리눅스 커널의 특징

한국전자통신연구원 손성훈

1. 서 론

1991년 최초의 버전이 공개된 이래 리눅스 운영체제는 십여 년이라는 짧은 기간에도 불구하고 매우 빠른 속도로 발전해 왔다. 현재의 리눅스는 개인용 데스크탑 운영체제에서부터 대규모 인터넷 서비스를 위한 서버의 운영체제나 특수한 목적으로 사용되는 내장형 시스템(embedded system)을 위한 운영체제에 이르기까지 여러 분야에서 두루 사용되고 있으며, 각 응용 분야에서 기존에 사용되고 있는 상용 운영체제들에 대한 강력한 대안으로 떠오르고 있다.

리눅스가 이러한 성과를 거둘 수 있었던 가장 큰 원동력은 소스 코드의 공개라는 원칙 하에 전 세계의 수많은 프로그래머들이 운영체제 커널과 응용 프로그램의 개발에 자발적으로 참여해 왔다는 점이다. 커널의 소스 코드가 공개된 상태에서 성능 향상을 위한 기존 코드의 변경이나 새로운 기능의 추가가 비교적 쉽게 이루어질 수 있었고, 한 개발자에 의해 새로 추가된 기능은 다른 개발자들에 의해 반복된 검증과 수정을 거치면서 더욱 안정된 커널로 발전할 수 있었다. 또한 여러 개발자의 공동 노력으로 하드웨어의 다양하고도 빠른 변화에 쉽게 대응할 수 있었다.

더불어 리눅스 발표와 비슷한 시기에 급격히 확산된 인터넷은 이러한 공동 작업의 효과적인 수단이 되어 리눅스의 발전에 크게 기여하였다. 최초의 개발자인 리누스 토발즈가 첫 버전을 공개한 것도 인터넷을 통해서였고, 이후 수많은 개발자들이 인터넷을 매체로 개발에 참여하고 의견을 주고받아 왔다. 이는 리눅스가 배포되고 사용되는 과정에서도 그대로 이어져 리눅스는 어느 누구라도 인터넷을 통해 무료로 얻어 자유롭게 사용할 수 있는 몇 안 되는 운영체제가 되었다.

그러나 이와 같은 리눅스만의 독특한 발전 과정은 소스 코드를 읽거나 커널의 구조를 이해하고자 하는 측면에서는 좋지 않은 영향을 미친 것도 사실이다. 리눅스는 사용자 수의 증가 못지 않게 커널 코드의 크기도 급격히 증가해 왔다. 1996년에 발표된 커널 버전 2.0만 해도 이미 커널 소스 코드의 크기가 C 언어 코드 50여 만 라인과 수 천 라인의 어셈블리어 코드 수 천 라인의 분량이었고, 최근의 커널인 버전 2.2에서는 전체 커널 소스 코드의 크기가 약 150만 라인 가량이나 된다. 이러한 방대한 코드들이 여러 개발자에 의해 다소 원칙 없이 작성되는 과정에서 코딩의 일관성이 결여되고 일부 개발에 관련된 문서가 충분히 제공되지 않는 등 소스 코드를 이해하고 수정하는데 어려움이 따르는 것도 사실이다.

본 원고에서는 리눅스 운영체제에 대한 이해를 돕기 위해 기능적, 구조적인 관점에서 커널의 각 서브시스템들의 특징을 살펴보고자 한다. 실제의 리눅스 커널은 원래 의도했던 바와 같이 기능상 기존의 유닉스 커널과 매우 흡사하다. 따라서 커널의 구조 또한 유닉스와 유사하기 때문에 유닉스 커널의 내부 구조와 기능에 익숙한 사람이라면 리눅스 커널을 이해하는데 별 어려움이 없을 것이다. 따라서 본 고에서는 일반적인 유닉스 커널과 유사한 부분보다는 리눅스 커널의 특징적인 면을 위주로 다루고자 한다.

2. 주기억장치 관리

리눅스는 사용자 프로세스에게 실제 물리적인 주소 공간보다 넓은 주소 공간을 제공하기 위하여 가상 기억장치 기능을 제공한다. 이를 위해 물리적인 주소 공간과 각 프로세스의 가상 주소 공간은 일정한 크기의 페이지 단위로 구성되며, 요구 페이징 알고리즘을 사용하여 필요할 때마다 원하는 페이지만을 주기억장치에 적재한다. 새로운 페이지의 적재를 위한 공간을 마련하기 위해서 LRU 알고리즘을 따르는 스와핑을 사용한다.

