Embedded System을 위한 GNU/Linux 최적화 기술[1]

한국전자통신연구원
표준연구센터 차세대인터넷표준팀
김 용 운, 김 용 진
qkim@pec.etri.re.kr

요약

데스크탑 PC 시스템과 내장형 시스템의 하드웨어 특성과 요구사항이 다르기 때문에 데스크탑 기반의 운영체제로 쓰이는 GNU/Linux를 내장형 시스템의 운영체제로 사용하기 위해서는 여러 가지 요소들에 대한 최적화가 뒤따라야 한다. 최적화를 위한 다섯 가지 방법을 설명하고, 실례로서 ZDISK와 LRP에서 만든 결과를 분석해보고, LFS와 몇 가지 문서를 바탕으로 Embedded Linux 제작 방법의 한 예를 설명하기로 한다.

1 서론

내장형 시스템(Embedded System)은 운영체제와 원하는 목적과 기능의 응용 프로그램을 내장하여 PC 컴퓨터처럼 원하는 일을 할 수 있게 하며, 그런 점에서 PC 컴퓨터의 다른 형태라 할 수 있다. PC의 경우에는 ROM, RAM, CPU, 모니터, 키보드, 및 하드디스크가 주요한 구성 요소이고 운영체제와 응용

[1] 문서 관리: V. 1.2, 2001. 3. 14.

프로그램들이 하드 디스크에 저장되어 있다.

반면에 내장형 시스템은 PC 시스템과 비슷한 형태로 구성되지만 운영체제와 응용 프로그램, 및 기타 파일들이 ROM에 저장되고, 하드 디스크에 저장되어 있는 파일 시스템을 이용하는 것이 아니라 ROM에 들어 있는 파일 시스템을 RAM에 로딩하여 사용하며, 주로 키보드 입력 장치를 사용하지 않고 시리얼 콘솔로 연결하여 가끔 일어나는 필요한 작업을 하고, 아무런 출력 장치를 사용하지 않는 경우가 많다. 그러나 필요에 따라 키보드를 입력장치로 활용하기도 하고, LCD 패널을 출력장치로 활용하기도 한다.

따라서 내장형 시스템은 PC 컴퓨터와 매우 유사한 기능을 한다고 할 수 있으나 하드웨어의 특성과 서비스 요구사항 및 목적이 다르다. PC 컴퓨터는 항상 전원이 들어와 있고 범용 목적으로 쓰이지만 내장형 시스템은 특정한 목적 전용으로 쓰이고 제한적 동작 환경에서 운용되는 경우가 많다. 계측기, 로봇 등 산업현장 기기는 물론이고, 전자 시계, TV용 인터넷 셋탑 박스, PDA, 프린터, 디지털 카메라, 오디오, VCR 카메라 등이 예가 될 수 있고, 전자 제어 장치가 들어가는 모든 제품에 대해 원하는 서비스 기능을 제공하기 위해 내장형 시스템으로 구성될 수 있다. 따라서 내장형 시스템은 PC 컴퓨터에 비해 시스템의 기능 구조는 비슷하지만, 제한된 목적으로 쓰이기 때문에 하드웨어 구조는 훨씬 더 단순할 수 있고, 더 다양하고 폭 넓은 응용 범위를 가진다.

특정 목적 전용의 시스템으로 만들어지고 그에 맞는 동작 환경에서 운용되기 때문에 시스템 하드웨어의 특성이 달라질 수밖에 없으며, PDA, 휴대용 PC, 셀룰러폰 등과 같이 건전지로 동작하는 장치의 경우에 절전이 매우 중요한 기능 목표가 되고, 내장형 시스템에 탑재되는 CPU 프로세서가 특정한 목적으로 개발된 것일 수도 있고, RAM이나 ROM의 용량도 비교적 소형일 경우가 많고,[2] 키보드 입력과 LCD 패널 출력이 없는 형태로 구성될 수도 있다. 또한 사람의 안전과 직결되는 중요한 제어 시스템이 있을 수도 있으며, 이 경우에는 신뢰성과 안정성이 매우 중요한 고려사항이 되어야 한다.

그러므로 기존의 PC 운영체제를 내장형 시스템에 그대로 사용할 수가 없으며 설계된 하드웨어와 기능 요구사항에 따라 최적화된 운영체제가 사용되어야 하고, 이를 바탕으로 한 응용이 개발되어 탑재되어야 한다.

GNU/Linux가 장시간의 안정성과 신뢰성이 증명되면서 내장형 시스템의 운영체제로 최근 주목을 받고 있으며, 이를 이용한 제품들이 속속 발표되고 있는 상황이다. GNU/Linux는 세계적인 규모의 수많은 개발자가 공동 참여하여 개발이 이루어지고 있으므로 비교적 빠르게 새로운 기술들이 접목되어 쓰이고 있고, 10여년 동안 개발과 실제 운영이 지속적으로 같이 이루어져 발전해온 것으로서 운영체제의 안정성과 신뢰성이 매우 높은 것으로 보고되고 있고, GNU 저작권 규정에 따라 별도의 저작권료를 낼 필요가 없어 많은 내장형 시스템에서 운영체제로 GNU/Linux가 선택되고 있는 상황이다.

GNU/Linux는 x86 PC 컴퓨터를 기반으로 만들어진 후에 MIPS, Alpha, PowerPC, SPARC, ARM 등과 같은 다양한 하드웨어 플랫폼에 이식되어 있으나, 범용 목적의 데스크탑 기반 컴퓨터에 운용되도록 만들어져 있으므로 내장형 시스템의 운영체제로 사용하기 위해서는 내장형 시스템의 운용 목적에 맞는 최적화가 이루어져야 한다.

2절에서는 GNU/Linux의 일반적인 최적화 방법에 대해 알아보고 3절과 4절에서는 ZDISK와 LRP (Linux Router Project)에서 개발한 것을 바탕으로 최적화의 실제 예를 분석하기로 하고, 5절에서는 LFS (Linux From Scratch)와 몇 가지 문서를 바탕으로 내장형 리눅스의 제작 예를 살펴보고, 마지막으로 6절에서 결론을 맺도록 한다.

2 GNU/Linux 최적화 방법

GNU/Linux를 내장형 시스템용 운영체제로 최적화하는 데는 일반적으로 다섯 가지 방법을 들 수 있으나, 이들 외에도 가능한 방법들이 있을 수 있다.

이러한 방법들은 운영체제의 크기를 최소화하여 ROM과 RAM의 크기를 최소화할 수 있도록 하는 것이며 절전형이나 Real-time OS 등의 기능을 위해서는 별도의 최적화 또는 확장 작업이 이루어져야 한다.[3]

---

[2] RAM 크기가 4MB ~ 16MB이고 Flash ROM 또는 Disk의 크기가 2MB ~ 16MB인 경우에 내장형 시스템이고 16MB 이상인 경우에 일반 데스크탑 컴퓨터로 구분하는 경우도 있으나 절대적인 기준은 아니다. 그만큼 제한적인 하드웨어 사양에서 내장형 시스템이 쓰인다는 것으로 이해하면 된다.

[3] GNU/Linux 기반의 대표적인 Real-time OS로 RT-Linux가 있다. http://www.rtlinux.org/에서 자료를 찾아볼 수

여기서는 운영체제의 규모 및 크기를 최소화할 수 있는 최적화의 방법에 대해서만 다루기로 한다.[4]

2.1 시스템 유틸리티 최적화

2.1.1 필수 유틸리티 선택

많은 경우에 내장형 시스템의 전체 소프트웨어 가운데 10%가 응용 프로그램이고 나머지 90%가 시스템 커널과 시스템 제어용 유틸리티이다. GNU/Linux에서의 시스템 유틸리티는 150여개에 이르고 있으며 내장형 시스템에서 모두 필요한 것이 아니다. 따라서 내장형 시스템 제어에 필요한 유틸리티만 선택하고 불필요한 것들을 제거함으로써 최적화가 가능하다.

예를 들어, 파일 시스템의 디스크 용량을 알려주는 `df`, 디렉토리를 만들어주는 `mkdir` 등과 같은 것들은 내장형 시스템의 목적과 기능에 따라 필요없는 것이 될 수 있으므로 제거하여 최적화할 수 있다. 이러한 방법은 절대적인 것이 아니며 목적과 기능에 따라 결정되어야 한다.

2.1.2 BusyBox 활용

BusyBox는 Debian Rescue/Install 디스크용으로 개발된 것이지만 소형 또는 내장형 시스템에서 최적의 유틸리티 환경을 제공해줄 수 있고, 다중호출 (multi-call) 바이너리 형태의 소형 유닉스 유틸리티이다.[1]

하나의 유틸리티는 고유한 목적의 기능 수행을 위한 코드 부분과 화면 입출력과 같은 다른 유틸리티와의 공통 코드 부분도 같이 포함하고 있다. 따라서 다수의 유틸리티에 있는 공통 코드와 각 개별 유틸리티의 고유 기능 코드를 한꺼번에 묶어서 다중호출 형태의 실행 프로그램 한 개로 만들게 되면 유틸리티 전체 크기를 대폭 줄이는 최적화 효과가 일어나게 된다. BusyBox는 이러한 방식으로 개발된 것이다.

