임베디드 시스템을 위한 가상 파일 시스템 구현

송재영, 이호송[0], 성영락[†], 이철훈, 권택근
충남대학교 컴퓨터공학과, 국민대학교 전자정보통신공학부
{jysong, hslee[0], chlee, tgkwon}@ce.cnu.ac.kr, [†] yeong@mail.kookmin.ac.kr

Implementation of Virtual File System for Embedded Systems

Jae-Young Song, Ho-Song Lee[0], Yeong Rak Seong[†], Cheol-Hoon Lee, and Taeck-Geun Kwon
Dept. of Computer Engineering, Chungnam National Univ.
[†] School of Electrical Engineering, Kookmin Univ.

요 약

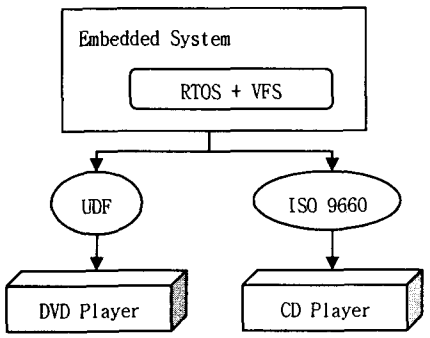
임베디드 시스템에 대한 관심과 개발이 급속도로 진행됨에 따라 임베디드 시스템에서도 파일 시스템을 필요로 하게 되었다. 특히 실 시간 운영체제 (Real Time Operating System : RTOS)를 사용하는 경우에는 운영체제가 사용되는 임베디드 시스템에 따라 필요로 하는 파일 시스템이 하나 또는 그 이상으로 다르게 되어 그 파일시스템을 통합하여 관리하기 위한 가상 파일 시스템 (Virtual File System : VFS)을 필요로 하게 되었다. 본 논문에서는 실 시간 운영체제와 함께 동작을 하는 가상 파일 시스템을 구현하는 방법을 제시한다.

1. 서 론

최근 많은 연구소 및 산업체에서 임베디드 시스템에 대한 개발과 연구가 활발하게 진행이 되는 가운데 실 시간 운영체제를 사용하면서 파일 시스템을 필요로 하는 시스템들이 급증하고 있다. 예를 들어 MP3 플레이어는 컴퓨터로부터 mp3 파일을 전송 받고, 현재 메모리에 존재하는 mp3 파일이 무엇인지를 PC 를 통해 사용자에게 확인시키기 위해 FAT 파일시스템을 사용하게 된다. DVD 플레이어는 UDF (Universal Disk Format) 파일 시스템을 통해 DVD 매체로부터 파일을 읽어 화면에 뿌려주게 된다. CD 플레이어는 ISO9660 파일 시스템을 통해 CD 에서 음악 파일을 읽어 음악을 재생하거나 PC 를 통해 그 목록을 사용자에게 확인 시켜주게 된다.

기존에는 하나의 임베디드 시스템을 개발하기 위해 각각의 특성에 맞는 하드웨어를 개발하여 각각이 필요로 하는 실 시간 운영체제와 파일 시스템을 사용하였다. 그래서 개발자는 MP3 플레이어를 위한 하드웨어와 이를 위한 파일 시스템을 개발하고, DVD 플레이어를 위한 하드웨어와 파일 시스템을 개발하는 등 각각에 맞는 하드웨어와 파일 시스템을 개발해야 하는 번거로움이 있었다. 하지만 하나의 하드웨어로 다양한 임베디드 시스템을 개발하려는 시도가 일어나면서 시스템이 필요로 하는 파일 시스템을 적절하게 지원해주어야 하는 요구가 발생하게 되었다. 이러한 요구로 인해 [그림 1]과 같이 임베디드 시스템에는 VFS 를 포함하는 실 시간 운영체제를 가지고 있고, 시스템 요구에 따른 파일 시스템을 외부에서 라이브러리 형태로 지원을 하여 컴파일 및 실행이 가능하게 하도록 하는 구현을 하게 되었다.

이 파일 시스템은 차후에 충남대학교 컴퓨터공학과 병렬처리 연구실에서 개발한 RTOS 에 통합될 예정이다. 이 RTOS 는 32-bit CPU 를 대상으로 설계되었으며, ARM 920T 를 기반으로 한 삼성 S3C2800™ 32-bit RISC Micro Processor 에서 테스트 중이다.



