그룹 디스크립터는 소실되면 안되는 매우 중요한 정보이기 때문에 각 블록 그룹마다 중복하여 기록

6. Ext3 파일 시스템과 Ext4 파일시스템
Ext3 파일 시스템: 해쉬 기반 HTree 기술, 저널링(journaling) 기능, online-resizing: 시스템 가용 시간 향상기능 추가
Ext4 파일시스템: 선할당(preallocation), 지연할당(Delayed allocation) 기법 도입, 대용량 파일의 메타데이터를 줄이는 extent 기반 데이터 블록 유지

7. 가상파일시스템

태스크는 어떻게 파일 시스템에 접근하는가 ← 시스템호출
반대로, 파일시스템은 어떤함수를 구현하여 상위계층에 제공하는가 ← VFS(가상파일시스템)

VFS(Virtual File System)
파일 시스템과 사용자 태스크 사이에 가상적인(Virtual)층 도입으로 리눅스에서 다양한 파일 시스템 지원 → 상위 계층에서 open, read() 단일한 함수를 통해 파일 시스템 접근

사용자 태스크는 특정 파일시스템이 아니라 VFS 와 데이터를 주고 받는다

사용자 태스크에게 제공할 일관된 인터페이스 정의하기 위해 VFS 내에 5 개의 객체를 도입
1) 수퍼 블록(super block) 객체
현재 사용중인(마운트된) 파일 시스템 당 하나씩 주어짐
각 파일 시스템은 자신이 관리하고 있는 파티션에 파일 시스템마다 고유한 정보를 수퍼 블록에 저장, VFS 는 이를 읽어서 관리하기 위해 범용적인 구조체인 수퍼 블록 객체 정의
2) 아이노드 객체
특정 파일과 관련된 정보를 담기 위한 구조체
파일 시스템은 파일을 저장하기 위해 각자 정의한 메타데이터 저장
→ msdos 라면 해당 파일의 디렉터리 엔트리, ext2 라면 해당 파일의 디렉터리 엔트리와 inode 를 찾아서 inode 객체를 채움
3) 파일 객체
태스크가 open 한 파일과 연관되어 있는 정보 관리 및 유지하는 용도
각 태스크가 아이노드 객체에 접근하는 동안만 메모리상에 유지됨
fd, f_pos

4) 디엔트리 객체
태스크가 파일에 접근하려면 해당 파일의 아이노드 객체를 자신의 태스크와 연관된 객체인 파일 객체에 연결해야함, 연결을 좀 더 빨리 하기 위한 캐시 역할
5) path > vfsmount : 현재 가상파일 시스템이 어떤 파일 시스템인지 본다.

8. 태스크 구조와 VFS 객체
1) task struct → files → files_struct
2) files_struct → fd_array (fd) - file
3) file
- f_dentry → dentry→ inode
- f_pos
- f_op → file_operations : 가상적인 파일 연산이 요청되면 커널은 파일에 적합한 파일 고유 함수를 사용하여 서비스 제공

결국, 리눅스가 다양한 파일 시스템 제공할 수 있는 이유
→ file_operations, inode_operation 로 가능
file_operations - 각 파일의 유형에 맞는 연산 함수를 호출할 수 있도록 지원
inode_operations - 각 파일 시스템에 맞는 연산 함수를 호출할 수 있도록 지원