BusyBox의 일반적 사용 방법은 각 유틸리티 이름에 대해 BusyBox로 링크를 만들어두는 것이다. 예를 들어, `ls` 명령어에 대해,

```
$ ln –s ./busybox ls
```

이와 같이 해두면, `./ls` 명령어 실행에 대해 `busybox`가 `ls`로서 동작하여 해당하는 기능을 수행하게 된다. 또한,

```
$ ./busybox ls
```

이와 같은 명령어 실행도 같은 기능을 수행한다.

현재 BusyBox는 0.45 버전까지 개발되어 있으며, 포함되어 있는 유틸리티들은 다음과 같다.[5]

```
ar, basename, cat, chgrp, chmod, chown, chroot, chvt, clear, cp,
cut, date, dc, dd, deallocvt, df, dirname, dmesg, du, dutmp, echo,
false, fbset, fdflush, find, free, freeramdisk, fsck.minix, grep,
gunzip, gzip, halt, head, hostid, hostname, id, init, insmod, kill,
killall, length, ln, loadacm, loadfont, loadkmap, logger, logname,
ls, lsmod, makedevs, mkdir, mkfifo, mkfs.minix, mknod, mkswap,
mktemp, more, mount, mt, mv, nc, nslookup, ping, poweroff, printf,
ps, pwd, reboot, rm, rmdir, rmmod, sed, setkeycodes, sfdisk, sh,
sleep, sort, swapoff, swapon, sync, syslogd, tail, tar, tee,
telnet, test, touch, tr, true, tty, umount, uname, uniq, update,
uptime, usleep, uudecode, uuencode, wc, which, whoami, yes, zcat, [
```

위와 같은 유틸리티들이 내장형 시스템에 모두 필요한 것은 아니기 때문에 내장형 시스템의 목적과 기능에 따라 적절히 선택하여야 하며, `busybox`에 포함시키거나 제외시키는 것은 `busybox.def.h` 파일에서 선택할 수 있도록 되어 있고, "`//`"를 이용하여 코멘트 표시를 하면 제외시킬 수 있다.

있다.

[4] Size Optimization이라고 말할 수 있다.

[5] 2000년 6월 21일 기준으로 BusyBox 0.45가 발표되었다.

2.1.3 TinyLogin 활용

TinyLogin도 BusyBox와 비슷한 목적으로 만들어진 것이다.[2] TinyLogin은 시스템 로그인, 사용자 인증 및 비밀번호 변경 등을 작업을 위해 필요한 몇 가지 유틸리티를 하나로 합친 것이다. TinyLogin은 시스템 보안을 향상시키는 방안으로 shadow password 기능을 제공하고 있으며, BusyBox를 보완하는 유틸리티로 활용할 수 있으나 BusyBox 없이 독립적으로 활용할 수도 있다. 하지만 대부분 경우에 같이 사용하곤 한다.[6]

GNU/Linux 시스템에 있는 관련 실행 파일들의 크기와 TinyLogin의 실행 파일 크기를 비교해보면 다음과 같다.

```
$ du -ch `which adduser deluser delgroup login sulogin passwd getty`
60k     /usr/sbin/useradd
20k     /bin/login
16k     /sbin/sulogin
24k     /usr/bin/passwd
32k     /sbin/getty
152k    total


$ ls -sh ./tinylogin
36k     ./tinylogin
```

TinyLogin도 BusyBox와 마찬가지로 모듈 설계로 되어 있어 내장형 시스템의 목적과 기능에 따라 필요한 기능만을 선택적으로 포함시키거나 제외시켜 만들 수 있으며, `tinylogin.def.h` 파일에서 "`//`"를 이용하여 코멘트 표시를 하면 제외시킬 수 있다.

2.1.4 Ash 활용

Unix 환경의 쉘 (shell) 프로그램으로 `bash`가 여러 가지 향상된 기능들을 갖고 있어 많이 쓰이고 있는데 크기가 다소 크다. 반면에 `ash`는 매우 작은 크기로서 쉘이 가지는 기본적인 기능들을 다 가지고 있으며 POSIX 인터페이스 규격을 지원하고 있고, `/bin/sh`로 설치하여 사용할 수도 있다. 따라서 내장형 시스템의 shell 프로그램으로 `ash`를 채택하게 되면 최적화에 도움이 될 수 있다. 참고로 두 가지 쉘의 크기를 살펴보면 다음과 같다.[3]

```
$ ls -sh /bin/bash
376k /bin/bash


$ ls -sh /bin/ash
68k /bin/ash
```

2.2 부팅 절차의 단순화

GNU/Linux 시스템에 전원이 공급되어 커널이 로딩되면 커널이 시스템 요소들에 대한 초기화 작업을 진행하고 이어서 시스템 최초의 프로세스로 `init` 프로그램을 실행시키고, `init` 프로세스는 프로세스 식별자로 '1'을 갖는다. 이렇게 되면 시스템 부팅 절차가 완료되었다고 할 수 있으며, `init` 프로세스는 시스템 콘솔을 열고 루트 파일 시스템을 마운트하는 등, 시스템 운영에 대한 초기화의 일부 역할을 담당하고, `/etc/inittab`의 정보를 참조하여 시스템 운영에 필요한 새로운 프로세스들을 생성시킨다.

`init` 프로세스가 생성시킨 새 프로세스들은 또 다른 프로세스들을 만들기도 하는데, 예를 들면, `getty` 프로세스는 사용자가 로그인을 시도할 때 `login` 프로세스를 만들기도 한다. 시스템 내에 있는 모든 프로세스들은 `init` 커널 프로세스의 자손들이다. 예를 들어, GNU/Linux 시스템의 프로세스 관계도 일부를 보면 다음과 같다. `init`은 모든 프로세스의 부모 프로세스이며, 그 외 프로세스들에 대한 부모와 자손 관계가 나타나있다.

---

[6] 2000년 6월 21일 기준으로 TinyLogin 0.78이 발표되었다.

```
$ pstree
init-+-apmd
     |-crond
     |-hanterm---bash---vi
     |-inetd---in.telnetd---login---bash---bash---hanterm---bash---pstree
     |-login---bash---startx---xinit-+-X
     |                               `-gnome-session
     |-sh---hanterm-org---bash---su---bash
     |-syslogd
     `-xscreensaver---attraction
```

따라서 프로세스로 동작할 시스템 유틸리티를 최적화하는 것뿐만 아니라, inittab의 내용을 단순화 시켜 내장형 시스템의 목적과 기능에 맞는 것만 수행시키도록 하여야 한다.

GNU/Linux 시스템에서 가장 많이 사용하는 init 프로그램은 sysvinit 패키지 속에 있으며,[4] 내장형 시스템에서도 그대로 사용할 수 있고, Debian 배포판에서 시스템 부팅을 더욱 빠르고 효과적으로 수행하도록 하기 위해 만든 start-stop-daemon[7] 프로그램을 같이 사용하여 기능 향상을 도모할 수 있다.[8] 이제는 sysvinit 패키지 속에 start-stop-daemon 프로그램도 같이 들어 있다.

BusyBox에는 가장 기본적인 형태의 init 기능도 포함되어 있다. 예를 들어, 여러 단계의 run level을 지원하지 않으며, inittab 파일이 없어도 다음과 같은 기본 설정으로 동작할 수도 있다.

```
::sysinit:/etc/init.d/rcS
::askfirst:/bin/sh
```

2.3 라이브러리 최적화

라이브러리 최적화는 두 가지 방향에서 접근할 수 있다. 필요한 시스템 유틸리티의 수가 적다면 공유 라이브러리를 사용하는 것보다는 static으로 컴파일하여 아예 라이브러리를 사용하지 않도록 할 수 있다. 반면에 시스템 유틸리티의 수가 많다면 공유 라이브러리를 사용하는 것이 더 효과적으로 최적화될 수 있다.

공유 라이브러리를 사용하는 경우에 모든 라이브러리가 다 필요한 것은 아니므로 시스템 유틸리티가 필요로 하는 라이브러리만 포함시키도록 하고, 많은 기능을 갖고 있는 최신 라이브러리를 사용하는 것보다는 필요로 하는 기능이 지원되고 크기도 작다면 구형 버전의 라이브러리를 사용하는 것이 더 효율적일 것이다.

예를 들어, glibc-2.1.2 패키지에 의해 만들어진 libc-2.1.2.so 라이브러리의 크기는 다음과 같다.

```
$ ls -sh /lib/libc-2.1.2.so
3.9M /lib/libc-2.1.2.so
```

반면에 구형 버전은 훨씬 작은 크기이다.

```
$ ls -sh libc.so.6
340k libc.so.6
```

GNU/Linux 기반의 PC 시스템에서는 보통 최신 라이브러리가 설치되어 있으며 libc.so.6를 libc-x.x.x.so로 링크해두고 있다.

```
$ ls -al /lib/libc.so.6
lrwxrwxrwx  1 root  root   13 Apr 19 12:53 libc.so.6 -> libc-2.1.2.so
```

특히 BusyBox를 이용하는 경우에 컴파일된 busybox 유틸리티는 /lib/libc.so.6 라이브러리를

---

[7] start-stop-daemon은 시스템 레벨의 프로세스를 생성시키고 종료시키는 제어 기능을 제공해준다.

[8] GNU/Linux 시스템의 부팅 절차를 자세히 알게 되면 부팅 과정에서 적용할 수 있는 최적화 방안에 대한 실마리를 추가적으로 찾을 수 있을 것이다.

