□특집□

리눅스에서의 이종 화일시스템 지원

정 갑 주† 김 동 욱††

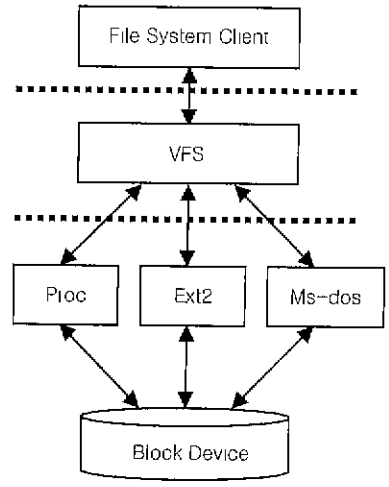
◈ 목 차 ◈

1. 서 론

91년 핀란드의 Linus Torvalds라는 한 대학원생에 의해 개발되기 시작한 PC 기반 운영체제 리눅스는 지난 10년 동안 눈부신 발전을 거듭하여 지금은 강력한 차세대 운영체제로 부각될 만큼 괄목할 발전을 이룩했다. 이러한 눈부신 발전의 배경에는 리눅스 운영체제가 다양한 하드웨어 기종 및 소프트웨어 제품들에 대해 개방적·포용적이도록 개발되어있다는 기술적 장점이 존재한다. 이러한 개방적 설계의 대표적 사례 중의 하나로 리눅스의 화일시스템을 들 수 있다. 리눅스 화일시스템의 가장 큰 장점은 MS-DOS, Windows NT, Windows 98, OS/2, Minix 화일시스템 등과 같은 다양한 이종 화일시스템(heterogeneous file systems)들이 손쉽게 동시에 지원될 수 있도록 설계되어 있다는 점이다.

본 글에서는 리눅스 화일시스템에서 이종 화일시스템 지원에 관해 설명한다. 글의 초점은 새로운 화일시스템이 지원될 때 고려해야 할 사항들에 주어진다. 리눅스 화일시스템은 널리 잘 알려진 유닉스 화일시스템을 기본으로 해서 설계되어 있다. 이 글에서는 독자들이 어느정도 유닉스 화일시스템에 대해 지식이 있다고 가정한다. 유닉스 화일시스템 전반에 관해서는 [1]을 참조하길 바란다.

본 글의 구성은 다음과 같다. 2절에서는 리눅스 화일시스템의 전반적인 구조에 대해서 설명한다. 3절에서는 이종 화일시스템 지원에서 핵심적인 역할을 하는 가상화일시스템(Virtual File System, 혹은 그냥 짧게 VFS)을 설명한다. 4절에서는 리눅스에서 기본 화일시스템으로 인정받고 있는 Ext2 화일시스템을 이종 화일시스템 지원 관점에서 간략히 설명한다.



(그림 1) 리눅스 화일시스템

† 정회원 : 건국대학교 컴퓨터공학과 조교수
†† 정회원 건국대학교 대학원 컴퓨터 정보통신 공학과 석사과정

2. 리눅스 화일시스템의 구조

리눅스 화일시스템의 구조는 (그림 1)에서와 같이 2개의 계층으로 구성된다. 각 계층들간에는 표준 인터페이스가 정의된다. 그림에서 점선들이 인터페이스를 나타낸다. 각 계층은 다음과 같다.

- 가상화일시스템 계층: 주기억장치에 위치하는 가상의 화일시스템으로 데이터저장 목적보다는 다양한 이종 화일시스템의 효과적 지원에 주 목적이 있다.
- 이종 화일시스템 계층: 다양한 이종 화일시스템들이 독립적으로 존재하는 계층으로 실제 디스크에서의 데이터 저장 및 관리 기능을 제공한다.