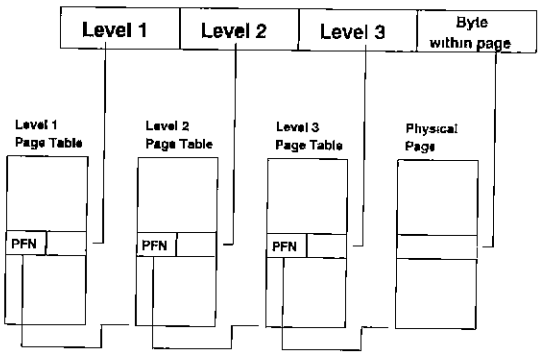

그림 1 리눅스에서의 가상 주소의 변환

운영체제의 메모리 관리는 일반적으로 특정 하드웨어의 MMU와 매우 밀접한 관련을 가지고 있다. 리눅스의 메모리 관리 서브시스템은 다양한 하드웨어 플랫폼을 지원하기 위하여 가상 기억장치를 포함한 메모리 관리 모듈을 하드웨어 독립적인 부분과 종속적인 부분으로 분리하여 설계, 구현하였다. 따라서 리눅스를 새로운 하드웨어 플랫폼으로 이식하는 경우 새로운 프로세스의 MMU에 종속적인 모듈만 수정하면 쉽게 이식할 수 있다. 이 장에서는 리눅스의 장치 독립적인 메모리 모델에 대해 주로 살펴보고자 한다.

리눅스의 장치 독립적인 메모리 모델에서는 프로세스가 만들어 내는 가상 주소의 변환에서 세 단계의 페이지 테이블이 존재하는 것을 가정한다(그림 1). 리눅스가 지원하는 하드웨어 플랫폼마다 매크로를 제공하여 커널이 특정 페이지 테이블 구성과 주소 변환 과정에 관계 없이 동작할 수 있도록 하였다. 그리하여 세 단계의 페이지 테이블을 사용하는 알파 프로세서와 두 단계의 페이지 테이블만 존재하는 x86 계열의 프로세서 모두 페이지 테이블을 조작하는 코드는 동일하다.

이 밖에도 리눅스의 메모리 관리 시스템은 가상 기억장치나 파일 시스템 입출력 등과 같이 디스크 연산의 성능 향상을 위해 블록 장치 드라이버를 위한 버퍼 캐쉬, 새로 주기억장치에 적재된 페이지의 재사용을 위한 페이지 캐쉬, 불필요한 스와핑을 줄이기 위한 스왑 캐쉬 등을 사용한다.

3. 프로세스 관리

이 장에서는 리눅스 커널이 어떤 식으로 프로세스들을 관리하는지에 대해 기술한다.

리눅스 커널은 내부적으로 각 프로세스마다 task_struct라고 하는 자료 구조를 두어 프로세스를 관리한다. 또한 전체 task_struct들은 task라고 하는 고정된 크기의 배열을 통해 유지되며, 이 task라는 배열은 유닉스에서의 프로세스 테이블에 해당한다고 할 수 있다. 따라서 이 테이블의 크기에 따라 전체 시스템에서 동시에 존재할 수 있는 프로세스의 개수가 결정된다(이 테이블 크기의 기본 값은 512이다).

task_struct 자료 구조는 커널이 해당 프로세스를 관리하기 위해 필요한 모든 정보를 가지고 있다. 예를 들어 이 자료 구조에는 프로세스의 식별자, 현재 상태, 스케줄링 정보, 다른 프로세스들과의 관계를 나타내는 포인터, 현재 작업 디렉토리나 루트 파일 시스템에 대한 i-node 포인터 등의 파일 시스템 관련 정보, 사용 중인 파일에 대한 포인터와 같은 파일 관련 정보, 사용자 모드와 커널 모드에서 수행한 시간, 문맥 교환 시 레지스터나 스택을 저장할 수 있는 영역, 할당된 메모리 영역에 대한 포인터 등이 담겨 있다.