포함하기 때문에 /lib/libc-2.1.2.so를 사용하기에는 너무 큰 크기이므로 /lib/libc.so.6 구형 라이브러리를 사용해야 한다.

2.4 커널 최적화

Linux 커널을 만드는 과정에 많은 선택 사항들이 있으며 이에 의해 특정 기능 모듈이 커널에 포함되어 만들어질 수 있고 제외될 수도 있으며 필요한 때만 선택적으로 커널로 로딩되었다가 필요 없을 때 제거되는 방식으로 만들어질 수 있다.

따라서 커널 최적화의 방법으로는 아예 필요 없는 기능을 제외하거나 대체 가능한 최소 기능 모듈을 선택하거나 커널 모듈의 선택적 로딩 등으로 나눌 수 있다.

예를 들어, 네트워킹이 필요 없는 내장형 시스템이라면 TCP/IP 네트워킹 모듈을 제외할 수 있고, SCSI 인터페이스를 사용하지 않고 HDD를 사용하지 않는다면 이 기능 또한 제거할 수 있으며, 그 외의 많은 기능들에 대해서도 불필요한 것이라면 Linux 커널 컴파일 옵션에서 제외하도록 선택한다. 또한 GNU/Linux 시스템이 주로 사용하는 ext2 파일 시스템 대신에 minix 파일 시스템을 쓸 수가 있고, 콘솔 장치를 사용하지 않는 내장형 시스템이라면 Linux 콘솔 대신에 시리얼 콘솔을 사용하도록 할 수 있다. 앞으로 사용할 가능성이 있는 기능 모듈에 대해서는 처음부터 커널에 포함시키는 것보다는 모듈 로딩 방식으로 커널을 만들도록 하여 커널 내부에 포함시키지는 않지만 향후 필요에 따라 사용할 수 있도록 하는 것이 더 좋다.

2.5 컴파일링 최적화

대부분의 프로그램과 라이브러리들은 기본적으로 특정한 CPU에 맞게 만들어지고, 또한 디버깅 심볼을 가지고 레벨 2로 (GCC 옵션에서 -g와 -O2) 컴파일되도록 하고 있다. 만약 목표로 하는 장치에서만 운용하고 다른 장치에서 사용할 계획이 없다면 컴파일 옵션을 바꾸어 목표 시스템에 맞게끔 최적화할 수 있다. 즉, 좀 더 높은 최적화 레벨을 적용하고, 디버깅 심볼을 포함시키지 않도록 하고, 목표 시스템의 하드웨어 플랫폼에 맞도록 컴파일하는 것이다.[5]

2.5.1 컴파일러와 라이브러리 선택

소스 코드를 컴파일하여 만들어지는 실행 프로그램의 크기는 어떤 종류의 컴파일러를 사용하느냐에 따라 달라질 수 있고, 같은 컴파일러를 사용하더라도 어떤 버전의 C 라이브러리를 사용하느냐에 따라 달라지기도 한다.

따라서 실행 코드 최적화를 위한 방안으로서 목표로 하는 시스템의 하드웨어 플랫폼에 최적화시킨 상용 컴파일러를 구입하는 것도 고려하여야 한다. 그러나 많은 경우에 GCC 범용 컴파일러를 이용한다.

GCC 컴파일러를 이용한다면 라이브러리 버전의 차이에 따른 크기 변화를 비교해 보고 선택하면 될 것이다.

2.5.2 디버깅 심볼 제거

이미 만들어진 바이너리 실행 프로그램으로부터 디버깅 심볼을 제거하기 위해서는 아래와 같은 strip 프로그램을 이용하면 된다. 여러 개의 파일에서 디버깅 심볼을 제거하고자 할 때는 와일드 카드 (*) 표시를 이용할 수 있다.

```
$ strip –strip-debug filename
```

또는

```
$ strip –strip-debug /usr/bin/*
```

실행 프로그램을 만들기 위해 컴파일하는 경우라면 아예 디버깅 심볼을 포함시키지 않도록 컴파일하면 될 것이다. 따라서 GCC 컴파일 옵션에서 -g 옵션을 빼도록 한다.

쉘 프로그램인 bash 실행 프로그램에 대해 디버깅 심볼을 포함하는 경우와 그렇지 않은 경우에 대한 비교 사례를 보면 다음과 같다. 따라서 시스템 운영에 불필요한 디버깅 심볼들을 제거하여야 하며, /lib와 /usr/lib에 있는 glibc와 gcc 파일들을 디버깅 심볼을 포함하는 것과 하지 않는 것을 비교하여 보면 그 차이가 더욱 뚜렷해서 87MB와 16MB라는 차이를 보이고 있다. 따라서 반드시 디버깅 심볼들을

포함하지 않도록 컴파일해야 한다.

BASH	with debugging symbols	without debugging symbols
static	2.3MB	645KB
dynamic	1.2MB	478KB

2.5.3 컴파일 옵션 최적화

컴파일 최적화를 위해 사용하는 옵션은 `-O`인데, `-O0`, `-O1`, `-O2`, 및 `-O3` 네 가지로 나뉜다. `-O0` 옵션은 최적화를 하지 않는다는 뜻이고, 숫자가 높을수록 최적화 수준이 높아진다. 각각의 차이는 GCC 사용 설명서를 참조하면 된다. 또한 컴파일을 할 때 목표 시스템의 하드웨어 구조를 명시하여 최적화하는 것이 좋다.

이러한 최적화 기능을 수행하도록 하기 위해 기본 컴파일 옵션을 변경해야 하는데, 첫 번째 방법으로는 컴파일 대상 소스 코드 패키지 내에 있는 모든 `Makefile`에 들어 있는 `CFLAGS` (GCC 컴파일러 옵션) 및 `CXXFLAGS` (G++ 컴파일러 옵션) 변수를 적절히 수정하는 것이다. 그러나 모든 디렉토리를 찾아서 일일이 수정하는 것은 어려운 일이므로 `CFLAGS`와 `CXXFLAGS` 환경 변수를 만들어서 `Makefile`에 있는 변수에 우선한다는 것을 알려주는 것이 좋다. 이것은 `make`에서 `-e` 옵션을 사용하면 된다.

BASH 쉘 환경에서 `CFLAGS`와 `CXXFLAGS` 환경 변수를 만들어 주는 방법은 다음과 같다.

```
$ export CFLAGS="-O3 -mcpu=xxx -march=yyy"
```

또는

```
$ export CXXFLAGS="-O3 -mcpu=xxx -march=yyy"
```

여기에서 `xxx`와 `yyy`는 적절한 CPU 식별자를 의미하는데, 예를 들어, 인텔 80686 CPU의 경우라면 `i686`이란 식별자를 사용하면 된다. 즉, 다음의 예와 같다.

```
$ export CFLAGS="-O3 -mcpu=i686 -march=i686"
```

그러나 이러한 컴파일 옵션 대치 방식은 원래의 `CFLAGS`나 `CXXFLAGS` 내에 매크로나 다른 옵션을 포함하고 있을 때 문제를 일으킬 수 있다. 즉, 원래에 있던 컴파일 옵션이 제외될 가능성이 있는 것이다. 따라서 이러한 문제 가능성을 염두에 두고 유의하여 작업해야 한다. `make`와 `make -e` 두 가지 컴파일에 대한 결과를 비교해 본다면 어떤 문제가 발생하는지 알 수 있을 것이다.

3 ZDISK 사례 분석

ZDISK는 3.5" 디스켓 한 장을 1.722MB 고용량 MS-DOS 형식으로 만들어 최소형의 GNU/Linux 시스템을 구성할 수 있는 커널, 시스템 유틸리티, 및 기본 응용을 담고 있으며, 시스템에 문제가 생겼을 때 응급 복구용 부팅 디스켓으로 사용하기 위해 만든 것이다.[6]

따라서 ZDISK는 두 가지 관점에서 살펴볼 수 있는데, MS-DOS 형식의 1.722MB 디스켓을 GNU/Linux 시스템 부팅을 할 수 있도록 하는 것과, 1.722MB MS-DOS 디스켓 한 장에 최소형 GNU/Linux 시스템을 구성할 수 있도록 최적화하는 것이다.

3.1 GNU/Linux 시스템 부팅

모든 PC 시스템은 전원을 켠 후에 ROM BIOS의 코드를 실행하는 것으로부터 동작을 시작하고, 주어진 시스템 점검을 마친 후에 부트 드라이브의 헤드0, 실린더 0, 섹터 1의 자리에 있는 부트 섹터를 RAM으로 로딩(loading)한다. 그런 후에 프로그램 실행은 로딩된 부트 코드로 넘어가게 된다.[7]

대부분의 부트 드라이브 헤드0, 실린더 0, 섹터 1의 자리에는 다음 두 가지 경우 가운데 한 가지가 들어

있는데, LILO와 같은 부트 로더가 있거나, Linux와 같은 운영체제 커널의 시작점이 있다.[9]

만약 Linux 커널을 디스켓이나 하드 드라이브의 첫 번째 섹터로부터 로딩해서 부팅을 시키게 되면 BIOS 코드가 실행된 후에 곧바로 커널 부팅이 시작되게 된다. 그러나 이런 방식은 사용자가 운영체제를 자유로이 선택할 수 없는 제한을 만들게 되므로 부트 로더 (boot loader)를 이용하여 부팅 절차를 제어할 수 있도록 한다.