이러한 2단계 계층구조는 다음과 같은 관점에서 해석될 수 있다. 일반적으로 운영체제에서 화일의 데이터처리는 크게 두 단계 (디스크 단계 작업과 주기억장치 단계 작업)로 구분된다. 이러한 설계는 디스크의 데이터 액세스 속도가 주기억장치에 비해 매우 느리다는 데서 기인한다. 주기억장치 단계 작업은 주로 성능향상을 위한 캐싱 (혹은 버퍼링)에 초점이 주어지며, 주기억장치는 기본적으로 화일시스템의 임시 저장장소 역할을 하게 된다. 따라서 화일처리를 위한 주기억장치내의 자료구조들과 처리함수들은 일반적으로 특정 화일시스템에 종속적이게 된다.

이에 반해서, 리눅스는 주기억장치 처리 단계 작업을 독립적인 화일시스템(VFS)으로 구현한다. 이 VFS는 실제 데이터를 갖고 있지 않는 가상의 화일시스템으로 특정 화일시스템에 종속되지 않는 표준 응용인터페이스를 제공한다. 리눅스가 기본적으로 유닉스 표준을 따르는 것과 같이 이 인터페이스는 유닉스 화일시스템 응용인터페이스 표준을 따르고 있다. 이러한 설계를 통해 응용 시스템들이 특정 화일시스템에 종속되지 않고 개발될 수 있게 한다.

디스크 단계 작업에서는 기존의 여러 화일시스템들이 사용되는데 이들은 표준 통합인터페이스를 통해 VFS에 통합된다. VFS에서 제공되는 이 통합인터페이스는 통합절차를 표준화·단순화시켜 다양한 이종 화일시스템들이 용이하게 지원될 수 있게 한다.

3. 가상화일시스템 (VFS)

3.1 객체지향 설계

VFS는 이종 화일시스템 지원을 위해 객체지향 방식으로 설계되어 있다. 객체지향설계 기술들 중에서 데이터 추상화(data abstraction) 기법이 이종 화일시스템의 효과적 지원을 위해 활용된다. 데이터 추상화에서는 데이터와 처리함수들이 결합되어 제공되고, 데이터 사용자는 그 처리함수들을 통해서만 데이터에 접근하도록 되어 있다. 이 때 사용자들에게는 이 처리함수들에 대한 인터페이스 정보만 제공된다.

이러한 방식에서는 데이터 내부 자료구조와 처리함수들의 구현방법들이 사용자에게는 공개되지 않는다. 이를 정보 은닉 (information hiding)이라 하는데 이를 통해서 데이터 관리자는 사용자들에게 영향을 주지 않고 자료구조나 처리함수 구현방법을 변경할 수 있다.

이러한 데이터 추상화와 이를 통한 정보은닉은 사용자들이 이종의 데이터들을 동일한 방식으로 접근될 수 있도록 한다. VFS는 이러한 점을 이용해서 이종 화일시스템들을 효과적으로 지원한다. VFS는 객체지향언어로 구현되어 있지 않기 때문에 이러한 데이터 추상화을 완전하게 이용할 수 없다. 대신 그러한 장점을 최대한 이용하도록 구현되어 있다.

어떤 특정 화일시스템(예. Ext2)이 VFS를 통해

사용중이면 VFS내의 주요 자료구조들은 당연히 그 화일시스템에 의존적인 내용을 갖게 된다. 이들 내용을 처리할려면 처리함수들이 그 내용을 이해할 수 있어야 한다. 만약 VFS가 이들 처리함수를 제공하는 경우 VFS는 특정 화일시스템에 종속하게 된다. 이러한 문제를 해결하기위해서 VFS에서는 모든 주요 자료구조들이 자신들에 관련된 처리함수들에 관한 정보(즉, 함수포인터들의 테이블)를 포함하도록 설계되어 있고, 이들 함수들이 해당 화일시스템 코드에서 제공되도록 한다.

이러한 설계를 통해 VFS는 특정 화일시스템에 종속적인 내용을 직접 다룰 필요없이, 자료구조에서 기록된 관련 처리함수를 호출함으로써 처리한다.