리눅스 스케줄러는 전통적인 유닉스 스케줄러처럼 대화형 프로세스를 선호하는 스케줄링을 통하여 사용자에게 짧은 응답 시간을 제공하는 것을 목표로 하고 있다. 각 프로세스마다 한 번 CPU가 주어지는 시간 할당은 200ms이고, CPU를 사용 중인 프로세스는 이 시간 내에 시스템 호출을 통해 스스로 대기 상태로 전환되거나 커널에 의해 강제로 CPU를 빼앗기게 된다.

실제 스케줄러는 커널 내부의 여러 지점에서 수행될 수 있다(그림 2). 우선 스케줄러는 현재 수행 중이던 프로세스가 시스템 호출을 통해 대

기 큐에 들어간 직후에 수행된다. 또한 인터럽트 처리 루틴을 수행하고 난 이후에나, 프로세스가 시스템 호출을 마치고 커널 모드에서 사용자 모드로 복귀하기 직전에도 스케줄러가 수행된다. 이 때마다 수행 큐에서 CPU 할당을 기다리고 있는 프로세스들에 대해 task_struct의 우선 순위와 카운터 값을 이용해 다음 번 수행할 프로세스를 선택하게 된다.

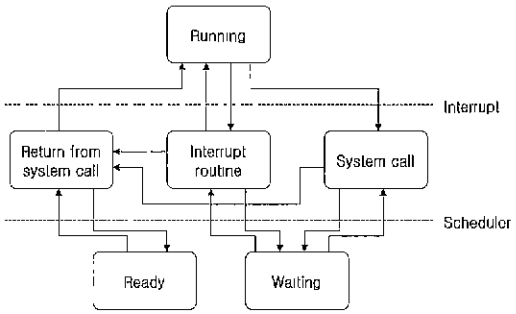

그림 2 프로세스 상태 전이와 스케줄링

리눅스 커널의 프로세스 스케줄링에서 특징적인 면은 앞서 언급한 전통적인 유닉스 방식의 스케줄링 외에도 실시간 클래스 프로세스를 위한 스케줄링도 지원한다는 점이다. 실시간 프로세스들은 다른 일반적인 프로세스보다 항상 높은 우선 순위를 가지고 수행되며, task_struct에 지정된 스케줄링 정보에 따라 FIFO와 라운드 로빈 중 하나의 실시간 스케줄링 정책이 적용된다. 아울러 커널 2.0 이후의 리눅스에서는 다중 처리기 시스템을 지원하는 SMP 버전의 커널도 개발되었다.

리눅스 커널은 프로세스간 통신을 위한 여러 가지 수단을 제공하고 있다. 기존의 유닉스에서 제공되는 대표적인 프로세스간 통신 기법들인 시그널, 파이프, ptrace 시스템 호출 등이 거의 동일한 환경으로 제공된다. 또한 메시지 큐, 세마포어, 공유 메모리 등과 같은 System V 계열의 프로세스간 통신 기법들도 제공된다.

4. 파일 시스템

잘 알려진 바와 같이 초기의 리눅스 커널은 MINIX 상에서 개발되었고, 당시에는 MINIX 파일 시스템을 그대로 사용하였다. 이후 Xiafs, Ext 등 리눅스를 위한 몇몇 전용 파일 시스템이 등장하였고, 최근에는 Ext2(The Second Extented) 파일 시스템이 리눅스의 대표적 파일 시스템으로 사용되고 있다.

Ext2 파일 시스템은 그 설계에 있어서 BSD 유닉스의 FFS(Fast File System)로부터 많은 영향을 받았다. 예를 들어, Ext2 파일 시스템에서는 FFS에서의 실린더 그룹과 유사한 개념으로 블록 그룹(block group)을 사용한다. 즉, 하나의 Ext2 파티션은 여러 개의 블록 그룹(block group)이라는 단위로 나누고, 각 블록 그룹마다 수퍼 블록의 복사본, i-node 테이블, 데이터 블록이 위치한다. 이렇게 함으로써 한 파일의 i-node와 해당 데이터 블록, 또는 파일 i-node와 관련된 디렉토리의 i-node 등을 같은 블록 그룹에 위치 시켜 디스크 상에서 물리적으로 가까운 위치에 둠으로써 디스크 접근 시 탐색 시간을 최소화하여 파일 입출력의 성능 향상을 꾀하고 있다. 또한 중요한 파일 시스템의 가장 중요한 정보인 수퍼 블록의 복사본을 여러 개 둠으로써 유사시 파일 시스템을 쉽게 복구할 수 있다. Ext2 파일 시스템은 현재에도 개발이 진행되고 있으며, 앞으로도 삭제된 파일의 복구, 자동적인 파일 압축 등과 같은 새로운 기능들이 추가될 것으로 알려져 있다.