부트 로더는 같은 하드웨어 시스템에서 단지 다시 부팅만 함으로써 원하는 운영체제를 선택하여 다른 운영체제로 전환할 수 있도록 한다. GNU/Linux 시스템에서 가장 많이 쓰이는 부트 로더는 LILO이며, 커널이 있는 위치를 알고 있어 커널 로딩을 하고 실행시키도록 함으로써 정상적인 부팅을 시작하도록 한다. LILO 부트 로더는 부트 섹터에 저장되어 있고 `lilo.conf` 파일에 부트 로더에 대한 환경 설정을 하도록 하고 있다. `lilo.conf` 설정 파일을 수정하여 `lilo` 프로그램을 실행시키면 수정된 LILO 부트 로더가 부트 섹터에 다시 저장된다.

전원을 켜고 부팅이 시작되면 BIOS에 의해 LILO 부트 로더가 로딩되어 부트 로더 코드가 실행된다. LILO 부트 로더는 운영체제와 부팅에 관련된 옵션 파라미터를 선택할 수 있는 인터페이스를 제공하며,[10] 선택에 따라 커널을 로딩하거나 다른 운영체제의 부트 섹터를 로딩한다. 이어서 로딩된 커널이나 새로운 부트 코드가 실행되는데, Linux 커널이 로딩된 경우에 각종 하드웨어 장치들에 대한 점검 및 초기화 작업을 하고 루트 파일 시스템을 마운트한다.[11]

루트 파일 시스템이 마운트되고 나면, 커널은 파일 시스템 내에 있는 `init` 프로그램을 찾아 실행시킨다. 이로써 시스템 최초의 프로세스가 생성되고, `init` 프로세스는 `inittab` 설정 파일에 따라 지정된 작업을 수행한다.

이러한 작업 가운데 하나로서 `sysinit` 스크립트를 찾아서 실행시킨다. 이 스크립트는 시스템 기본 서비스를 제공할 수 있도록 하는 쉘 명령어들의 집합으로 이루어져 있는데, 모든 디스크에 대해 `fsck`를 실행시키고, 필요한 커널 모듈들을 로딩하고, 스와핑과 네트워크 초기화 작업을 수행시키고, `fstab`에 기록되어 있는 디스크 마운팅 작업을 수행시키고, 그 외의 다른 스크립트들도 동작시킨다.

`sysinit` 스크립트가 작업을 끝내고 나면 시스템 제어 권한을 다시 `init` 프로세스에게 돌려주고, `init` 프로세스는 `inittab` 파일의 `initdefault`에 표시되어 있는 default runlevel로 진입한다. 통상의 경우에 runlevel은 시스템 콘솔과 `tty` 사이의 통신을 담당하는 `getty`를 나타내기 위해 쓰이는데, `getty`는 `"login: "` 프롬프트를 표시하고, 사용자 인증 및 세션을 관리하는 `login` 프로그램을 실행시킨다. 이로써 부팅 절차는 완료되고 사용자 로그인을 위한 대기 상태로 있게 된다.

3.2 ZDISK의 부팅

ZDISK를 위해 사용하는 디스켓은 MS-DOS 형식을 사용하고 있다. 그러나 디스켓의 부트 섹터에는, MS-DOS/Windows FAT 파일 시스템에 저장되어 있는 GNU/Linux 운영체제를 로딩하기 위한 SYSLINUX [8] 부트 로더가 들어 있다. LILO와 비슷한 목적이기는 하지만 더욱 간단하게 Linux 커널 로딩을 할 수 있게 해주고, MS-DOS 명령어를 가지고 쉽게 디스켓에서 파일 관리를 할 수 있으므로, 응급처치용 또는 특정한 목적의 부트 디스크를 만드는 데에 쓰인다.

따라서 SYSLINUX는 일반적인 목적의 범용 부트 로더용으로는 적당하지 않고, FAT 파일 시스템에서의

---

[9] 부트 섹터는 512 바이트로 이루어져 있으며, 부트 섹터에 부트 로더 전체가 들어가 있을 수도 있고, 부트 로더의 크기가 커서 부트 섹터 밖의 데이터 영역까지 자리할 수 있다. 후자의 경우와 마찬가지로 Linux와 같은 운영체제 커널도 부트 섹터 1의 자리부터 커널이 시작되어 데이터 영역까지 자리잡을 수 있다. 참고로 Linux 커널의 앞부분 512바이트에는 부트 로더가 들어 있다. 따라서 섹터 1번지에 부트 로더를 설치하지 않고, Linux 커널을 두더라도 정상적인 부팅이 가능하다.

[10] 부트 로더가 커널을 로딩하면서 각종 선택 변수들을 전달해주는데, LILO의 경우에 `lilo.conf` 파일에 그 내용이 저장된다. 다양한 선택 변수들에 대한 설명이 http://www.linuxdoc.org/HOWTO/BootPrompt-HOWTO.html 문서에 있다.

[11] 디스켓으로부터 부팅되는 경우와 같은 상황에서 종종 루트 파일 시스템은 RAM 디스크 장치로 로딩되곤 한다. 이때 파일 시스템은 마치 하드 디스크에 있는 것처럼 RAM을 이용한다. 파일 시스템을 RAM 디스크로 로딩하는 데는 두 가지 이유가 있는데, 첫 번째는 플로피 디스크보다 RAM이 매우 빠르기 때문에 시스템 동작으로 빠르게 할 수 있으며, 두 번째는 커널이 플로피 디스크로부터 압축된 형태의 파일 시스템을 로딩하여 RAM 디스크로 풀어서 로딩할 수 있으므로 디스켓에 보다 많고 큰 크기의 파일들을 담아둘 수 있기 때문이다.

GNU/Linux 부팅용으로만 쓸 수 있다. Ext2 파일 시스템에서는 LILO를 사용하는 것이 좋고, MS-DOS 프롬프트에서 커널을 로딩하고자 하는 경우에는 LOADLIN이 좋다.

부트 섹터를 가진 MS-DOS 디스켓을 만들어서 SYSLINUX 부트 로더를 설치하면 `ldlinux.sys` 파일이 생성되고, Linux 커널 로딩 환경을 설정하는 파일로 `lilo.conf`와 비슷하게 `syslinux.cfg` 파일을 추가하면 된다. 설정하는 형식도 LILO와 비슷하다. 예를 들어 다음과 같으며, 상세한 설정 방법은 SYSLINUX 패키지 속에 들어 있는 사용 설명서를 참조하면 된다.

```
DEFAULT rescue

LABEL rescue
KERNEL kernel
APPEND vga=normal load_ramdisk=1 prompt_ramdisk=0 \
       ramdisk_size=2800 initrd=rescue.gz root=/dev/ram0 rw
DISPLAY message.txt
PROMPT 1
```

위와 같은 `syslinux.cfg`를 가지고 부팅을 시켜보면 다음과 같이 동작한다. 전원을 켜고 BIOS가 실행되면 플로피 드라이브로부터 부트 섹터와 `ldlinux.sys`를 읽어 들이고, 이때 로딩된 SYSLINUX 부트 로더는 `syslinux.cfg` 파일을 참조하여 작업을 수행한다. 위 변수들에 대해 간단히 설명하면 다음과 같다.[9][10][11]

- 기본적으로 설정되는 부팅 커널은 `rescue` 라벨에 등록되어 있는 것이다. (`DEFAULT rescue`)
- `rescue` 라벨이 선택된 경우에 디폴트로 로딩할 커널 이름은 `kernel`이다. (`KERNEL kernel`)
- 부팅할 때 선택할 VGA 텍스트 모드는 `normal` 모드로서 80x25이다. (`vga=normal`)
- 부팅되는 동안 `message.txt` 파일을 화면에 보여 준다. (`DISPLAY message.txt`)
- RAM 디스크를 사용하고 크기는 2800KB이다. (`load_ramdisk=1 ramdisk_size=2800`)
- RAM 디스크 이미지를 가진 플로피 디스켓을 삽입하라는 프롬프트를 보여주지 않는다. (`prompt_ramdisk=0`)
- RAM 디스크로 초기화 로딩할 압축 파일은 `rescue.gz`이고, 루트 파일 시스템의 위치는 RAM 디스크이다. (`initrd=root.lrp root=/dev/ram0`)
- 부팅되는 동안에 "`boot: `" 프롬프트를 표시하도록 하여 어떤 `LABEL`의 커널을 로딩할 것인지 입력할 수 있도록 한다. (`PROMPT 1`)

ZDISK 패키지가 들어 있는 MS-DOS 디스켓에는 다음과 같은 파일들이 들어 있다.

```
$ mount –t msdos –o rw /dev/fd0H1722 /mnt/fdd
$ cd /mnt/fdd
$ ls -al
total 1622
drwxrwxr-x   2 root     root         7168 Jan  1  1970 ./
drwxr-xr-x  27 root     root         4096 Jun 23 16:42 ../
-rwxrwxr-x   1 root     root       658523 Jun 24 14:57 kernel
-r-xr-xr-x   1 root     root         5860 Jun 24 14:43 ldlinux.sys
-rwxrwxr-x   1 root     root          858 Jun 24 14:58 message.txt
-rwxrwxr-x   1 root     root       982177 Jun 24 14:56 rescue.gz
-rwxrwxr-x   1 root     root          527 Jun 24 15:15 syslinux.cfg
```

3.3 ZDISK의 GNU/Linux 최적화

ZDISK 패키지에서 GNU/Linux를 최적화하는 방법은 이미 앞서 2절에서 설명한 것과 같다. 커널은 Linux 2.2.14를 컴파일하여 만든 것이며, 하드 디스크 기반의 기존 GNU/Linux 시스템에 긴급 상황의 응급 복구를 위해 사용하고자 하는 것이기 때문에 커널 최적화를 할 필요가 없으며, 기존 커널 컴파일 옵션을 그대로 사용하지만, RAM 디스크와 RAM 디스크 초기화 (`initrd`) 옵션과 Loopback 장치 지원 옵션은 선택하여야 한다. 그래야만 RAM 디스크 상에 루트 파일 시스템을 구성하여 부팅을 시킬 수 있기

때문이다.

