리눅스 환경을 위한 USB MP3 Player의 커널 모듈과 제어 시스템의 설계 및 구현

이준택° 신동헌 조수현 김영학
금오공과대학교 컴퓨터공학과 대학원
{next°, genius, shcho, yhkim}@cespc1.kumoh.ac.kr

Design and Implementation of Control system and Kernel module of USB MP3 Player for Linux Environment

Joon-Taek Lee° Dong-Hern Shin Soo-Hyun Cho Young-Hak Kim
Graduate School, Dept. of Computer Engineering, Kumoh National Institute of Technology

요 약

PC의 주변장치를 사용하기 위한 운영체제의 디바이스 드라이버의 구현은 하드웨어에 대한 정확한 명세가 없으면, 많은 노력과 비용이 요구된다. 현재 대부분의 PC 주변장치를 생산하는 업체는 리눅스를 위해서 별도의 디바이스 드라이버와 사용자 인터페이스 프로그램을 제공하고 있지 않다. 본 논문에서는 윈도우 운영체제에서 사용하기 위해 만들어진 MP3 Player의 디바이스 드라이버 및 사용자 프로그램이 윈도우 상에서 작동되는 내부적인 과정을 분석하고 리눅스의 proc 파일 시스템을 이용하여 하드웨어의 명세 및 프로토콜을 조사하였다. 그리고 여기에서 얻은 장치의 정보를 리눅스를 위한 디바이스 드라이버 및 사용자 프로그램의 구현에 응용하여, 리눅스 환경에서 적은 비용과 시간을 들여 디바이스 드라이버를 개발하는 방법을 제안한다.

1.서 론

최근 PC 시장은 제한된 연결 장치 수나 사용의 편리성을 해결하기 위한 USB[1]라는 외부 인터페이스의 보급으로, 이를 이용한 디지털 카메라, MP3 Player, 휴대용 저장매체 등과 같은 장치들이 급속도로 증가하고 있다. 또한 저렴한 비용을 가지면서도 안정적인 운영체제로 주목 받고 있는 리눅스의 사용 역시 다양한 분야에서 증가하고 있는 추세이다.

현재까지는 대부분의 주변장치를 생산하는 업체에서는 윈도우 운영체제에서의 사용을 위한 디바이스 드라이버와 사용자 프로그램만을 제공하고 있으며, 생산한 하드웨어에 대한 자세한 명세를 공개하지 않고 있다. 때문에 이러한 하드웨어를 이용한 리눅스 기반 소프트웨어의 개발은, 하드웨어를 생산하는 업체의 지원이나 정보 없이는 많은 노력과 비용이 요구된다.

본 논문에서는 리눅스에서 USB MP3 Player의 디바이스 드라이버[2-4] 및 사용자 인터페이스 프로그램의 구현에 필요한 하드웨어의 정보를 리눅스의 proc 파일 시스템을 이용하여 조사하고, 윈도우를 운영체제로 사용하는 PC와 MP3 Player 사이에 주고받는 데이터를 추적하여 제어를 위해 필요한 정보를 추출하는 방법을 제시하고 이를 구현한 내용을 기술하였다.

본 논문의 구성은 다음과 같다. 2 장에서는 리눅스의 USB 디바이스 드라이버, 플래시 메모리, FAT(File Allocation Table)[5][6] 파일 시스템의 특성 및 구조에 대해서 고찰한다. 3 장에서는 구현 과정에서의 고려 사항 및 관련된 내용과 설계 과정을, 4 장에서는 구현 및 평가를, 마지막으로 5장에서는 결론 및 향후 연구 과제에 대하여 논한다.