VFS의 통합인터페이스는 핵심 자료구조들로 구성되며 새로운 화일시스템은 이 자료구조들에서 요구되는 정보와 코드를 제공함에 의해 손쉽게 추가될 수 있다.

3.2 화일시스템 타입 관리

이종 화일시스템을 논할 때는 화일시스템 타입과 화일시스템 객체를 구분해야 한다. 예를 들어, 화일시스템 타입은 화일시스템에 관한 일반적인 사항 (예, 자료구조 및 코드)을 의미하고, 화일시스템 객체는 실제 데이터를 관리하고 있는 특정 화일시스템을 의미한다. 전자는 C++에서의 Class에 해당되고 후자는 Instance에 해당된다고 볼 수 있다. 예를 들면, Ext2에 해당되는 화일시스템 타입은 하나이지만 실제 Ext2 화일시스템 객체는 여러개가 존재한다. 이후에는 화일시스템 객체를 구분이 필요한 경우가 아니면 그냥 화일시스템이라 부른다.

VFS는 여러 이종 화일시스템 타입을 지원하기 위해서, 지원되는 화일시스템 타입들의 목록을 관리한다. 이러한 목록 정보는 file_systems[8]라 불리우는 선형 리스트에 관리된다. 각 노드는 file_system_type[9]구조체로 정의되며, 특정 화일시스템 타입을 나타낸다. 이 구조체의 주요 필드는 <표 1>과 같다.

(표 1) file_system_type 노드

name	화일시스템 이름
read_super	super block를 VFS로 읽어오는 함수

name 필드에는 화일시스템 타입 명이 저장된다. read_super 필드는 함수 포인터 형으로 선언되어 있으며, 화일시스템 객체를 VFS로 마운트 시킬 때 호출되는 함수를 가리킨다. 마운트는 기본적으로 해당 화일시스템에 관한 핵심정보를 VFS로 읽어오는 작업을 의미한다.

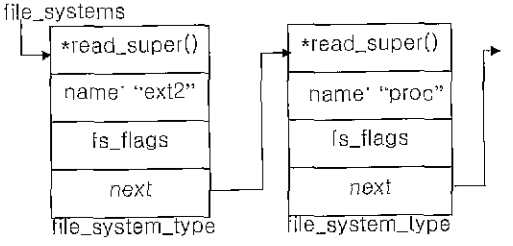
VFS에서는 동적으로 새로운 화일시스템 타입을 등록할 수 있다. 이때 이 화일시스템 코드 (예를 들어, Ext2 화일시스템 코드)가 이미 모듈로서 존재해야 하는데, 등록시 이 코드가 커널로 로드된다. 이 코드에는 read_super 필드에 기록될 함수가 존재해야 한다. 이러한 등록은 register_filesystem 함수[8]를 호출함에 의해 이루어 진다. (그림 2)는 file_systems 자료구조를 묘사하고 있다.

화일시스템 마운트 작업은 화일시스템 내부구조에 의존하는 작업인데 VFS에서는 이러한 내부구조에 대한 정보없이 read_super 필드에 기록된 함수를 호출함에 의해 여러 이종 화일시스템들을 동일한 방법으로 마운트한다.

3.3 화일시스템 마운트

3.2에서 설명된 화일시스템 타입 등록은 기본적으로 해당 화일시스템 코드를 VFS에 등록하는 것이라 볼 수 있다. 이에 반해 화일시스템 마운트는 실제 데이터를 관리하고 있는 화일시스템 객체를 등록하는 것으로 볼 수 있다. 구체적으로 그 화일시스템 내의 데이터를 사용자가 접근할 수

있게 하는 것이다. VFS (보다 구체적으로 버퍼)에는 현재 액세스중이거나 최근에 액세스된 데이터들만 존재하기 때문에 마운트 시점에서는 그 화일시스템에 관한 핵심 관리정보 (수퍼블럭)만을 읽어온다. VFS는 화일시스템 타입 목록 리스트 file_systems에서 해당 화일시스템의 file_system_type 노드를 찾고, 그 노드에 기록된 read_super 함수를 호출하여 수퍼블럭을 읽어온다.