따라서 ZDISK에서의 실제적인 최적화는 루트 파일 시스템을 구성할 각종 파일들의 크기를 축소시키는 것이다. 3.2절의 끝에 나열되어 있는 파일 목록에서 rescue.gz가 RAM 디스크 상에 설치될 파일 시스템이 들어 있는 파일이고, 이것의 크기를 축소시킬 모든 방법들을 동원하면 되는 것이다. 여기에 적용되어 있는 최적화의 방법들은 이미 앞서 설명한 2절의 모든 사항들이다.

이들 최적화의 한 가지 예로서 ZDISK는 BusyBox를 사용하여 시스템 유틸리티 최적화를 하고 있고, init 프로세스를 실행시키기 위해 /etc/inittab 파일 없이 BusyBox의 init 기능을 바탕으로 동작하게 하고 있다. BusyBox의 init 기능은 여러 단계의 runlevel을 지원하지 않도록 되어 있어, 사용자 로그인 환경도 지원하지 않는다. 이 때문에 TinyLogin 최적화를 이용할 필요는 없다.

rescue.gz는 다음과 같은 디렉토리와 관련 유틸리티 및 응용 파일들로 구성되어 있다.

```
$ cd rescue.tmp
$ dir -l
total 52
drwxr-xr-x   2 root     root         4096 Jun 18 10:30 bin
drwxr-xr-x   2 root     root         4096 Jun 11 08:06 boot
drwxr-xr-x   2 root     root         8192 Apr 15 05:35 dev
drwxr-xr-x   2 root     root         4096 Jun 10 07:09 etc
drwxr-xr-x   3 root     root         4096 Jun 19 23:46 lib
drwxr-xr-x   9 root     root         4096 Dec  7  1999 mnt
drwxr-xr-x   2 root     root         4096 Aug 15  1998 proc
drwxr-xr-x   3 root     root         4096 Dec 15  1999 root
drwxr-xr-x   2 root     root         4096 Jun 18 09:01 sbin
drwxr-xr-x   2 root     root         4096 Jan  9  1999 tmp
drwxr-xr-x   7 root     root         4096 Jun 18 23:35 usr
drwxr-xr-x   5 root     root         4096 Feb  8 18:41 var
```

ZDISK에서는 rescue.gz 파일을 만들기 위해 genext2fs라는 실행 프로그램을 이용하고 있다. 위의 예와 같은 디렉토리를 만들고 최적화 기술을 적용하여 필요한 파일들을 각 디렉토리에 넣고 나면 부팅될 때에 초기화 로딩 (initrd) 옵션에서 로딩될 수 있도록 하기 위해 압축하여야 한다.

이렇게 만들어지는 파일 시스템은 루트 관리자와 일반 사용자를 구분하여 서로 다른 권한으로 시스템을 사용하도록 구성되기 때문에 구분할 필요가 없는 내장형 시스템과 같은 특수한 환경에서는 불편하게 할 뿐이다.

그래서 genext2fs 프로그램을 이용하여 일반 사용자 권한의 ext2 파일 시스템을 만들어 일반 사용자도 장치 노드를 만들 수 있는 등의 일이 가능하도록 한다. rescue.gz 파일은 rescue.tmp 디렉토리 속에 들어 있는 모든 디렉토리와 파일들을 다음과 같은 genext2fs 명령으로 일반 사용자 권한의 ext2 파일 시스템 설치용 파일로 만들어서 압축한 것이다.

```
$ genext2fs –r 0 –i 600 –b 2800 –d rescue.tmp rescue
$ gzip –9 rescue
$ ls –al rescue.gz
-rw-rw-r--   1 root     root       982177 Jun 24 14:54 rescue.gz
```

genext2fs에서 –r 옵션은 예약할 블록의 수를 나타내고, -i 옵션은 inode의 수를 나나태고, -b 옵션은 블록의 크기를 나타내고, -d 옵션은 대상 디렉토리를 나타내고, 마지막의 것은 최종 결과를 만들어낼 파일 이름이다. 이렇게 만들어지는 최종 rescue.gz 파일을 가지고 초기화 로딩을 하도록 하고 있다.

이렇게 하여 RAM 디스크로 로딩되는 루트 파일 시스템은 ext2 형식으로 마운트된다.

ZDISK의 /etc/mtab의 내용에는 다음과 같이 들어 있다.

```
/dev/ram0 / ext2 rw 0 0
```

/etc/fstab의 내용을 살펴보면 다음과 같다.

```
proc  /proc  proc  defaults  0  0
```

4 LRP (Linux Router Project) 사례 분석

LRP는 1.44MB 디스켓 한 장에 최소형의 GNU/Linux 시스템을 구성할 수 있는 커널, 시스템 유틸리티, 및 응용을 담고 있으며, 사실상 무용지물로 구석에 버려져 있는 인텔 80386, 80486과 같은 구형 시스템을 가지고 소형 라우터로 만들 수 있도록 하기 위해 만든 것이다.[12]

따라서 LRP 사례도 ZDISK의 경우와 마찬가지로 두 가지 관점에서 볼 수 있는데, 한 가지는 MS-DOS 형식의 디스켓으로부터 GNU/Linux 시스템 부팅을 할 수 있도록 하는 것이고, 다른 한 가지는 1.44MB 또는 1.722MB MS-DOS 디스켓 한 장에 최소형 GNU/Linux 시스템을 구성할 수 있도록 최적화하는 것이다.

4.1 LRP의 부팅

LRP를 위해 사용하는 디스켓의 파일 시스템도 ZDISK의 경우와 같이 MS-DOS 형식을 사용하고 있으며, MS-DOS 플로피 디스켓으로부터 GNU/Linux 운영체제를 로딩하기 위해 SYSLINUX 부트 로더를 사용한다.

LRP 디스켓에는 다음과 같은 파일들이 들어 있으며, `lrp` 확장자를 가진 파일들은 `tar`로 묶여져 `gzip`으로 압축되어 있는 형식으로 이루어져 있다. 즉, `*.lrp`는 `*.tar.gz`와 완전히 똑같다.

```
$ mount –t msdos –o rw /dev/fd0 /mnt/fdd
$ cd /mnt/fdd
$ ls -al
total 1222
drwxrwxr-x   2 root     root         7168 Jan  1  1970 ./
drwxr-xr-x  27 root     root         4096 Jun 23 16:42 ../
-rwxrwxr-x   1 root     root        32889 May 29  1999 etc.lrp
-r-xr-xr-x   1 root     root         5476 May 29  1999 ldlinux.sys
-rwxrwxr-x   1 root     root       362995 May 29  1999 linux
-rwxrwxr-x   1 root     root          488 May 29  1999 local.lrp
-rwxrwxr-x   1 root     root          628 May 29  1999 log.lrp
-rwxrwxr-x   1 root     root        52302 Jun 23 16:46 modules.lrp
-rwxrwxr-x   1 root     root       782604 May 29  1999 root.lrp
-rwxrwxr-x   1 root     root          179 Jun 23 16:52 syslinux.cfg
-rwxrwxr-x   1 root     root          515 Jun 23 16:51 syslinux.dpy
```

LRP에 들어 있는 `syslinux.cfg`의 내용을 보면 다음과 같다.

```
DISPLAY syslinux.dpy
TIMEOUT 0
DEFAULT linux
APPEND=load_ramdisk=1 initrd=root.lrp initrd_archive=minix \
       ramdisk_size=4096 root=/dev/ram0 boot=/dev/fd0,msdos \
       LRP=etc,log,local,modules
```

위와 같은 `syslinux.cfg`를 가지고 부팅을 시켜보면 다음과 같이 동작한다. 전원을 켜고 BIOS가 실행되면 플로피 드라이브로부터 부트 섹터와 `ldlinux.sys`를 읽어 들여 SYSLINUX 부트 로더를 로딩하고, 이때 부트 로더는 `syslinux.cfg` 파일을 참조하여 작업을 수행한다. 다음과 같은 사항들이 정의되어 있다.[9][10][11]

- 부팅되는 동안 `syslinux.dpy` 파일을 화면에 보여 준다. (`DISPLAY syslinux.dpy`)
- 로딩 대상 커널을 선택하기 위해 제공하는 "`boot: `" 프롬프트에서 사용자 입력 대기 시간을 주지 않도록 한다. 즉, 사용자 입력 없이 다음 단계로 넘어간다는 뜻이다. (`TIMEOUT 0`)
- 디폴트로 로딩할 커널의 이름은 `linux`이다. 3.2절의 ZDISK 경우에서는 디폴트로 실행할 것이 `rescue` 라벨을 가진 부분이며, `rescue` 라벨에 있는 `KERNEL` 변수에 적혀 있는 파일 이름을 커널로서 로딩한다는 뜻이다. 그러나 LRP의 경우에서는 `DEFALUT` 변수에서 로딩할 커널의 파일 이름을 표시하도록 하고 있다. 따라서 `DEFAULT` 변수는 `LABEL`이 있느냐 없느냐에 따라 약간 다른 의미를 가진다. (`DEFAULT linux`)