2.관련 개념의 이해

2.1 USB 디바이스 드라이버

리눅스는 프로세스 관리, 메모리 관리, 파일 시스템, 디바이스 제어 같은 커널의 모든 기능들이 하나의 커널에 들어가 있고, 모두 커널 모드에서 실행되고 있다. 디바이스 드라이버는 장치를 직접적으로 다루는 소프트웨어로서 커널 모드에서 실행 되어야 한다. 때문에 디바이스 드라이버는 커널에 동적으로 로드, 언로드 될 수 있는 커널 모듈 형태로 만들어진다. 리눅스는 하드웨어 장치들을 파일로 표현하여 관리하고 있으며 다양한 장치와 일반 파일들을 동일한 시스템 콜(System Call)을 사용하여 접근할 수 있다. 리눅스의 USB 디바이스 드라이버는 장치파일을 통하여 USB 장치에 대한 응용 프로그램의 입출력 요구를 적절한 형태로 USB장치의 core쪽으로 전송하고 결과를 되돌려 주는 역할을 한다.

2.2 플래시 메모리(Flash Memory)

플래시 메모리는 전원이 끊어져도 데이터가 지워지지 않는 비휘발성 메모리로서 블록 단위로 내용을 지울 수 있고, 다시 프로그래밍할 수도 있다. 현재 일반적으로 사용되는 플래시 메모리는 크게 NOR형, NAND형 두 가지로 구분한다. MP3 Player와 같은 기기들은 대부분이 데이터 저장 매체로서 NAND형 플래시 메모리를 사용하며 표 1과 같은 특징을 가지고 있다.

표 1. NAND형 플래시 메모리의 특징

NAND형 플래시 메모리의 장단점	
장점	단점
빠른 속도의 프로그래밍 빠른 속도의 지우는 작업 작은 블록 크기	느린 임의 접근 바이트 단위의 프로그래밍 불가능

2-3 FAT(File Allocation Table)

FAT 파일 시스템은 여러 운영체제에서 지원하고 있기 때문에 PC와 연동하여 사용할 수 있는 대부분의 플래시 메모리를 사용하는 기기들은 파일 시스템으로서 FAT를 사용하고 있다. FAT 파일 시스템은 몇 개의 섹터로 구성되는 클러스터 단위로 저장 매체를 관리하고 하나의 파일에 관련된 클러스터 들을 연결 리스트로 관리하고 있다. 저장 매체 내에 가질 수 있는 섹터의 수에 따라 FAT12, FAT16, FAT32 으로 나누어 지며 본 논문에서는 FAT16을 사용하여 16MB 이하의 메모리에 대해서는 고려하지 않았다.

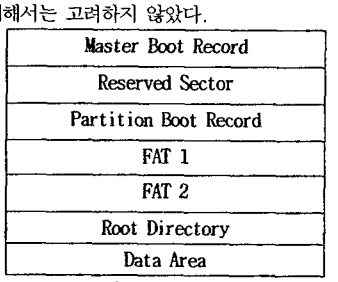

그림 1. FAT 구조

그림 1은 FAT의 구조를 나타낸다. Master Boot Record는 부트코드가 들어가는 부분이며, FAT1은 저장 매체 내에 위치해 있는 클러스터들의 위치를 나타낸다. 그리고 FAT2는 FAT1의 손상시를 위한 복사본이다. Root Directory는 파일 시스템의 루트 디렉토리에 위치하는 파일 및 디렉토리들에 대한 정보이며 Data Area에는 실제 파일의 데이터들이 저장된다. 본 논문에서는 표준 FAT 파일 시스템의 각 영역을 물리적으로 매핑 하는데 있어서 다른 형태를 사용하였다.

3.커널 모듈과 제어 시스템의 설계

본 논문에서 구현한 내용은 크게 하드웨어를 제어하기 위해 커널 모듈로서 구현한 디바이스 드라이버와 이를 이용하는 상위 계층의 제어 시스템으로 나누어져 있다.