(그림 2) 화일시스템 타입 목록

VFS는 마운트된 화일시스템들의 수퍼블럭 정보를 super_blocks[9]라 불리우는 선형 리스트로 관리한다. super_blocks에서 각 노드는 구조체 super_block[9]으로 정의되어 있고, 특정 화일시스템 객체의 수퍼블럭 정보를 갖고 있다. 구조체 super_block의 주요 필드들은 <표 2>에서 제시된다.

(표 2) super_block 구조체

s_dev	화일시스템 저장장치의 장치번호 (device number)
s_blocksize	저장장치의 블럭크기
s_type	file_systems내의 해당 타입노드에 대한 포인터
s_time	마운트 시간
s_root	root 디렉토리
s_op	수퍼블럭을 위한 처리함수들에 대한 포인터

s_op 필드에는 수퍼블럭을 위한 처리함수들이 기록된다. 이들 함수는 실제 해당 화일시스템 코드에 존재하고, s_op 필드에는 함수포인터들만 기록된다. VFS는 이 필드에 기록된 함수들을 단순히 호출함에 의해 이종 화일시스템들을 동일한 방법으로 관리할 수 있다. s_op 필드는 구조체 super_operations[9]로 정의되는데 <표 3>에 일부 필드들이 소개된다.

(표 3) 구조체 super_operations

read_inode	inode 읽어오기
write_inode	inode 저장
put_inode	inode cache 삭제
delete_inode	inode 삭제
notify_inode	inode 속성변경 처리
put_super	super block 해제
write_super	super block 저장
statfs	화일시스템 정보얻기
remount_fs	화일시스템 remount

수퍼블럭 정보외에 VFS는 각 화일시스템 객체들에 대해서 마운트 위치정보도 관리한다. VFS는 선형리스트 vfsmntlist[8]에 모든 마운트 위치에 관한 정보를 관리한다. 각 노드는 특정 마운트에 관한 정보를 기록하는데, 구조체 vfsmount[10]로 나타낸다. <표 4>에서 주요 필드를 보여준다.

(표 4) vfsmount 구조체 필드들

mnt_dev	저장장치 번호
mnt_devname	저장장치 이름
mnt_dirname	마운트되는 디렉토리 위치
mnt_sb	해당 super_block의 포인터

3.4 화일 관리

리눅스에서는 내부적으로 화일을 inode로서 나타낸다. 따라서 화일의 관리는 inode를 통해 수행된다. inode들은 원래 디스크내에 상주하고 화일시스템이 관리한다. VFS는 이런 디스크 상주 inode에 대응해서 주기억장치 상주 inode 구조체

[9](이후, VFS inode라 부른다)를 갖고 있다.

이 VFS inode 구조체는 VFS를 위한 관리정보들과 디스크 상주 inode에 저장되어 있는 정보들을 포함하고 있다. <표 5>는 inode 구조체 내의 주요 필드들을 간략히 소개하고 있다.

(표 5) VFS inode 구조체

i_ino	inode 번호
i_count	참조 횟수
i_dev	저장장치 번호
i_mode	액세스 모드
i_uid	소유자의 id
i_gid	그룹 소유자 id
i_size	화일크기
i_atime	마지막 접근시간
i_blksize	블럭크기
i_sb	super_block 포인터
i_op	inode 처리함수들

i_op에는 VFS inode 관리를 위해 요구되는 처리함수들에 대한 정보가 저장되며, i_op의 형은 구조체 inode_operations로 정의되는데, <표 6>에 주요 필드들이 소개된다. default_file_ops 필드에는 3.5에서 설명될 오픈 화일을 액세스할 때 사용되는 함수들이 기록된다.

inode_operations에서 요구되는 함수들은 해당 화일시스템코드에서 제공되어야 하고, VFS가 외부로부터 작업 요청(시스템 함수 호출)을 받았을 때 호출된다.