- RAM 디스크를 사용하고 크기는 4096KB이다. (`load_ramdisk=1 ramdisk_size=4096`)
- 루트 파일 시스템은 RAM 디스크를 사용하고, 부팅할 때 RAM 디스크로 초기화 로딩할 압축 파일은 `root.lrp`이고, RAM 디스크에 있는 루트 파일 시스템이 사용할 파일 시스템의 종류는 `minix`이다. (`root=/dev/ram0 initrd=root.lrp initrd_archive=minix`)
- LRP 시스템을 구성하기 위해 루트 파일 시스템으로 로딩할 파일들은 `etc.lrp`, `log.lrp`, `local.lrp`, `modules.lrp` 이다.
- 시스템 부팅 위치는 플로피 디스크 드라이브이며, MS-DOS 파일 시스템으로 이루어져 있다. 이 위치를 바꾸면 하드 디스크로부터 부팅을 시작하도록 할 수 있다. 즉, 플로피 디스켓으로부터 SYSLINUX 부트 로더를 읽어 들인 후에 `boot` 변수에 지정된 장소에서 커널과 파일 시스템, 및 그 외 필요한 파일들을 로딩하도록 하는 것이다. (`boot=/dev/fd0,msdos`)

`syslinux.cfg` 파일에서 GNU/Linux 시스템이 사용할 파일 시스템으로 `minix`를 선택하였기 때문에 `/etc/fstab`의 내용에도 이것을 표시하는 내용이 아래의 예와 같이 들어가야 한다.

```
# /etc/fstab: static file system information.
#
# <file system>    <mount point>   <type>  <options>   <dump>  <pass>
/dev/ram0           /               minix   rw
proc                /proc           proc    noauto      0       0
```

4.2 LRP의 GNU/Linux 최적화

LRP는 GNU/Linux 시스템에 대한 응급 복구용이 아니라 일반 PC를 라우터 박스로 만드는 용도이기 때문에 ZDISK와는 약간 다른 최적화 방법을 쓰고 있다.

LRP에서는 GNU/Linux 라우터 시스템을 운용하고자 하기 때문에 많은 시스템 유틸리티가 필요 없어 `/bin`과 `/sbin` 디렉토리에 작은 수의 시스템 유틸리티를 포함하고 있으며, 부가적으로 네트워크 관리용 유틸리티들을 갖고 있다. 따라서 BusyBox를 이용할 필요가 없는 것으로 보인다.

시스템 로그인을 위해 TinyLogin을 이용한 최적화는 하지 않고 있으나 다음과 같은 최소형의 로그인 환경을 만들어 사용하고 있다.

```
$ ls -al ./bin/login ./usr/bin/passwd ./sbin/getty
-rwxr-xr-x 1 root root 22968 Nov  3  1998 ./bin/login
-rwxr-xr-x 1 root root 12156 Mar 10  1999 ./sbin/getty
lrwxrwxrwx 1 root root 15 Jun 26 13:19 ./usr/bin/passwd -> ../../bin/login
```

라이브러리 최적화를 위해 시스템에 필요 없는 많은 라이브러리를 제외하고 있으나 LRP 라우터 시스템 유틸리티에 필요한 라이브러리를 추가적으로 사용하고 있어 ZDISK보다 많은 종류의 것을 사용하고 있다.

LRP의 커널은 Linux 2.0.36pre15를 사용하여 많은 기능들을 모듈로 추가하여 쓸 수 있도록 만들어진 것으로서 커널 내에 내장하는 것보다는 훨씬 크기가 작다. 이에 따라 모듈에 대한 파일이 있는지 여부에 따라 해당 기능에 대한 지원 여부가 결정된다.

```
$ ls -sh linux
360k linux
```

GNU/Linux 시스템의 최적화는 ZDISK와 LRP 사례에서와 같이 목적과 기능에 따라 다양한 방법을 통해 이루어진다.

5 Embedded Linux 시스템 제작 방법

사용자 목적에 따라 선택적으로 GNU/Linux 시스템을 구축할 수 있는 방법을 알기 위해서는 LFS (Linux From Scratch) [13], 'Embedded Linux Howto' [14], 'The Linux Bootdisk HOWTO' [7] 문서를 참조하는 것이 좋다. 만들어야 할 파일 시스템 구조와 만드는 방법에 대한 설명들이 있으므로 앞서

설명한 최적화의 방법들을 적용한다면 내장형 시스템에 적용할 수 있는 GNU/Linux 시스템 패키지를 완성할 수 있다.

5절에서 만들고자 하는 GNU/Linux 시스템은 아무런 운영체제도 설치되어 있지 않는 시스템에서 만들 수는 없는 것이며, 기존의 GNU/Linux 시스템 내의 별도 장소에다 간단한 GNU/Linux 시스템을 만드는 것을 의미한다.

시스템 제작 순서로는 다음과 같이 할 수 있다.

- 루트 파일 시스템 설치
- 기본 시스템 설치
- 라이브러리 설치
- 필요한 응용 설치

5.1 루트 파일 시스템 설치

루트 파일 시스템은 GNU/Linux 시스템을 설치할 새로운 파티션을 만들고, GNU/Linux 시스템에서 주로 사용하는 ext2 파일 시스템을 설치하고, 필요한 디렉토리들을 만드는 작업이다.[12]

5.1.1 새로운 파티션 만들기

기존의 GNU/Linux 시스템에 새로운 하드 디스크를 추가하여 파티션을 만들 수도 있고, 기존 시스템의 여유 하드 디스크 용량에서 파티션을 나눌 수도 있다.

`fdisk` 프로그램을 이용하여 새로운 파티션을 만들고, 이에 따라 변경된 파티션 테이블을 저장하고, 프로그램 종료를 한다. 이때 파티션 테이블 변경을 위해 시스템을 다시 부팅하라고 할 수도 있으니, 만약 그렇다면 다시 부팅시키도록 한다.

이렇게 만들어진 새로운 파티션이 `/dev/hda5`라고 가정하도록 한다.

5.1.2 ext2 파일 시스템 만들기

위에서 만들어진 새로운 파티션에 ext2 파일 시스템을 설치하기 위해 `mke2fs` 명령어를 사용한다.

```
$ mke2fs /dev/hda5
```

5.1.3 파티션 마운트

새롭게 만들어진 파티션에 각종 디렉토리 및 파일들을 저장하기 위해서는 마운트가 되어야 한다. 이 파티션에 대한 마운트 위치를 나타내기 위해 디렉토리를 하나 만들도록 하고, 여기서는 `/lfs` 라고 가정한다.

```
$ mkdir /lfs
```

대상 파티션을 위 디렉토리 이름과 마운트시킨다.

```
$ mount /dev/hda5 /lfs
```

5.1.4 파일 시스템 구성

루트 파일 시스템에는 완전한 GNU/Linux 시스템을 구성하는 데 필요한 모든 것들이 담겨 있어야 한다. 이를 위해 필요한 디렉토리를 구성하도록 하며 FHS [15] 표준에 따르도록 한다.

```
$ cd /lfs
$ mkdir bin dev home proc sbin usr boot etc lib mnt root tmp var
```

---

[12] Tom Fawcett가 작성한 'The Linux Bootdisk HOWTO' [7] 문서에는 루트 파일 시스템을 만드는 데 있어 고려해야 하는 사항들에 대해 보다 자세하게 설명되어 있다.

```
$ mkdir -p usr/bin usr/sbin usr/share usr/lib
$ mkdir -p etc/config etc/default etc/init.d etc/rc.boot
$ mkdir -p etc/rc0.d etc/rc1.d etc/rc2.d etc/rc3.d  \
etc/rc4.d etc/rc5.d etc/rc6.d etc/rcS.d
```

주요 디렉토리들의 역할은 다음과 같다.

- `/proc`: 프로세스 파일 시스템에 대한 내용들이 들어가며 커널이 관리한다.
- `/etc`: 시스템 환경 설정에 관한 파일들이 들어 있다.
- `/sbin`: 시스템 관리에 관계된 주요 실행 프로그램들이 들어 있다.
- `/bin`: 시스템에 관한 기본적인 실행 프로그램들이 들어 있다.
- `/lib`: 실행 프로그램들이 사용할 런 타임 공유 라이브러리가 들어 있다.
- `/mnt`: 관리용 마운트 위치들에 대한 내용이 들어간다.
- `/usr`: 추가적인 유틸리티나 응용들이 들어 있다.
- `/dev`: 시스템 입출력에 필요한 파일들이 들어 있다.
- `/boot`: 시스템 부팅과 운영을 위해 사용할 수 있는 시스템 커널이 있는 장소를 의미한다.