[그림 1] VFS 를 이용한 Embedded System 설계 예

본 논문의 구조는 다음과 같다. 2 장에서는 RTOS 와 RTOS 커널의 구조를 소개하고, 3 장에서는 구현된 VFS 의 자료 구조를 간략히 기술한다. 4 장에서는 일반 파일 시스템을 구현된 VFS 에 등록하고 마운트하는 절차를 기술하고 5 장에서는 VFS 에서 사용하기 위해 구현한 시스템 콜(System Call) 들을 소개하며 결론 및 향후 과제는 6 장에서 기술하도록 한다.

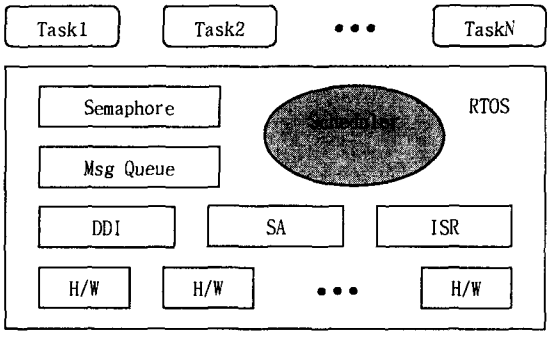
2. RTOS 소개

2.1 실시간 운영체제

실시간 운영체제는 임베디드 시스템에 사용되는 대표적인 운영체제이다. 임베디드 시스템은 자원(메모리, 전원 등)이 제한된 환경에서 특정 기능을 수행하도록 설계된 시스템을 의미한다. 실시간 운영체제에서 가장 중요한 부분은 여

러가지 기능들을 관리하는 태스크(Task)들에게 CPU 점유를 할당하는 기능을 하는 스케줄러(Scheduler)이다 [1].

2.2 RTOS 커널 구조



[그림 2] RTOS 커널 전체 구성

DDI : Device Driver Interface
SA : Storage Allocation
ISR : Interrupt Service Routine

RTOS 스케줄러에서 사용하는 스케줄링 알고리즘은 단일 프로세서 시스템에서는 최적이라 알려진 EDF (Earlist Deadline First)를 사용하는데, 이 EDF 는 여러가지 이벤트들이 발생했을 때 데드라인까지의 시간여유가 가장 짧은, 즉 급한 이벤트부터 처리하도록 한다 [1].

범용 운영체제와 마찬가지로 RTOS 에서도 동적 메모리 관리를 하게 되는데, 이는 시간 결정성을 파괴하고 외부 단편화 문제를 일으킬 수 있다. 이와 같은 문제 해결을 위해 RTOS 에서의 동적 메모리 관리 체계에서는 시스템 부팅 초기에 임의의 크기 메모리를 할당 받을 수 있는 Heap Storage Manager 와 이를 통한 고정된 메모리 블록을 사용할 수 있도록 하는 Memory Pools 를 제공한다. 시스템 부팅 초기에는 시간 결정성 및 메모리 단편화 문제가 발생하지 않으므로 내부적으로 필요한 메모리 및 Memory Pools 에서 필요로 하는 메모리를 미리 할당 받는다. 이후부터는 Memory Pools 를 통한 고정된 크기의 메모리를 사용하게 된다.[1]

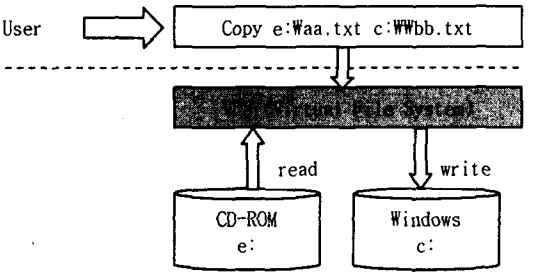
RTOS 커널의 핵심은 태스크의 생성이다. 각각의 태스크는 이를 관리하는 태스크 관리 블록과 1:1 로 매핑이 되며 생성 가능한 태스크 관리 블록의 최대수는 시스템에서 생성 가능한 태스크의 최대수가 된다.

3. VFS 자료 구조

저장 장치마다 각 저장 장치에 맞게 파일을 다루기 위한 파일시스템을 가지고 있다. 저장 장치가 서로 다른 파일시스템을 사용하는 경우 사용자가 파일 시스템에 관계없이 통일된 인터페이스를 통해 각각의 저장 장치에 접근하여 파일을 다루도록 도와주는 것이 가상 파일 시스템의 역할이다. 예를 들어 [그림 3]에서와 같이 사용자가 CD-ROM 으로부터 하드 디스크로의 파일 복사를 수행하려 하는 경우 사용자가 각각의 파일 시스템에 맞추어 동작을 수행하는 것이 아니라 일관된 인터페이스를 통해 VFS 에 접근을 하고, VFS 는 각 파일 시스템에 접근해서 사용자가 원하는 동작을 수행하도록 하게 된다. 구현된 VFS 는 일반적으로 VFS 가 가지고 있는 자료 구조들을 가지고 있다.