3.1 커널 모듈

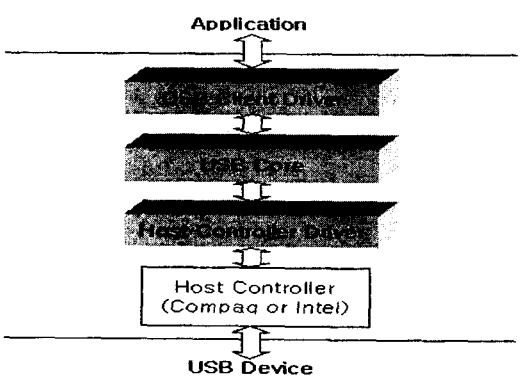

그림 2. USB Device Driver Stack

그림 2는 장치에서 사용자 프로그램까지의 데이터 전달 순서를 나타내고 있으며 Host Controller Driver는 운영체제가 담당하고 있다. 때문에 장치의 사용을 위해서는 Client Driver만을 구현하면 된다. USB 디바이스 드라이버의 구현을 위해서는 몇 가지 하드웨어에 대한 정보를 알아야 하는데, 이것을 proc 파일 시스템의 내용을 이용해 얻어 낼 수 있다. 다음은 장치가 호스트 PC에 연결 되었을 때의 /proc/bus/usb/devices 파일 내용 중 일부이다.

```
E:  Ad=03(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=83(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=1ms
```

Ad가 나타내는 것이 데이터의 논리적인 source 또는 sink가 된다고 할 수 있는 endpoint descriptor의 주소이며 입력/출력 가운데 어느 것인지를 나타내는 문자를 포함하고 있다. Atr은 endpoint에 사용되는 전송 타입을 나타내며 이를 나타내는 문자열이 뒤에 나온다. MxPS는 요구되는 대역폭의 크기를 나타내며 lvl은 endpoint descriptor 호출의 간격을 밀리 세컨드(ms) 단위로 나타낸다.

USB와 같은 장치들은 호스트 PC에 항상 접속된 상태라고 할 수 없기 때문에 이를 위해, 장치가 호스트 PC에 연결 되거나 끊어 졌을 때의 처리를 위해 일반적인 디바이스 드라이버와는 다른 자료 구조가 추가적으로 존재한다.

3.2 제어 시스템

사용자 인터페이스 프로그램은 일반적인 파일을 다루는 것과 같이 장치 파일을 다루면 된다. 장치 파일에 대한 operation의 집합이 커널에 모듈이 로드 될 때 등록 되므로 커널에 등록된 장치의 시스템 콜을 사용하여 장치를 다룰 수 있다. 개괄적인 동작을 살펴 보면, open()으로 장치 파일을 열고, write()로 장치에 제어 명령어를 보내고, read()로 결과를 되돌려 받아 처리하여 윈도우의 manager 소프트웨어와 동일한 기능을 할 수 있게 한다.

MP3 Player의 기본적인 동작들과 입출력 요청 처리 등을 위한 펌웨어의 동작을 알기 위해, sniffusb[7]라는 USB Bus의 트래픽을 관찰하기 위한 툴을 사용하여 윈도우용 manager 프로그램과 MP3 Player간에 USB를 통해 주고 받는 전체 데이터를 볼 수 있었으며 파일의 송수신, 디렉토리의 변경 등과 같은 동작들을 직접 수행해 봄으로서 각각의 기능에 대한 제어 명령을 알 수 있었다. 또한 전체 섹터의 내용을 덤프해서 조사 함으로서 MP3 Player가 사용하고 있는 변형된 형태의 FAT의 구조도 파악 할 수 있었다.

4.구현 및 평가

본 논문에서는 linux kernel 2.4.18을 기반으로 하는 Hancom Linux 3.0을 일반적인 Desktop 환경에 설치하고 gcc version 3.2를 사용하여 커널 모듈 및 제어 시스템을 개발 하였다.