여기서 `/dev` 디렉토리에 있는 장치 파일들은 `mknod` 명령어를 이용하여 만들 수도 있으나, 시간 절약을 위해 기존 GNU/Linux 시스템에 있는 것들을 그대로 복사하여 사용하도록 하고, 목적으로 하는 기능에 필요한 것들만 복사하도록 한다. 예를 들어, 플로피 드라이브가 없는 내장형 시스템이라면 `/dev/fd0`나 `/dev/fd1` 같은 장치 파일들은 복사할 필요가 없을 것이다. 아래의 예는 모든 장치 파일들을 복사하는 예를 나타내고 있다.

```
$ cp -av /dev/* /lfs/dev/
```

이상의 과정을 통해 루트 파일 시스템이 구성되었으며 GNU/Linux 시스템에 필요한 각종 유틸리티들과 파일들을 만들어 넣어야 한다.

5.2 기본 시스템 설치

여기서는 시스템 부팅 환경과 로그인 환경을 구성하도록 함으로써 가장 기본적인 형태의 GNU/Linux 시스템을 만들도록 한다.

Linux 커널이 로딩되어 실행에 들어가면 각종 시스템 요소들에 대한 초기화 작업을 수행하고, `init` 프로그램을 로딩함으로써 시스템 부팅 절차를 완료한다. GNU/Linux 시스템에서는 sysvinit 패키지에 들어 있는 `init` 프로그램을 주로 사용한다. 앞서 3.3절의 ZDISK 경우에서 언급한 것처럼 최적화를 위해 sysvinit의 `init` 프로그램을 이용하지 않고 `busybox`를 대신 이용할 수도 있을 것이다.

각종 유틸리티 프로그램들의 소스 코드 컴파일은 LFS 문서를 참조하도록 한다.

5.2.1 Sysvinit 및 start-stop-daemon 설치

먼저 대상 하드웨어 구조에 맞게 sysvinit과 start-stop-daemon 소스 코드를 컴파일하여, `init` 실행 프로그램은 `/lfs/sbin` 디렉토리에 넣고, `start-stop-daemon` 실행 프로그램은 `/lfs/usr/sbin` 디렉토리에 넣는다.

5.2.2 Sysvinit 환경 설정

Sysvinit에 대한 환경 설정 파일은 `inittab`이다. 이것은 시스템 목적에 따라 적절히 구성되어야 하는데, LFS 문서에서 제시하는 가장 기본적인 형태는 다음과 같다. 이러한 파일을 만들어 `/lfs/etc`에 둔다.

```
# Begin /etc/inittab

# default run level
id:2:initdefault:
```

```
# CTRL+ALT+DEL pressed
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# /sbin/mingetty invocations for run levels
1:2345:respawn:/sbin/sulogin

# End /etc/inittab
```

만약 3.2절의 설명과 같이 `init` 프로세스가 시스템 초기화를 위해 `sysinit` 스크립트를 수행시키고 추가적인 동작 환경을 정의하도록 한다면 다음과 같은 `inittab` 예를 만들 수 있다.

```
# Begin /etc/inittab

# default run level
id:2:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rc.sysinit

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change of runlevel
# 0:halt 1:single-user 2-5:multi-user (5 may be X with xdm or other)
# 6:reboot.
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# Keyboard Request Action on special keypress (ALT-UpArrow).
kb::kbrequest:/bin/echo "Edit /etc/inittab to let this work."

# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
1:2345:respawn:/sbin/getty 9600 tty1
2:23:respawn:/sbin/getty 9600 tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
#3:23:respawn:/sbin/getty 38400 tty3
#4:23:respawn:/sbin/getty 38400 tty4
#5:23:respawn:/sbin/getty 38400 tty5
#6:23:respawn:/sbin/getty 38400 tty6

# Example how to put a getty on a serial line (for a terminal)
#
#T1:23:respawn:/sbin/getty -L ttyS1 19200 vt100
```

```
# Example how to put a getty on a modem line.
#
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3

#Example how to run portslave
#
#T0:23:respawn:+/usr/sbin/portslave 0
#T1:23:respawn:+/usr/sbin/portslave 1
#T2:23:respawn:+/usr/sbin/portslave 2
#T3:23:respawn:+/usr/sbin/portslave 3
# End /etc/inittab
```

5.2.3 초기 부트 스크립트 작성

5.2.2절의 첫 번째 inittab의 예와 같이 시스템 초기화 부트 스크립트를 사용하지 않는다면 이 절은 무시할 수 있으나, 두 번째 예와 같이 sysinit 부트 스크립트를 사용한다면 작성해야 하는 것이며, inittab 파일에 설정에 따라 부트 스크립트는 rc.sysinit이란 이름으로 /etc/init.d 디렉토리에 있어야 한다.

이러한 작업을 위해 먼저 적당한 디렉토리들을 만들어야 한다.

```
$ cd /lfs/etc
$ mkdir rc0.d rc1.d rc2.d rc3.d rc4.d rc5.d rc6.d init.d rcS.d rc.boot
```

/lfs/etc/init.d/rc.sysinit 스크립트의 예는 다음과 같다.

```
#!/bin/sh
#
# /etc/init.d/rc.sysinit:
#     run once at boot time, and
#     call all S??* scripts in /etc/rcS.d
#     in numerical/alphabetical order
#

# Set the path
PATH=/bin:/sbin:/usr/bin:/usr/sbin
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

# Source defaults.
#
. /etc/default/rcS
export VERBOSE

# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
#
trap ":" 2 3 20

#       Call all parts in order.
#
for i in /etc/rcS.d/S??*
do
        # Ignore dangling symlinks for now.
        [ ! -f "$i" ] && continue

        case "$i" in
                *.sh)
                        # Source shell script for speed.
```

```
                        (
                                trap - 2 3 20
                                . $i start
                        )
                        ;;
                *)
                        # No sh extension, so fork subprocess.
                        $i start
                        ;;
        esac
done

# For compatibility, run the files in /etc/rc.boot too.
#
[ -d /etc/rc.boot ] && run-parts /etc/rc.boot
```

/etc/default/rcS 파일은 기본적으로 (default) 설정할 환경 변수들에 대해 정의하고 있는데, 다음과 같은 명령을 통해 VERBOSE만 유효 변수로 선언해둔 상태이다.

```
export VERBOSE
```

아래 파일은 /lfs/etc/default/rcS 예이다.

```
# Defaults for the boot scripts in /etc/default
#

# Time files in /tmp are kept.
TMPTIME=0
# Set to yes if you want sulogin to be spawned on bootup
SULOGIN=no
# Set to no if you want to be able to login over telnet/rlogin
# before system startup is complete (as soon as inetd is started)
DELAYLOGIN=yes
# Set GMT="-u" if your system clock is set to GMT, and GMT="" if not.
GMT="-u"
# Set VERBOSE to "no" if you would like a more quiet bootup.
VERBOSE=yes
# Set EDITMOTD to "no" if you don't want /etc/motd
# to be editted automatically
EDITMOTD=yes
# Set FSCKFIX to "yes" if you want to add "-y" to
# the fsck at startup.
FSCKFIX=no
# Update links in all rc?.d dirs from init.d/script headers at boot?
DYNARCD=yes
```

/lfs/etc/rc.boot 디렉토리에 있는 파일의 예는 다음과 같다.

```
[rc.boot] $ ls -al
total 12
drwxr-xr-x   2 root     root         4096 Mar 20  1998 ./
drwxr-xr-x  18 root     root         4096 Jun 28 17:19 ../
-rwxr-xr-x   1 root     root          412 Jun 10  1998 0setserial

[rc.boot]$ cat 0setserial
#!/bin/sh
# Initializes the serial ports on your system
#
# Distributed with setserial version 2.12
#
```

```
STD_FLAGS="autoconfig session_lockout ^fourport spd_vhi"

SETSERIAL=/bin/setserial

echo -n "Configuring serial ports.... "

${SETSERIAL} -b  /dev/ttyS0 uart 16550A port 0x3F8 irq 4 ${STD_FLAGS}
${SETSERIAL} -b  /dev/ttyS1 uart 16550A port 0x2F8 irq 3 ${STD_FLAGS}

echo "done."

#${SETSERIAL} -bg /dev/ttyS*
```

5.2.4 기본 스크립트 작성

부트 스크립트 외에 reboot, halt, mountfs, umountfs와 같은 시스템 운영을 위한 몇 가지 스크립트 파일을 만들어야 한다.

