TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
35 일차 (2018. 04. 11)

목차

Chapter 5 : 파일시스템과 가상 파일시스템

1) 파일시스템 일반

메모리 관리 기법은 주 기억 장치라고 불리는 저장장소를 관리하는 소프트웨어이고 파일시스템은 보조 기억 장치라고 불리는 저장장치를 관리하는 소프트웨어이다. 보조 기억 장치로는 주로 하드 디스크와 같은 장치가 사용된다. 그러나 이 디스크도 결국 실행되기 위해서는 메모리에 올라가야 한다. 이 때, 메모리 관리 기법과 마찬가지로 단편화 문제가 발생한다. 이를 효율적 관리할 수 있는 매커니즘이 필요하게 되며, 그 것을 가상 파일시스템과 커널이 해준다.

그렇다면 메모리 관리 기법과 파일시스템간의 차이점은 무엇일까? 이는 '**이름**'이다. 나머지는 메모리 관리 기법과 동일하다고 봐도 무방하다.

파일시스템은 사용자에게 이름이라는 속성으로 접근되는 추상적인 객체인 파일이라는 개념을 제공한다. 즉, **'이름'을 입력 받아 해당 데이터를 리턴 해주는 소프트웨어**가 파일시스템이라는 것이다. 하드디스크에는 메타 데이터(meta data, inode)와 사용자 데이터(user data, 실제 내용)이 저장된다. 파일시스템은 디스크 블록을 하나 할당(디바이스 드라이버(커널)이 실제 4KB 를 할당)해준다. 그리고 다시 읽을 때는 디스크 블록 번호를 외워서 진행하는 것이 아닌 '이름'을 통해 추산적인 자원인 파일을 통해 접근하게 된다.

2) 디스크 구조와 블록 관리 기법

디스크는 **원판(plotter), 팔(arm), 헤드(head)**로 구성되어 있으며, 원판에 **원 모양의 트랙(track)**들이 존재하고, 이 **트랙들의 집합을 실린더(cylinder)**라 한다. **트랙은 몇 개의 섹터(sector)**로 구성되며 일반적으로 섹터의 크기는 **512byte** 이다.

헤드는 각 원판의 읽기/쓰기가 가능한 면마다 하나씩 존재(위뿐 아니라 아래에서도 읽고 쓰기가 가능하도록)한다. 헤드가 몇 개, 트랙(실린더)가 몇 개, 트랙마다 섹터가 몇 개 등이 결정되면 디스크의 전체 용량 등 해당 디스크의 물리적 특성을 결정할 수 있게 된다. 이 정보는 디바이스 드라이버(모터 제어, 팔 제어 등)가 가진다.

디스크에서 데이터를 접근하는데 걸리는 시간을 **탐색시간(seek time), 회전 지연시간(rotational latency)** 그리고 **데이터 전송 시간(transmission time)**이라는 세가지로 구성된다.

탐색시간 : 헤드를 요청한 데이터가 존재하는 트랙 위치까지 이동하는데 걸리는 시간

회전시간 : 요청한 섹터가 헤드 아래로 위치될 때까지 디스크 원판을 회전시키는데 걸리는 시간

데이터 전송 시간 : 헤드가 섹터의 내용을 읽거나 기록하는데 걸리는 시간

여기서 탐색시간이 제일 느리다. 체감이 가능하게 움직인다. 제일 빠른 것은 전송 시간으로 체감할 수도 없을 정도로 빠르다.

하지만 파일시스템은 디스크를 물리적인 구조로 보지 않고, **논리적인 디스크 블록(disk block)들의 집합**으로 본다. 디스크 블록은 0, 1, 2, 등의 논리적인 번호를 하나씩 갖고, 디스크 블록의 크기는 일반적으로 페이지 프레임과 같이 4KB 이다. 이는 앞서 설명했듯이, 메모리에 올려야 하는데 이때, 단편화 현상이 생길 수 있기 때문에 블록 단위로 나누는 것이고, 페이지 프레임 크기에 따라 크기가 결정된 것이다. 파일시스템 성능의 최대 병목 요소는 디스크 I/O 이다. 따라서, 최근에

개발된 파일시스템의 경우 버디할당자처럼 (4K x 2^n)로 블록 크기를 크게 할당하는 성향이 있다. 어느 정도 메모리를 낭비하더라도 디스크를 빠르게 읽기 위함이다.

디스크 블록의 크기가 4KB 이고 섹터의 크기가 512byte 라면, 하나의 디스크 블록에 8 개의 섹터가 대응하게 된다. 따라서 파일시스템이 하나의 디스크 블록을 읽어 달라고 요청하면 결국 8 개의 섹터 내용이 읽혀지게 된다. 디스크 블록의 번호를 이에 대응되는 섹터들로 매핑 시키는 일은 디바이스 드라이버 또는 디스크 컨트롤러가 담당한다.

디스크 블록의 할당 방법은 크게 두 가지로 연속할당과 불연속 할당이 있다.

연속할당 : 파일에게 연속된 디스크 블록을 할당하는 방법.
탐색 시간이 오래 걸리지 않아 빠르다.