(표 6) 구조체 inode_operations

default_file_ops	오픈 화일 처리함수들
create	화일 생성
lookup	화일 탐색
mkdir	디렉토리 생성
rmdir	디렉토리 삭제
readpage	페이지 read
writepage	페이지 write

VFS에서는 현재 사용중이거나 최근에 사용되었던 VFS inode들을 inode_hashtable이라 불리우는 해쉬 테이블에 보관한다. 이 테이블에서 inode를 저장하고 검색할 때 저장장치 번호와 inode 번호가 키로서 사용된다. 이 테이블은 inode 캐쉬의 역할을 수행한다.

3.5 화일 액세스

여러 프로세스가 동일한 화일에서 동시에 서로 영향을 주지않고 작업을 하기 위해서, VFS는 프로세스가 화일을 오픈할 때 그 화일의 inode와는 별도로 구조체 file[9]을 할당한다. 이 구조체에는 그 프로세스의 화일 액세스 관련정보(오픈 모드, 액세스 위치 등)가 저장된다. 프로세스별로 오픈한 화일들에 대해서 별도로 file 구조체 테이블들이 유지된다.

(표 7) file 구조체

f_mode	접근권한
loff_t	현재 액세스 위치
f_node	해당 화일의 inode 구조체에 대한 포인터
f_op	화일 액세스를 위한 함수들의 포인터

화일에서의 실제 입출력을 위해서 f_op필드에 기록된 함수들이 사용된다. f_op 필드는 구조체 file_operations[9]로 정의된다. 이 구조체의 주요 필드들이 <표 8>에 제시된다.

(표 8) file_operations 구조체

lseek	화일내 특정위치로 이동
read	데이터 읽기
write	데이터 쓰기
readdir	디렉토리 읽기
ioctl	open 화일 제어 함수
mmap	메모리 매핑함수
open	화일 open 함수
release	사용이 해제된 화일
fsync	메모리 캐쉬와 디스크 동기화

3.6 요약

지금까지 VFS와 이종 화일시스템들간의 통합 인터페이스에 관해서 설명했다. 이 인터페이스는 통합시에 사용되는 자료구조들과 처리함수들로 구성된다.

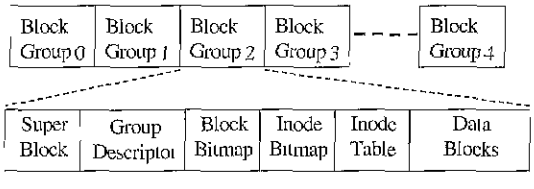
이 인터페이스는 대표적으로 super_block, inode, file 등과 같은 자료구조와 super_operations, inode_operations, file_operations 구조체들에서 지정되는 처리함수들로 구성된다. 이 인터페이스는 새로운 이종 화일시스템을 리눅스에 추가할 때 요구되는 정보들을 정의한다.

4. Ext2(Second Extended File System)

이종 화일시스템의 구체적 예로서 리눅스 표준 화일시스템으로 인정받고 있는 Ext2 화일시스템을 설명한다. Ext2의 기본 설계와 VFS와의 연동 방법에 초점이 주어진다.

4.1 Ext2 화일시스템의 구조

저장장치에 존재하는 Ext2 화일시스템의 구조는 블럭그룹들의 열로써 구성되어져 있고, 다시 각 블럭그룹은 수퍼블럭, 그룹 설명자 테이블, inode 테이블, 데이터 블럭들로 구성된다. 이 구조에서 수퍼블럭은 매 블럭그룹마다 나타나지만 이는 수퍼블럭의 손상 시 복구를 위해 복제된 것이고 논리적으로는 한 개의 수퍼블럭이 존재하는 것이다. (그림 3)는 Ext2의 구조를 나타낸다.