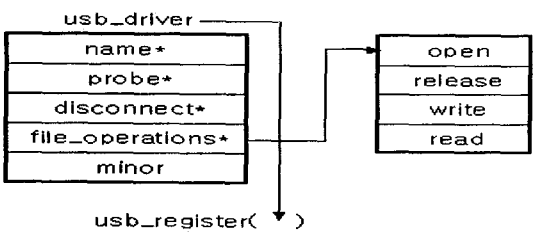

그림 3. 디바이스 드라이버 구조체

그림 3은 USB 디바이스 드라이버의 구조체와 초기화 할 때의 과정을 보인 것이다. usb_register()함수를 이용하여 usb_driver구조체를 매개변수로 하여 커널에 USB 디바이스 드라이버를 등록한다. *probe* 함수는 USB장치가 Bus에 연결 되었을 때 자신이 제어할 장치인지를 벤더의 ID나 제품의 ID로서 식별하며, disconnect()함수는 디바이스 드라이버에 의해서 사용되어지던 장치가 제거 될 때 호출된다. 드라이버를 등록할 때에는 장치의 이름이 필요하며, USB 장치의 major 번호는 180번으로 고정 되어 있기 때문에 minor 번호의 정보만 가지면 된다.

USB의 데이터 전송은 *transaction format*에 따라 4가지 종류로 나누어 지는데, 본 논문에서 사용한 장치는 데이터를 패킷 형태로 전송하기 위해 Bulk Transfer 모드를 사용하는 것을 리눅스의 proc 파일 시스템을 통해서 확인 하였다. 장치로의 입출력은 각각의 endpoint descriptor의 주소를 Bulk Transfer를 위한 pipe이며 usb_bulk_msg() 함수의 매개변수로 사용하여 수행 할 수 있다.

사용자 프로그램의 주요 기능인 파일을 읽고, 쓰고, 디렉토리를 변경하는 동작들은 모두 FAT entry 정보를 읽어서 실제 파일과 디렉토리의 클러스터 위치를 알아 내는 동작이 기본이 된다. 예로서 파일을 호스트 PC로 다운로드 하는 동작을 설명 하겠다.

본 논문에서는 FAT와 동일 하지만, 8자의 파일 이름과 3자의 확장자명의 제한을 없앤 vfat 파일 시스템을 사용하였다. 루트 디렉토리를 기준으로 *Config.dat*라는 파일을 다운로드 하고자 할 때, 먼저 루트 디렉토리의 엔트리에서 파일 이름이 일치하는 32byte 파일 엔트리 정보를 읽는다. 읽은 파일 엔트리에서 27, 28번째 바이트가 파일의 시작 클러스터의 위치를 나타내는 정보이다. 하나의 섹터를 나타내기 위해 16byte를 사용하는 FAT 엔트리를 검색하여 엔트리의 첫번째 바이트가 파일의 첫 섹터임을 나타내는 aa와 같고, 두번째 바이트가 시작 클러스터를 나타내는 값과 일치 하는 FAT 엔트리를 찾아낸다. 이 엔트리의 9에서 11번째 바이트는 다음 클러스터의 시작 섹터 값을 가지고 있기 때문에 이 값을 연속적으로 참조하여 파일의 전체 클러스터를 읽을 수 있다. 그림 4는 실제 각 *entry*들의 Hexa code를 덤프 한 값이다.

```
43 4f 4e 46 49 47 20 20 44 41 54 06 00 80 e5 59
24 2e 24 2e 00 00 68 5c 24 2e 07 00 00 fe 02 00
```

-Root Directory Entry(File Entry)

```
aa 07 00 0c ff ff 43 01 00 01 20 00 00 00 07 00
```

-FAT Entry(File Entry)

