PART I

# ARM 리눅스 커널

## 커널 분석을 위해 어떤 준비가 필요할까?

리눅스 커널은 압축된 소스 코드만 70MB 이상이다. 이런 방대한 소스 코드를 라인 단위로 따라가며 분석한다는 것은 많은 노력과 고통이 수반되는 과정이다. 하지만 우리가 쏟아 부어야 할 노력이라는 것이 어찌 보면 우리가 가지고 있는 리눅스 커널에 대한 지식, ARM 프로세서에 대한 이해, 소스 분석용 툴(tool)들에 대한 경험과 반비례할지도 모른다. 즉, 우리가 이런 지식과 경험을 충분히 가지고 있다면 소스를 분석하는 시간은 많이 줄어들 것이다. 따라서 우리는 본격적인 소스 코드 분석을 위해 준비운동을 해야 한다.

이번 파트에서는 커널 소스 분석을 위한 준비단계로 리눅스 커널에 대한 소개, 커널을 컴파일하여 메모리에 로드될 수 있는 이미지를 생성하는 절차, ARM 프로세서에 대해 이해하고 있어야 할 내용, 소스 분석의 효율을 높이기 위한 분석환경 구축에 대해 알아볼 것이다. 즉, 이번 파트는 분석을 위해 반드시 알아야 할 기본적인 내용들로 구성되어 있다.

본격적인 소스 코드 분석에 들어가기 전에 이번 파트를 통해 긴 호흡을 가다듬자.

pidhash_init()
bootmem_init()
kernel_thread()
rest_init()
cpu_idle()
ftrace_init()
kernel_init()
mem_init()

CHAPTER 1

# 커널에 대한 소개 그리고 2.6과 3.2의 차이

이 책은 리눅스 커널이 부팅되는 과정에서 호출되는 소스를 분석하여 리눅스가 어떻게 동작하는지, 리눅스가 동작하려면 어떤 부분이 초기화되어야 하는지를 살펴보는 분석서다.

이번 장에서는 분석에 앞서, 첫 번째로 리눅스의 개발이 리누스에 의해 시작되었고 현재는 가장 성공한 운영체제 중 하나라는 것에 대해 알아볼 것이다. 두 번째로 리눅스 커널이 다양한 서브시스템으로 구성되어 있는 모노리딕 커널이라는 것을 살펴볼 것이다.

마지막으로 이 책에서 분석한 커널 버전과 커널 3.2 버전 간의 차이를 알아볼 것이다. 구조상으로 커널이 2.4에서 2.6으로 버전업될 때만큼 큰 변화가 없다는 것을 확인하게 될 것이다. 따라서 이 책이 커널 3.2에 대해 중점을 두고 있진 않지만 최신 커널을 분석하는데 도움이 될 것이다.

## 1.1 커널의 탄생과 역할 그리고 내부 구조

### 1.1.1 리누스에 의해 탄생한 리눅스

리눅스[1]는 헬싱키 대학의 대학원생인 리누스 토발즈에 의해 1991년도에 공개된 운영체제다. 리누스는 헬싱키 대학 재학 시절에 미닉스(MINIX)의 라이선싱 정책에 불만을 느껴 리눅스를 만들게 되었다. 초기에는 미닉스 기반에서 시작하였지만 개발이 어느 정도 진척된 후에는 리눅스 커널이 동작하고 있는 리눅스 시스템 위에서 리눅스 커널이 개발되었다.[2]

그 후로 미닉스와 관련된 컴포넌트들이 모두 GNU 응용프로그램으로 교체되었고,[3] GNU 진영에서의 많은 지원과 수많은 커널 해커들의 기여로 전 세계에서 가장 많이 사용되고 가장 성공적인 오픈 운영체제가 되었다.

---

1) 리눅스의 초기 이름은 달랐다. 리누스는 자신이 만들던 커널의 이름을 프릭스(Freax)로 명명하였으나 자신의 커널을 올리던 사이트의 운영자의 제안으로 이 이름을 채택하게 되었고 지금까지 사용하고 있다. 리눅스(Linux)의 의미는 "리누스의 유닉스(Linus's Unix)", "리눅스는 유닉스가 아니다(Linux is not Unix)", 또는 "리누스의 미닉스(Linus's MINIX)" 등으로 통용된다.