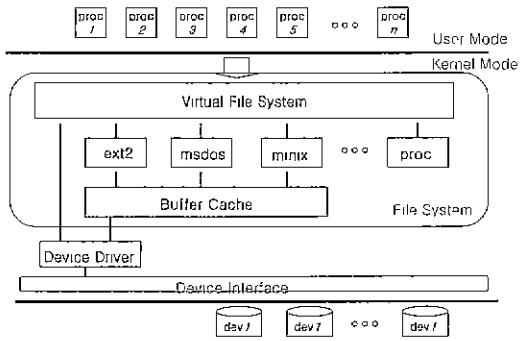

그림 3 Virtual file system

리눅스 파일 시스템의 또다른 특징으로는 Virtual File System(VFS)의 사용을 들 수 있다. VFS는 디스크 상에 존재하는 물리적인 파일 시스템이라고 하기 보다는 여러 종류의 파일 시스템들을 VFS가 제공하는 일관된 인터페이스

를 통해 사용할 수 있도록 해주는 사용자와 파일 시스템 간의 계층이라고 할 수 있다(그림 3). VFS는 일반적인 파일 시스템들이 가지고 있는 수퍼 블록이나 i-node의 구조를 가지고 있으며, 특정 파일 시스템이 마운트될 때마다 VFS와 마운트된 파일 시스템 간의 연결 정보를 유지함으로써, 사용자가 VFS의 인터페이스를 통해 여러 파일 시스템을 쉽게 사용할 수 있도록 해준다.

그밖에도 리눅스 파일 시스템은 Proc 이라는 독특한 파일 시스템을 사용하고 있다. Proc 파일 시스템은 디스크에 데이터를 저장하기 위한 파일 시스템이라고 하기 보다는 현재 동작 중인 하드웨어 정보나 사용자 프로세스들에 대한 정보를 쉽게 얻을 수 있도록 하는 가상의 파일 시스템이다. 리눅스 커널은 실제 내부 변수의 형태로 가지고 있는 이러한 정보들을 Proc 파일 시스템 하의 파일로 만들어 기록해 놓는다. 또 새로운 프로세스가 만들어질 때마다 프로세스 번호와 동일한 이름의 디렉토리를 만들고 그 아래 프로세스에 대한 정보를 파일로 기록한다. 이러한 방식을 통해 해당 파일을 읽어들임으로써 동적으로 변하는 자원 사용 상태 정보, 각종 하드웨어 상태 정보, 프로세스 관련 정보 등을 쉽게 얻을 수 있다. 예를 들어 프로세스들의 상태를 보여주는 ps 라는 명령어는 단순히 Proc 파일 시스템의 파일로부터 프로세스 관련 정보를 읽어 출력하는 형태로 구현되어 있다.

5. 기타

리눅스와 같은 모노리딕 커널의 한 가지 단점은 간단히 커널을 수정하는 경우나 일부 시스템의 구성을 바꾸는 경우에도 매번 전체 커널을 다시 컴파일해야 하는 단점을 가지고 있다. 이러한 번거로움을 피하기 위해서 리눅스는 모듈이라는 개념을 제공하고 있다. 모듈은 실행 중에 커널에 동적으로 연결하거나 제거할 수 있는 오브젝트 코드를 말한다. 자주 사용되지 않는 디바이스 드라이버나 파일 시스템 코드 등은 모듈로 구성해 필요한 경우에만 커널에 포함시켜 사용함으로써 커널의 크기가 불필요하게 커지는 것을 막을 수 있다. 또한 새로 개발하여 테스트 중인 커널 기능은 모듈로 구성함으로써 테스트 과정에서 여러 번 재부팅 해야하는 번거로움을 피할 수 있다.