불연속 할당 : 디스크 블록 빈 공간에 순차적으로 할당되는 방법으로 다음연결 되는 것이 어디인지 알려줘야 한다. 연속할당과 달리 탐색시간이 걸려 속도가 느리지만, 공간 활용이 되어 용량면에서는 좋다.
→간혹, 디스크 조각 모음을 해주는 이유이다. 연속할당으로 바꿔주어 속도↑

파일의 크기가 변하는 것은 매우 빈번한데 새로운 내용을 추가 할 때, 연속 할당을 하지 못하면 불연속 할당을 하게 된다. 이는 파일 크기가 커질 때 기존에 있던 블록들을 다른 곳으로 복사하는 것은 성능상 매우 큰 문제가 된다. 따라서 파일시스템을 설계할 때, 연속할당만을 사용하는 경우는 거의 없다.

불연속 할당의 경우 파일에 속한 디스크 블록들이 어디에 위치하고 있는지에 대한 정보를 기록해 두어야 하는데, 그 방법으로 **블록체인 기법, 인덱스 블록 기법, FAT(File Allocation Table)** 등이 있다.

- 블록체인 기법

같은 파일에 속한 디스크 블록들을 **체인으로 연결해 놓는 방법**으로 각 블록에 포인터를 두어 다음 블록의 위치를 기록한다. Linked list 와 유사하며 포인터를 저장해야 하므로 실질적으로 4KB 를 다 사용하지 못한다. file name, start, size 를 저장한다. 첫 번째 디스크 블록에 가면 포인터를 이용해 다음 블록의 위치를 찾아 갈 수 있다. lseek()같은 시스템 콜을 사용할 때 파일의 끝 부분을 읽으려는 경우에 앞 부분의 데이터 블록을 읽어야 하고 중간 블록이 유실되면 나머지 데이터까지 모두 잃게 되는 단점이 있다.

- 인덱스 블록 기법

블록들에 대한 **위치 정보들을 기록한 인덱스 블록**을 따로 사용하는 방법이다. file name, index, size 를 index block 에 저장하여 각 인덱스들은 디스크 블록을 가리키고 있다. lseek()를 사용하여 파일의 끝 부분을 접근할 때 데이터 블록을 일일이 읽을 필요가 없지만 인덱스 블록이 유실되면 데이터 전체가 소실되는 단점이 있다.

- FAT 기법

같은 파일에 속해 있는 블록들의 위치를 **FAT 라는 자료구조**에 기록해 놓는 방법이다. 블록체인 기법과 인덱스 블록 기법의 개념을 합친 기법으로, 파일시스템 전체적으로 하나의 FAT 이 존재하는 것이다. 이 구조에서 FF 는 파일의 끝을 의미하며 0 은 free 상태를 나타낸다. start 인덱스가 있

어 데이터가 중간에 유실되어도 복원이 가능한 장점이 있지만, FAT 구조의 유실은 파일시스템 내의 모든 파일이 소실되는 단점이 있다.

※ root file system 은 언제나 '2'번이 걸린다.

리눅스에서 사용되는 파일시스템인 Ext2 나 Ext3 는 인덱스 블록 기법과 유사한 기법을 사용한다. 그것이 바로 inode 이다.

3) FAT 파일시스템

파일시스템이 관리하는 데이터는 크게 메타 데이터와 유저 데이터로 구분된다. 메타 데이터는 FAT 테이블, 디렉터리 엔트리, 수퍼 블록으로 구성된다.

파일시스템들은 저마다 **자신만의 디렉터리 구조체를 선언**해 놓은 뒤, 파일이 생성될 때, 이 구조체의 각 내용들을 채워서 디스크에 저장한다. 사용자가 특정 이름을 가지는 파일의 내용을 읽어오도록 파일 시스템에 요청한다면, 파일시스템은 사용자가 요청한 파일의 이름을 가지고 디렉터리 엔트리를 찾아 낸 뒤, 디렉터리 엔트리가 가지고 있는 정보를 통해 데이터 블록을 찾아 사용자에게 제공한다.

그렇다면 파일시스템은 디렉터리 엔트리를 어떻게 찾을까? 사용자는 파일 이름을 통해 데이터를 접근하려 한다. 파일의 이름은 현재 디렉터리 위치를 기준으로 시작되는 상대 경로와 '/(root)'에서부터 시작되는 절대 경로 두 가지로 나뉜다. 리눅스는 사용자 태스크의 현재 작업 디렉터리(CWD : Current Working Directory)는 항상 task_struct 구조체에 유지하고 있기 때문에 언제든 상대 경로는 절대 경로로 변환 가능하다.

※ '**최상위 디렉터리가 위치하고 있는 곳**' 같은 정보를 적어서 자신이 관리하는 공간의 맨 앞부분에 적어두는 것을 슈퍼 블록(super block)이라 한다. 이렇게 하면 나중에 언제라도 이 디스크를 사용하려 하는 시점에 디스크의 맨 앞부분만 읽으면 '/' 디렉터리의 위치를 알 수 있다. '/' 디렉터리 하위에 존재하는 모든 파일을 찾을 수 있게 된다.

4) inode 구조

리눅스의 디폴트 파일 시스템인 ext4 등, ext 계열 파일 시스템이 채택하고 있는 구조이다.