2) 리눅스를 빌드한 컴파일러는 linux-gcc이다. 그렇다면 linux-gcc는 어떤 컴파일러로 빌드된 것일까? 이에 대해 어느 커널 해커와 리누스가 나눈 대화가 있다. 아래 URL에서 확인해보자.
http://groups.google.com/group/comp.os.linux/msg/4ae6db18d3f49b0e

3) GNU에게 어떤 영감이나 감동을 받았던 것일까? 이와 비슷한 시기인 1992년초에 리누스는 리눅스의 라이선스를 GPL로 교체하였다.

### 1.1.2 다양한 서브시스템이 모여 동작하는 모노리딕 커널

리눅스 커널은 여러 개의 서브시스템으로 이루어져 있다. 이것을 나타낸 그림 1-1을 보며 각 서브시스템의 역할에 대해 알아보도록 하자.


:: 그림 1-1 **리눅스 서브시스템 구조**

#### 아키텍처 의존적인 코드

리눅스는 아키텍처 종류에 상관없이 동일한 서비스를 제공하지만, 그 밑에서는 아키텍처마다 별도로 제어가 필요한 부분이 존재한다. CPU, MMU, 그리고 온보드(on board) 상태인 디바이스를 위한 로우레벨(low-level) 드라이버를 말하며, arch 디렉터리에 이 코드들이 존재한다.

#### 디바이스 드라이버

우리가 사용하는 컴퓨터에는 많은 주변기기(LCD, 버튼, 터치, 블루투스, 와이파이 등)들이 다양한 방식(I2C, SCSI, EIDE)으로 통신한다. 디바이스 드라이버는 이런 하이레벨(high-level)의 디바이스들을 위한 코드를 말하며, drivers 디렉터리에 존재한다. 리눅스 커널의 절반 이상이 디바이스 드라이버 코드이며, 최신 디바이스는 보통 리눅스에 가장 먼저 포팅된다.

### 프로세스 관리

프로세스가 실행되기 위해서는 CPU 자원을 분배받아야 한다. 동시에 여러 개의 프로세스가 동작 중일 때에는 정해진 정책에 의해 공평하게 자원을 분배받아야 한다. 또한 프로세스가 생성되고 상태가 전이되는 과정을 관리해주어야 한다. 이런 작업을 수행하는 서브시스템을 말하며, kernel 디렉터리에 존재한다.

### 메모리 관리

CPU만큼 중요한 시스템 자원은 메모리다. 메모리 관리 서브시스템은 메모리의 할당, 해제, 공유에 대한 관리를 담당하며, mm 디렉터리에 관련 코드가 존재한다.

### 가상 파일시스템

가상 파일시스템은 리눅스에서 흥미로운 부분 중 하나다. 다양한 파일시스템을 추상화하여 공통된 인터페이스를 제공한다. 이 덕분에 우리는 리눅스에서 동일한 파일 연산을 사용해서 많은 파일시스템이나 디바이스에 접근할 수 있게 된다.

### 네트워크 서브시스템

네트워크 서브시스템은 TCP/IP 통신 프로토콜과 소켓 인터페이스처럼 네트워크 장치를 추상화한 개념을 제공해주기 때문에 다양한 네트워크 장치를 일관된 방법으로 사용할 수 있다.

### 시스템 콜 인터페이스

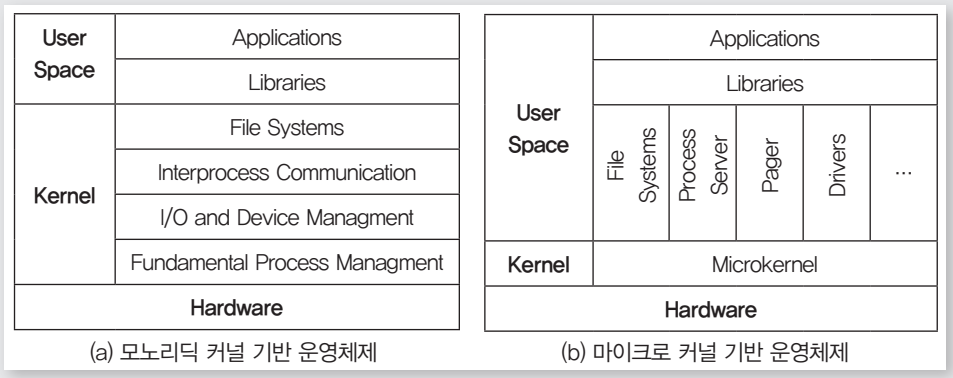
커널 내의 구현된 특정 기능을 유저 공간 레벨에서 호출할 수 있도록 인터페이스를 제공한다. 아키텍처별로 구현이 다를 수도 있으며, linux/kernel이나 linux/arch에 관련 코드가 위치해 있다.