시스템 운용 중에 동적으로 모듈을 사용하기 위해서는 사용자가 insmod/rmmod와 같은 명령어를 사용하여 필요할 때마다 명시적으로 추가, 제거할 수도 있고, 커널 데몬(kerneld)에 의해 필요시마다 자동적으로 추가와 삭제를 수행할 수도 있다. 이때 커널 데몬은 커널로부터 IPC를 통해 모듈의 추가나 제거가 필요한 시점을 알게 된다. 커널은 사용 중인 각 모듈의 심볼 테이블에 대한 리스트를 유지하고 있다(그림 4). 이 심볼 테이블을 통해 다른 커널 루틴이 모듈 내의 루틴과 자료 구조를 참조할 수 있으며, 반대로 모듈도 커널의 심볼 테이블을 참조해 기존의 커널 루틴을 호출할 수 있다. 이와 같이 일단 적재된 모듈은 완전히 커널의 일부로 동작하게 된다.

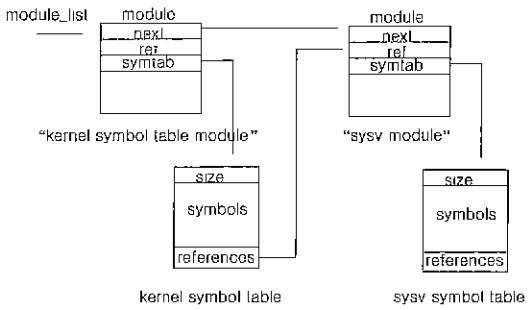

그림 5 커널 모듈의 구조

6. 맺음말

이제까지 리눅스 커널의 특징적인 면을 위주로 살펴보았다. 사실 현재의 리눅스는 그 구조나 기능 면에서 기존의 유닉스와 별반 차이가 없다. 오히려 유닉스에 비해 20 여년이나 뒤늦게 출발한 운영체제로서 아직은 뒤떨어지는 면이 많다고 할 수 있다. 다만 서론에서도 간단히 언급한 바와 같이 리눅스가 지향하고 있는 오픈 소스의 정책은 앞으로의 리눅스에 대한 전망을 밝게 해 주고 있다. 지금까지의 유닉스는 오랜 세월을 거치면서도 여러 가지 제약들로 인해 발전 속도가 그다지 빠르지 않았다. 반면 리눅스는 커널의 소스 코드가 공개되어 있다는 면에서 그 자체의 발전에도 크게 유리할 뿐만 아니라, 여러 개발자에게 이를 응용할 수 있는 기회를 제공하고 있다. 리눅스는 당분간 시스템 소프트웨어와 관련된 연구,

개발 주제로 많은 관심을 끌 것으로 예상된다.

참고문헌

[1] M. Beck. et. al, Linux Kernel Internals 2nd Ed.. Addison-Wesley
[2] R. Card, The Linux Kernel Book, John Wiley & Sons.
[3] A. Rubini, Linux Device Drivers, O'reilly.
[4] David A. Rusling, The Linux Kernel, Linux Documentation Project.

손 성 훈

1991 서울대학교 계산통계학과 졸업
1993 서울대학교 대학원 전산과학과 졸업(이학석사)
1999 서울대학교 대학원 전산과학과 졸업(이학박사)
1999 ~ 현재 한국전자통신연구원 선임연구원
E-mail: sonsh@etri.re.kr

2000년 정례회의 및 주요행사 연간일정표

월 별	이 사 회		총회 · 학술발표회		송년회
	상 임	정 례	임시 · 춘계	정기 · 추계	
1월	14일(금) 16:00				
2월	14일(월) 16:00	18일(금) 16:00			
3월	10일(금) 16:00				
4월	7일(금) 16:00	21일(금) 16:00	28일(금)~29일(토) 대구효성가톨릭대		
5월	12일(금) 16:00				
6월	9일(금) 16:00	23일(금) 16:00			
7월	7일(금) 16:00				
8월	4일(금) 16:00	25일(금) 16:00			
9월	8일(금) 16:00				
10월	6일(금) 16:00	20일(금) 16:00		27(금)-28(토)	
11월	3일(금) 16:00				
12월	1일(금) 16:00	15일(금) 16:00			12일(화) 18:00

※ 회의일정은 사정에 따라 변경될 수 있음.