(그림 3) Ext2 화일시스템의 구조

4.2 Ext2 수퍼블럭

Ext2에서 화일시스템 관리에 핵심적인 정보는 수퍼블럭에 저장된다. 이 정보는 화일시스템을 유지하고 운용하는데 매우 중요하다. 수퍼블럭의 손상은 화일시스템의 모든 데이터를 더 이상 사용할 수 없게 만드는 치명적인 결과를 낳을 수 있다. 이러한 이유에서 Ext2 화일시스템에서는 모든 블럭그룹에 수퍼블럭을 중복시켜 손상에 대비한다. 일반적으로 첫 블록그룹 (블럭그룹 0)에 존재하는 수퍼블럭이 화일시스템 마운트 시에 사용된다. 최근에 개발된 Ext2 시스템에서는 중복성을 줄이기 위하여 몇 개의 블럭그룹에만 수퍼블럭이 복제되기도 한다.

VFS와의 통합을 위해 Ext2 화일시스템 코드에는 Ext2 수퍼블럭을 VFS super_block으로 읽어들이는 함수 ext2_read_super[13]와 file_system_type 형의 자료구조 ext2_fs_type[13]가 제공된다. ext2_fs_type 변수는 등록을 위해 register_filesystem 함수가 호출될 때 사용된다. 다음에서 이 과정에 관련된 코드를 보인다.

```
static struct file_system_type ext2_fs_type = {
        "ext2",
        FS_REQUIRES_DEV,
        ext2_read_super,
        NULL
};

int init_ext2_fs(void) {
return
        register_filesystem(&ext2_fs_type);
}
```

Ext2 화일시스템에서 VFS에서 요구되는 수퍼블럭 관련 처리함수들은 super_operations 구조체 변수 ext2_sops[13]에 기록되어 있다. 이 변수는 ext2_read_super 함수에서 Ext2 수퍼블럭 정보를 VFS super_block에 기록할 때 사용된다.

4.3 그룹 설명자

각 블럭그룹에 대한 제어정보를 저장하기 위하여 그룹 설명자가 사용된다. 수퍼블럭이 전체 화일시스템에 관련된 정보를 표시하는 반면에 그룹 설명자는 각 블럭그룹에 관한 사용상태 정보를 표시한다. 그룹 설명자는 블록들의 사용상태를 나타내는 Block Bitmap, inode들의 사용상태를 나타내는 inode Bitmap 등으로 구성된다.손상에 대비하기 위해 그룹 설명자도 각 블럭그룹에 중복된다.

4.4 Ext2 inode

inode는 화일을 나타내는 객체로 Ext2 화일시스템에서 가장 기본적인 개념이다. inode는 해당 화일에 관한 관리 정보 (접근 권한, 소유권, 블럭크기, 저장장치 번호, 크기, 수정시간 등)와 데이터가 저장되어 있는 디스크 블럭들에 대한 포인터를 갖고 있다.

Ext2 화일시스템에서는 inode에서 나타내는 화일의 타입(디렉토리, 화일, link)에 따라서 별도의 inode 관리함수들이 사용된다. 이들 관리함수들은 통합을 위해 VFS에 제공되어야 한다.

Ext2 화일시스템은 화일의 타입에 따라 별도의 inode_operations 형 변수들(ext2_dir_inode_operations,ext2_file_inode_operations,ext2_symlink_inode_operations)을 제공한다. inode를 VFS로 읽어들이는 함수 ext2_read_inode (ext2_sops에 기록)는 화일 타입을 조사해서 해당되는 변수를 선택한다.

4.5 화일 액세스 관련 함수들

오픈 파일을 관리하는 구조체 file_operations에 정의되어지는 함수는 디렉토리로 구현된 계층적 트리에서 원하는 파일탐색에 관련된 함수와 화일내에 저장되어있는 데이터의 조작에 관련된 함수로 구분되어진다. 이 구분에 따라 변수 ext2_file_operations과 ext2_dir_operations [14,15]로 관리함수들이 분류되어 정의되어지고, VFS상에서 inode가 나타내는 객체의 종류에 따라 두 변수중의 하나가 정의되어지는 것이다.