**알아봅시다! 커널 구조의 구분 – 모노리딕 커널과 마이크로 커널**

대표적인 커널 구조인 모노리딕 커널과 마이크로 커널은 아래와 같은 특성을 가진다.

- **모노리딕 커널** – 커널 주소 공간에 모든 커널 서비스가 존재하고 동작한다. 커널 서비스를 직접 호출한다.
- **마이크로 커널** – 커널 서비스 중 일부가 유저 공간에 있다. 커널 서비스를 메시지 전달방식을 이용해서 호출한다.

커널 구조는 모든 커널 서비스가 커널 주소 공간에 모여서 동작하는지 여부에 따라 구분된다. 2.6 리눅스 커널이 모듈을 지원하는 모듈러 커널의 특성을 가지고 있어서 오해하는 경우도 있는데, 모듈도 역시 커널 주소 공간에서 동작하므로 리눅스 커널은 모노리딕 커널이라고 할 수 있다(그림 1–2 참조).

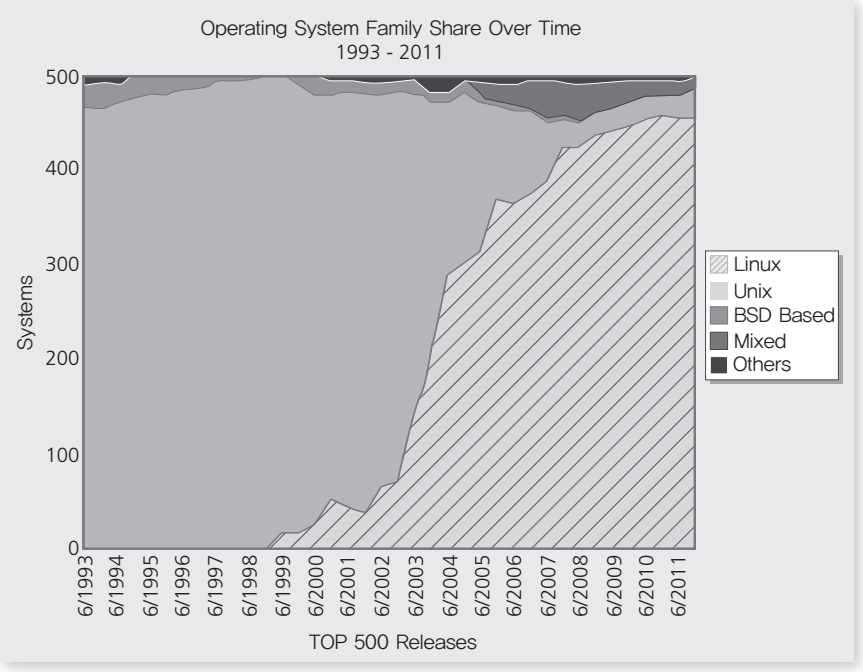



:: 그림 1-2 **모노리딕 커널 vs. 마이크로 커널**

※ **참고문헌:** Monolithic kernel vs. Microkernel(http://www.vmars.tuwien.ac.at/courses/akti12/journal/04ss/article_04ss_Roch.pdf)

## 1.1.3 전 세계에서 가장 유명한 범용 운영체제

리누스는 예상하지 못했겠지만 현재의 리눅스는 그 어떤 운영체제보다 더 많은 곳에 쓰이고 있다. 신뢰성과 성능이 중요한 수퍼컴퓨터에서도 리눅스의 진가가 드러났다(그림 1-3 참조). 또한 네트워크 장비, 텔레비전, 게임기까지 쓰임이 넓어졌고, 현재는 휴대전화 시장을 양분하고 있는 안드로이드 덕분에 그 유용성이 다시금 입증되고 있다.