그림 4. Root Directory 와 FAT의 Entry Hexa code

본 논문에서는 긴 파일 이름을 나타내는 파일 엔트리의 첫번째 바이트가 헥사값 0x40이상의 값을 가지고 있다면, 0x40을 뺀 나머지 값 만큼의 파일 엔트리가 긴 파일 이름을 표시하기 위해 사용된다. 이런 구조를 이용해 긴 이름을 가진 파일의 경우, 파일 이름을 나타내기 위해 순차적으로 나타나는 파일의 전체 엔트리를 검색 하지 않고도 파일의 시작 클러스터의 위치를 빠르게 찾아 낼 수 있도록 구현 하였다. 그림 5는 긴 파일 이름을 가지는 파일의 entry 전체 Hexa Code를 덤프 한 값이다.

```
43 74 00 78 00 74 00 00 00 ff ff 0f 00 00 ff ff  Ct.x.t..........
ff ff ff ff ff ff ff ff ff ff 00 00 ff ff ff ff  ................
02 74 00 51 00 74 00 65 00 73 00 0f 00 00 74 00  .t.Q.t.e.s....t.
51 00 65 00 6e 00 74 00 72 00 00 00 79 00 2e 00  Q.e.n.t.r...y...
01 6c 00 6f 00 6e 00 67 00 5f 00 0f 00 00 66 00  .l.o.n.g._....f.
69 00 6c 00 65 00 6e 00 61 00 00 00 6d 00 65 00  i.l.e.n.a...m.e.
4c 4f 4e 47 5f 46 7e 31 54 58 54 20 00 00 b2 ae  LONG_F~1TXT ....
53 2e 53 2e 00 00 b2 ae 53 2e 20 00 1e 00 00 00  S.S.....S. .....
```

-긴 이름을 갖는 파일의 파일 엔트리

그림 5. 긴 파일 이름을 나타내는 엔트리

다음은 쉘에서 제어 시스템을 구동시켜 MP3 Player의 루트 디렉토리의 내용을 보기 위해 *dir* 명령을 실행시킨 결과이다. 파일의 생성 날짜, 시간, 크기, 이름 순이며 파일의 크기가 0으로 표시된 것은 디렉토리를 의미한다.

```
command>dir
2003.04.01 11:35 196096-Config.dat
2003.04.01 22:53      0-Stratovarius
2003.04.17 20:11      0-테스트
2003.05.01 00:03      0-etc
2003.02.19 21:53      30-long_filename_test_entry.txt
```

표 2는 2.1MB 크기의 파일을 MP3 Player로 업/다운 로드 한 시간을 측정 한 것이다. 평균적으로, 초당 427K의 데이터를 다운로드, 510K의 데이터를 업로드 할 수 있었다. 회전식 자기매체에 비해 플래시 메모리가 빠른 속도로 접근 되어짐을 확인 할 수 있었다.

표 2. 파일의 업/다운 로드 소요 시간(micro second)

시도횟수	Download 소요시간	Upload 소요시간
1	4974236	4105112
2	4897119	4103011
3	5009590	4103414
4	4943129	4104213

5. 결론

본 논문에서는 윈도우 운영체제에서 사용되는 MP3 Player의 사용자 인터페이스 프로그램과 호스트 PC 사이에 송수신 하는 데이터를 분석하고, 리눅스의 proc 파일 시스템에서 USB Bus에 연결된 하드웨어 장치의 정보를 알 수 있었다. 또한 하드웨어의 자세한 명세 없이 장치의 디바이스 드라이버와 사용자 인터페이스 프로그램을 좀 더 작은 노력과 시간으로 개발 할 수 있는 방법을 제시하고 구현하였다.

향후 연구 과제로는 MP3 Player의 저장 매체인 플래시 메모리를 블록 디바이스처럼 처리하여, Mass Storage와 같이 리눅스에서 마운트 하여 사용 할 수 있도록 지원 하는 것이다.

참고 문헌

[1] Universal Serial Bus Revision 1.1 specification, <http://www.usb.org>

[2] LINUX DEVICE DRIVERS, Alessandro Rubini, O'Reilly & Associates, Inc. 1998

[3] USB Device Driver Programming Guide, Detler Fligel, 2000

[4] http://usb.cs.tum.edu, Linux USB Developer Pages

[5] General Overview of On-Disk Format, Microsoft C-orporation

[6] $KERNEL_SOURCE_HOME/Documentation/filesystem/vfat.txt

[7] http://home.jps.net/~koma, USB Traffic watcher