3.1 Superblock

모든 파일 시스템은 각각의 파일시스템에 대한 정보를 유지하기 위해 superblock object 를 가지고 있으며 VFS 도 마찬가지로 superblock object 를 가지고 있다. Superblock 과 연관된 작업들은 superblock operations 라 불리는 함수들로 처리를 하게 된다 [2].



[그림 3] 가상 파일 시스템의 동작 개념도

3.2 Inode

파일 시스템이 파일을 다루는데 있어 필요한 일반적인 모든 정보는 inode object 가 유지하게 된다. Inode operations 라 불리는 함수들을 통해서 inode 에 연관된 작업들을 처리할 수 있다 [2].

3.3 File

Open 된 파일에 대한 정보를 가지고 이 파일을 처리하는데 필요한 정보는 file object 에서 유지하게 된다. File object 는 파일이 open 될 때 생성이 된다. 파일에서 어떠한 정보를 읽는 read operation 이나 파일에 어떤 정보를 기록하는 write operation 등의 operation 들은 file operations 로 정의가 되어있다 [2].

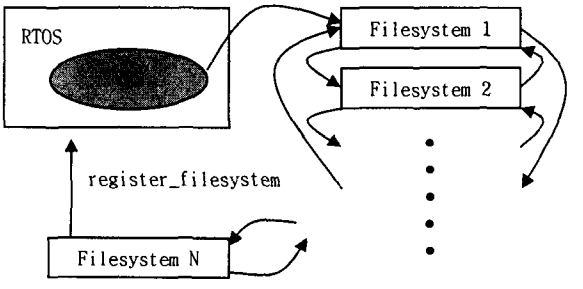
3.4 Dentry

VFS 에서는 디렉토리를 파일과 다른 디렉토리들의 리스트를 가지고있는 파일로 간주를 한다. 디렉토리를 읽었을 경우 디렉토리에 대한 정보를 dentry object 에 저장하고 유지하게 된다. Dentry object 와 연관된 작업을 하기 위해서는 dentry operations 를 이용하게 된다.

4. 일반 파일 시스템의 등록 및 마운트 / 언마운트

Linux 나 Windows 등의 범용 운영체제에서는 기본적으로 ROOT 를 차지하는 파일시스템이 존재한다. Linux 의 경우 ext2 파일 시스템이 그 예인데, 시스템이 부팅되는 시점에서 루트 파일 시스템으로 자리를 잡게 된다. 하지만 자원이 제한된 임베디드 시스템에서 고정된 루트 파일시스템을 가지기는 힘들다.

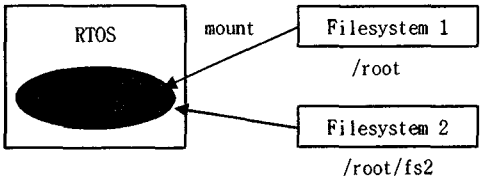
4.1 파일 시스템의 등록



[그림 4] 파일 시스템의 등록

사용자는 register_filesytem 함수를 이용하여 등록하고자 하는 일반 파일 시스템의 정보를 가상 파일 시스템에 등록하게 된다. [그림 4]에서와 같이 사용자는 사용하기를 원하는 파일 시스템들을 VFS 에 미리 등록을 하고, 등록된 파일 시스템은 이중 연결 리스트로 유지가 된다. 등록할 때 사용하는 이름으로 차후 필요한 경우 파일 시스템을 리스트에서 찾아올 수 있다. 여러 개의 파일 시스템을 사용하고자 할 경우에는 루트 파일 시스템으로 사용하려는 파일 시스템을 먼저 마운트하여 사용하도록 구현하였다.