:: 그림 1-3 **TOP 500 수퍼컴퓨터의 OS별 점유율**

## 1.2 커널 2.6과 3.2의 차이

2011년 7월, 리눅스 커널 3.0이 릴리즈되었다. 2003년에 처음으로 2.6 버전의 커널이 나왔으니 버전 2에서 3으로 넘어가는 데 8년이 걸린 것이다. 하지만 앞 번호가 바뀐 것이 어떤 큰 의미를 가진다기보다는 버전 넘버링 체계를 간결하게 하기 위해서, 그리고 리눅스 커널 20주년을 기념하는 것이라고 생각하는 게 맞을 것이다.

실제로 커널 아키텍처가 크게 변경된 부분은 없다. 하지만 매 릴리즈 때처럼 많은 부분에서의 수정은 늘 있다. 이 부분 역시 매 릴리즈 때마다 있던 수준이므로 큰 의미를 두지는 않아도 된다. 다음 표는 이 책에서 타깃으로 하고 있는 2.6.30 버전부터 현재까지 릴리즈된 3.2까지의 주요 변경사항을 나열했다.

::표 1-1 **리눅스 커널 버전별 주요 변경사항**

| 버전 | 변경사항 |
|---|---|
| 2.6.31 | • USB 3 지원 추가<br>• CUSE(Character device in Userspace) 추가<br>• Performance Counter 서브시스템 추가<br>• Kmemcheck, Kmemleak 기능 추가 |
| 2.6.32 | • Per-backing-device based writeback 추가<br>• Btr 파일시스템 개선<br>• Kernel Samepage Merging 기능 추가<br>• Run-time 전원 관리 기능 추가 |
| 2.6.33 | • Android 지원 제거<br>• Compcache 추가(스왑 장치 같은 램 기반의 블록 디바이스)<br>• DRBD 추가<br>• 시스템 콜 recvmmsg() 추가 |
| 2.6.34 | • Log 파일시스템 추가<br>• Ceph 파일시스템 추가<br>• RCU lockdep 기능 추가 |
| 2.6.35 | • Memory Compaction 구현 추가<br>• L2TP Version 3(RFC 3931), CAIF 지원 추가 |
| 2.6.36 | • Tile 프로세서 지원 추가<br>• Concurrency-managed workqueues<br>• OOM 재작성 |
| 2.6.37 | • Ext4의 확장성 개선, mkfs의 속도 개선<br>• XFS의 확장성 개선<br>• 성능상 핵심적인 커널 코드에서 BKL 모두 제거, 아직 driver, ioctl 등에서는 제거 안 됨 |

::표 1-1 (계속)

| 버전 | 변경사항 |
|---|---|
| 2.6.38 | • 세션 ID를 사용해서 자동으로 프로세스를 그룹짓는 기능 추가<br>• VFS의 확장성 개선(덴트리 캐시 재작성)<br>• Transparent huge pages 기능 추가 |
| 2.6.39 | • 2.6.37의 Ext4 개선사항을 기본으로 설정<br>• BKL 코드를 모두 제거 |
| 3.0 | • Btr 파일시스템의 자동 단편화 방지, 스크러빙 추가 및 성능 개선<br>• 시스템 콜 sendmmsg() 추가<br>• XEN dom0 지원 추가<br>• 페이지 캐시 성능을 개선하는 클린 캐시 추가<br>• Berkeley Packet Filter JIT filtering 추가 |
| 3.1 | • OpenRISC 프로세서 지원 추가<br>• writeback 성능 개선<br>• 슬랩 할당자 성능 개선 |
| 3.2 | • ext4에서의 큰 블록 크기 지원<br>• Btr 파일시스템의 scrub 속도 증가<br>• CFS에서의 CPU 대역폭 제한 제거<br>• Hexagon 프로세서 지원 추가 |

표를 보면 알 수 있듯이 커널이 버전업되면서 새로운 디바이스 드라이버, 아키텍처 지원, 파일시스템이 추가되거나 개선 혹은 불필요한 부분은 삭제되는 정도의 변경사항이 대부분이다. 우리는 리눅스가 동작하는 데 필요한 가장 기본적인 사항에 대해 초점을 맞추어 분석하고 있으므로 이런 변경사항에 크게 영향을 받지 않을 것이라고 생각한다.

리눅스가 부팅 과정에서 수행하는 작업과 리눅스가 동작하기 위해서 필요한 최소한의 요구사항은 쉽게 변하지 않기 때문이다. 따라서 이 책의 내용을 기반으로 2.6 커널을 분석한 다음 3.x 커널에 대한 분석을 추천한다.