- /lfs/etc/init.d/reboot 스크립트

```
#!/bin/sh
# Begin /etc/init.d/reboot

echo -n "System reboot in progress..."

/sbin/reboot -d -f -i

# End /etc/init.d/reboot
```

- /lfs/etc/init.d/halt 스크립트

```
#!/bin/sh
# Begin /etc/init.d/halt

/sbin/halt -d -f -i -p

# End /etc/init.d/halt
```

- /lfs/etc/init.d/mountfs 스크립트

```
#!/bin/sh
# Begin /etc/init.d/mountfs

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo -n "Remounting root file system in read-write mode..."
/bin/mount -n -o remount,rw /
check_status

> /etc/mtab
/bin/mount -f -o remount,rw /

echo -n "Mounting proc file system..."
```

```
/bin/mount proc
check_status

# End /etc/init.d/mountfs
```

- /lfs/etc/init.d/umountfs 스크립트

```
#!/bin/sh
# Begin /etc/init.d/umountfs

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo "Deactivating swap..."
/bin/swapoff -av
check_status

echo -n "Unmounting file systems..."
/bin/umount -a -r
check_status

# End /etc/init.d/umountfs
```

이렇게 만들어진 시스템 스크립트 파일들에 대해 실행 프로그램 모드로 바꾸어야 하고 적절한 링크를 만들어 넣어 주어야 한다.

```
$ cd /lfs/etc/init.d
$ chmod 755 rcS reboot halt mountfs umountfs
$ cd ../rc0.d
$ ln -s ../init.d/umountfs S90umountfs
$ ln -s ../init.d/halt S99halt
$ cd ../rc6.d
$ ln -s ../init.d/umountfs S90umountfs
$ ln -s ../init.d/reboot S99reboot
$ cd ../rcS.d
$ ln -s ../init.d/mountfs S10mountfs
```

5.2.5 파일 시스템 테이블 작성

LFS 문서에서는 다음과 같은 형태의 파일 시스템 테이블을 작성하여 /lfs/etc 밑에 fstab 파일 이름으로 두도록 하고 있으나, 대상 시스템에 따라 달라져야 한다.

```
/dev/hda5 / ext2 defaults 0 1
/dev/hda6 none swap sw 0 0
proc /proc proc defaults 0 0
```

만약 RAM 디스크 상에서 운용되는 경우라면 ZDISK나 LRP의 경우와 같은 파일 시스템 테이블이 만들어질 수 있다.

5.2.6 passwd 및 group 파일 작성

5.2.2절의 첫 번째 inittab 파일의 경우에 시스템이 부팅되면, init 프로세스는 sulogin 프로그램을 실행시키고, sulogin 프로세스는 root의 비밀번호를 물어본다. 따라서 이에 관련된 파일이 있어야

한다. /lfs/etc 디렉토리에 다음과 같은 passwd와 group 파일을 만들어 두도록 한다.

- passwd

  ```
  root:s394ul1Bkvmq2:0:0:root:/root:/bin/bash
  ```

- group

  ```
  root::0:
  ```

위 root 비밀번호는 기존 시스템에 있는 /etc/passwd 파일에서 자기가 사용하는 것을 복사하여 사용해도 된다. 위 암호코드에 대한 비밀번호는 lfs123이다.

5.2.7 Ash 쉘 만들기

root 비밀번호를 입력하여 로그인을 하면 passwd 파일에 등록되어 있는 쉘 프로그램이 실행되는데, GNU/Linux 시스템에서는 주로 bash 쉘을 사용하지만, 내장형 시스템의 최적화를 위해서는 ash 쉘을 사용한다. 대상으로 하는 시스템용으로 Ash를 컴파일하여 /lfs/bin 디렉토리에 넣어서 적절한 링크를 만들어 둔다.

```
$ ln –s /bin/ash bash
$ ln –s /bin/ash sh
$ ln –s /bin/ash csh
```

5.2.8 BushBox와 TinyLogin 설치

시스템 최적화가 필요하다면 2.1.2절과 2.1.3절에 설명한 BushBox와 TinyLogin을 설치하도록 한다.

5.2.9 부트 로더 설치

LILO이든 SYSLINUX이든 부트 로더는 사용자 선택에 따라 설치할 수도 있고 하지 않을 수도 있다. 실행 프로그램을 만들어 넣고 /etc/lilo.conf 파일을 적절히 수정하면 된다. 만약 기존 PC 데스크탑에서 GNU/Linux 시스템을 LFS 형태로 만들어 사용하는 것이라면 /etc/lilo.conf 파일만 수정하면 되고, MS-DOS 파일 시스템에서 GNU/Linux를 설치하도록 할 것이라면 SYSLINUX 부트 로더를 쓸 수도 있을 것이다.

5.3 내장형 시스템에서의 SWAP

내장형 시스템에서는 스왑 파티션을 사용하지 않아도 되지만, 사용해도 아무런 문제가 없다는 보고가 있다.[16] 그러므로 /lfs/etc/init.d/rc.sysinit 스크립트 속에 스왑 동작을 시작하게 하는 명령을 넣어줄 필요가 없다.

5.4 시스템 시험

이상의 과정으로 가장 최소한의 GNU/Linux 시스템 구성이 완료되었다. 시스템 시험을 하기 위해 다시 부팅을 시키면 root 비밀번호로 로그인할 수 있을 것이다.

이러한 상황에서는 shutdown 명령어가 동작하지 않을 수 있으며 재부팅을 위해서는 reboot –f 명령어를 사용하면 된다. 시스템 시험에 관련된 추가적인 고려 사항은 LFS 문서를 참조하도록 한다.

6 결론

내장형 시스템을 위한 GNU/Linux의 최적화 방법은 시스템 유틸리티 및 라이브러리들을 가능한 한 작은 크기로 만드는 데에 핵심이 있다. 이에 따라 2절에서 시스템 유틸리티 최적화, 단순화된 부팅, 커널 최적화, 라이브러리 최적화, 컴파일러 최적화 등의 방법들을 살펴보았다.

이러한 최적화의 방법들은 일률적으로 똑같이 적용되는 것이 아니라 목표로 하는 시스템의 목적과 기능 특성에 따라 좌우되는 것이므로 보다 효과적으로 최적화하기 위해서는 시스템 특성을 먼저 고려하는 것이 좋다. 그 결과에 따라 무의미한 최적화가 발견될 수 있어 생략할 수도 있고, 모든 최적화 기술을 다 동원해야 할 수도 있고, 상호 대칭적이어서 선택해야 하는 것도 있을 수 있다.

이러한 최적화 기술을 통해 거대하기만 했던 GNU/Linux 시스템이 매우 제한적인 하드웨어 동작 환경

속에서도 운용될 수 있도록 만들 수 있다.

참고문헌

[1] BusyBox, http://busybox.lineo.com/

[2] TinyLogin, http://tinylogin.lineo.com/

[3] Ash, http://www.debian.org/Packages/unstable/shells/ash.html

[4] Sysvinit, ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/

[5] Gerard Beekmans, "Compiler optimizations", http://www.linuxfromscratch.org/view/

[6] ZDISK, http://metalab.unc.edu/pub/Linux/system/recovery/!INDEX.html

[7] Tom Fawcett, "The Linux Bootdisk HOWTO", http://www.linuxdoc.org/HOWTO/Bootdisk-HOWTO/index.html

[8] H. Peter Anvin, "SYSLINUX – A boot loader for Linux using MS-DOS floppies"

[9] Werner Almesberger, "LILO User's guide", ftp://metalab.unc.edu/pub/Linux/system/boot/lilo/

[10] Paul Gortmaker, "The Linux BootPrompt-HowTo", http://www.linuxdoc.org/HOWTO/BootPrompt-HOWTO.html

[11] "Using the RAM disk block device with Linux", /usr/src/linux/Documentation/ramdisk.txt

[12] LRP – Linux Router Project, http://www.linuxrouter.org/

[13] LFS – Linux From Scratch, http://www.linuxfromscratch.org/

[14] Sebastien Huet, "Embedded Linux Howto", http://www.linux-embedded.com/howto/Embedded-Linux-Howto.html

[15] "Filesystem Hierarchy Standard", http://www.pathname.com/fhs/

[16] Paul Moody, "miniHOWTO Embedded Linux 1.1c", http://www.linux-embedded.com/pmhowto.html

보완 예정 항목

http://www.linuxdevices.com/articles/AT9888936014.html

http://www.fefe.de/dietlibc/

http://www.fefe.de/djb/

http://www.fefe.de/libowfat/

http://robur.slu.se/jensl/ugrep/

http://www-ec.njit.edu/~asm3072/programming.html

http://www.fefe.de/fgetty/

http://www.fefe.de/fget/

http://www.fefe.de/ncp/

http://www.fefe.de/embutils/

http://cvs.uclinux.org/uClibc.html

http://opensource.lineo.com/projects.html

Particularly impressive is the ability of Mobile Linux's cramfs to squeeze a more or less normal Linux system, complete with glibc, other libraries, the kernel and even the X windowing system into a paltry 8MB. This leaves plenty of room on the typical 32MB or 64MB storage devices used in embedded systems for applications such as Web browsers and mp3 music systems.

Cramfs saves space by optimizing inode table size and eliminating the space wasted between files in traditional filesystems. It also uses zlib compression for a better than 2-to-1 compression rate. The overhead for decompression is not especially bad because cramfs allows decompression of arbitrary blocks, rather than entire files, a function that the Linux kernel is able to exploit.

In addition to saving space, cramfs saves the appliance enduser from the prospects of waiting for fsck or worse, having to perform fsck manually after an abnormal termination. It does this by being read-only. This limitation makes some sense since flash ROM can withstand only a few hundred thousand write cycles. It could also be a security advantage in many circumstances. Still, an ongoing project is to incorporate compression into a journaling file system such as reiserfs which could be useful in some applications.

Another key to space savings is the use of BusyBox, a single executable offering lite versions of lots of standard Unix commands. Busybox will be familiar to those who have played around with floppy disk-based Linux routers, rescue disks and the like, but most users would find its utilities pretty limited.

Configuration and other data that must persist across reboots is first saved to a temporary filesystem built in RAM on a "ramfs" filesystem, another bit of Torvalds handywork. "Linus was on a filesystem spree there for a while," according to Quinlan. When it's time to commit ramfs to persistent storage, a utility called packramfs is called to stuff it into the cramfs filesystem. The ramfs is similar to a RAM disk, but supports file "limits" so that it can be prevented from overrunning available RAM space.