4.2 파일 시스템의 마운트 및 언마운트



[그림 5] 일반 파일시스템의 마운트

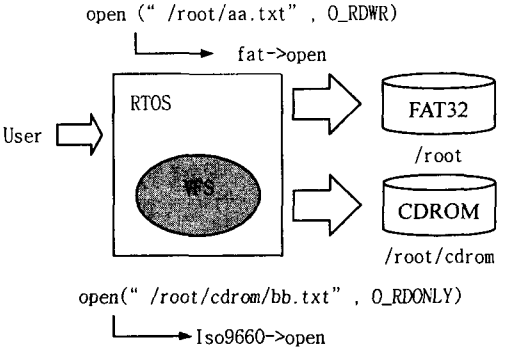
범용 운영체제에서 사용되는 파일 시스템들과는 달리, 기본적으로 주어지는 루트 파일 시스템이 없기 때문에 원하는 파일 시스템을 처음으로 마운트 하여 루트 파일 시스템으로 사용할 수 있고, 루트로 정한 파일 시스템 아래에 다른 파일 시스템을 마운트 하여 사용할 수 있게 된다. [그림 5]에서 볼 수 있듯, 제일 먼저 마운트 되는 파일 시스템은 /root 또는 / 주소를 가지게 되고, 그 다음부터 마운트 되는 파일 시스템은 사용자가 그 주소를 변경할 수 있도록 구현되었다. 마운트 하려는 파일 시스템은 반드시 등록이 되어 있어야 한다. 언마운트 하려는 파일 시스템이 루트 파일 시스템인 경우에는 마운트 된 모든 파일 시스템들이 언마운트 된다.

5. 시스템 콜

[표 1] VFS의 시스템 콜

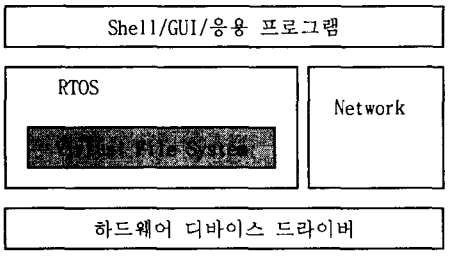
mount, umount	Mount/unmount filesystems
sysfs	Get filesystem information
statfs, fstatfs, ustat	Get filesystem statistics
chdir, fchdir, getcwd	Manipulate current directory
mkdir, rmdir	Create and destroy directory
stat, fstat, lstat	Read file status
open, close, create	Open and close file
lseek	Change file pointer
read, write	Carry out file I/O operation

사용자가 VFS 를 통해 파일 시스템이나 일반적인 파일 또는 디렉토리 등에 어떠한 동작을 취하고자 할 경우에 시스템 콜을 이용하게 된다. 대표적인 시스템 콜의 경우 파일 시스템을 마운트 하거나 언마운트 하기 위해 사용하는 mount() / umount() 가 있고, 디바이스로부터 데이터를 읽거나 디바이스에 데이터를 기록하기 위해 사용하는 read() / write() 가 있다. 그 외에도 파일을 열거나 닫을 때 사용이 되는 open() / close() 등이 지원이 된다. [표 1]에서 구현된 몇 개의 시스템 콜들을 소개하였다. [그림 6]에서 볼 수 있는 것 처럼 사용자는 읽고자 하는 파일의 실제 위치가 CD-ROM 에 있는 것인지 또는 Floppy Disk 에 있는 것인지에 상관 없이 파일 시스템이 마운트 된 위치와 파일 이름만 가지고 원하는 파일을 오픈 할 수 있게 된다.



[그림 6] 시스템 콜을 이용한 파일 오픈 예

6. 결론 및 향후 과제



[그림 7] 실시간 운영체제 전체 구성도

본 논문에서는 임베디스 시스템을 위한 가상 파일 시스템을 구현하였다. 그 구현된 가상 파일 시스템을 가진 실시간 운영체제는 [그림 7]과 같은 모습을 가지게 된다. 가상 파일 시스템은 사용자에게 표준적인 인터페이스를 제공할 뿐 가상 파일 시스템 자체만으로 저장 매체로부터 데이터를 읽어 오거나 저장 매체에 데이터를 기록하는 작업을 할 수는 없다. 현재는 DVD 를 지원하는 UDF 파일 시스템, CD 를 지원하는 ISO9660 파일 시스템, MS-DOS 에서 사용하는 FAT16/32 파일 시스템 등을 VFS 에 등록하고, 마운트 하여 데이터를 읽고 쓰는 작업을 진행중에 있다. 향후 과제로는 현재 구현된 VFS 가 자원이 제한되어 있는 임베디드 시스템을 위한 것인 만큼, 적은 자원을 효율적으로 사용할 수 있는 성능 개선이 필요로 한다. 또 더 많은 디바이스를 위한 파일 시스템들을 함께 사용할 수 있도록 하기 위해서는 더 많은 디바이스 드라이버의 구현이 필요하다.

7. 참고문헌

[1] 박희상, " 태스크 기반 선점형 실시간 운영체제 설계 및 구현" , 충남대학교 졸업논문, 2003.
[2] Daniel P.Bovet & Marco Cesati, *Understanding the Linux Kernel 2nd*, O' REILLY, 2003.
[3] 강석민, " 임베디드 시스템을 위한 In-memory 파일 시스템 구현" , 충남대학교 졸업논문, 2002.