4.6 요약

Ext2 화일시스템은 VFS 통합인터페이스에서 요구되는 정보들을 VFS 자료구조체 형의 변수들로 제공한다. Ext2 화일시스템의 경우 VFS와 논리적 구조가 유사해서 통합을 위한 절차가 단순하다.

리눅스에서 지원되는 다른 화일시스템들의 경우도 VFS와의 통합을 위해 이와 비슷하게 정보들이 제공된다. 그러나 VFS와는 구조가 많이 다른 화일시스템의 경우 (예를 들면, MS-DOS 화일시스템)는 필요에 따라서는 그 화일시스템에는 없는 정보도 가상으로 만들어서 제공을 해야 할 필요도 있다. 이는 리눅스에서 이종 화일시스템 지원이 VFS를 중심으로 이루어지기 때문이다.

5. 결 론

본 글에서 우리는 리눅스 화일시스템이 이종 화일시스템을 효과적으로 지원하기 위해 어떻게 설계되었는 지를 살펴 보았다. 개념적 설명보다는 새로운 화일시스템의 추가에 관심이 있는 사람들에게 필요한 구체적 정보의 제공에 초점이 주어졌다.

국내에서도 최근 리눅스에 대한 관심이 매우 높아졌다. 그러나 안타깝게도 외국에 비해 리눅스 운영체제 개발과정에 국내 컴퓨터 전문가들의 참여는 아직 매우 미비한 편이다. 하루 빨리 국내에서도 리눅스 관련 소프트웨어들이 왕성하게 개발되길 바라면서, 본 글이 조금이나마 그러한 관점에서 도움이 되었으면 한다.

참고문헌

[1] Maurice J Bach, The Design of the UNIX Operating System, Prentice-Hall, Inc., 1986.

[2] Michael Beck, Harald Bohme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus, Linux Kernel Internals, Addison-Wesley, 1998.

[3] Remy Card, Eric Dumas, Franck Mevel, The Linux Kernel Book, John Wiley & Sons, Inc., 1998. 12.

[4] Michael K. Johnson, Design and Implementation of the Second Extended Filesystem, http://kldp.org/LDP/LDP/khg/HyperNews/get/fs/vfstour.html.

[5] Michael K. Johnson, Linux Kernel Hacker's Guide, Linux Document Project, 1995.

[6] Michael K. Johnson, A Tour of the Linux VFS, http://kldp.org/LDP/LDP/khg/HyperNews/get/fs/ext2intro.html.

[7] David A Rusling, The Linux Kernel, Linux Document Project, 1999. 1.

[8] fs/super.c

[9] include/linux/fs.h

[10] include/linux/mount.h

[11] fs/inode.c

[12] include/linux/sched.h

[13] fs/ext2/super.c

[14] fs/ext2/file.c

[15] fs/ext2/dir.c

[16] fs/ext2/symkink.c

정 갑 주

1984년 서울대학교 공과대학 컴퓨터공학과 졸업(공학사)
1986년 서울대학교 대학원 컴퓨터공학과 졸업(공학박사)
1996년 New York University, Department of Computer Science (전산학 박사)
1995년-1996년 University of Florida, Postdoc
1996년-1997년 Cornell University, Visiting Fellow
1997년-현재 건국대학교 컴퓨터공학과 조교수
관심분야 : 분산컴퓨팅, 리눅스 운영체제, 고장허용(Fault Tolerance), 병렬화일시스템

김 동 욱

1999년 안양대학교 공과대학 컴퓨터학과 졸업(공학사)
1999년-현재 건국대학교 대학원 컴퓨터·정보통신공학과 석사과정
관심분야 : 리눅스 운영체제, 분산컴퓨팅, 병렬화일시스템