초기 RAM 디스크 사용하기 (initrd)

Werner Almesberger werner.almesberger@epfl.ch & Hans Lermen lermen@fgan.de 1996, 2000

서성용 pooh@kldp.org 2000년 12월 29일

시스템 설치시 모듈식의 커널 설정을 가능하게 하는 initrd 의 소개와 사용법을 담고 있다.

목 차

제 1 절 소개 **1**

제 2 절 작동 **1**

제 3 절 부트 명령행 옵션 **2**

제 4 절 설치 **2**

제 5 절 루트 장치 바꾸기 **4**

제 6 절 사용 시나리오 **5**

제 7 절 구식 루트 변경 메커니즘 **6**

제 8 절 참고자료 **6**

제 1 절 소개

initrd 는 부트 로더에 의해 RAM 디스크를 읽어 들이는 능력을 제공한다. 이 RAM 디스크는 그 후에 루트 파일시스템으로 마운트되어 프로그램들이 그것으로부터 실행될 수 있다. 그 후에, 새로운 루트 파일 시스템이 다른 장치로부터 마운트 될 수 있다. 그 후 이전의 루트 (initrd 로부터) 는 디렉토리로 옮겨지고 결국 언마운트 될 수 있다.

initrd 는 크게 시스템의 시작을 두 단계로 일어나는 것을 가능하게 하기 위해 설계되었다. 이 두 단계는 커널이 컴파일되어 내장된 최소 집합의 드라이버와 함께 올라오고, 추가의 모듈들이 initrd 에서 읽어들여지는 것이다.

이 문서는 initrd 사용에 대해 간단한 개관을 제공한다. 부트 과정에 대한 보다 자세한 논의는 [1] 에서 찾아볼 수 있다.

제 2 절 작동

initrd 를 사용할때, 시스템은 다음과 같이 부팅된다:

1. 부트 로더는 커널과 초기의 RAM 디스크를 읽어들인다.

2. 커널은 initrd 를 "보통의" RAM 디스크로 변환하고, initrd 에 의해 사용된 메모리를 풀어놓는다.
3. initrd 는 루트에 읽고-쓰기 모드로 마운트된다.
4. /linuxrc 가 실행된다 (이것은 어떤 유효한 실행파일도 될 수 있다, 쉘 스크립트도 포함하여; 그것은 uid 0 으로 실행되고 기본적으로 init 가 할 수 있는 것은 모두 할 수 있다)
5. linuxrc 가 "진짜" 루트 파일 시스템을 마운트한다
6. linuxrc 가 pivot_root 시스템 콜을 사용하여 루트 파일 시스템을 루트 디렉토리에 마운트한다.
7. 보통의 부트 과정 (예를 들면, /sbin/init 를 부르는 것) 이 루트 파일 시스템에서 수행된다
8. initrd 파일 시스템이 제거된다

루트 디렉토리를 변경하는 것이 그것을 언마운트 하는 것을 포함하지 않음에 주의하라. 따라서 initrd 에서 돌아가는 프로세스를 이 과정동안 남겨놓을 수 있다. 또한 initrd 아래에 마운트된 파일 시스템도 계속해서 접근가능함에 주의하라.

제 3 절 부트 명령행 옵션

initrd 는 다음의 새로운 옵션을 추가한다:

initrd=<path> (예, LOADLIN)

지정된 파일을 초기 RAM disk 로 읽어들인다. LILO 를 사용할 때, RAM 디스크 이미지를 /etc/lilo.conf 안에 지정해야 하며, INITRD 설정 변수를 이용한다.

noinitrd

initrd 자료는 보존되나 RAM 디스크로 전환되어 "보통의" 루트 파일 시스템 이 마운트되지는 않는다. initrd 자료는 /dev/initrd 에서 읽을 수 있다. initrd 이 있는 자료가 이 경우에는 어떠한 구조든지 가질 수 있고, 반드시 파일 시스템 이미지일 필요는 없다. 이 옵션은 주로 디버깅에 쓰인다.

주의: /dev/initrd 는 읽기 전용이고 한번만 사용될 수 있다. 마지막 프로세스가 그것을 닫자마자, 모든 자료는 풀려지고(freed) /dev/initrd 는 더이상 열릴 수 없다.

root=/dev/ram0 (devfs를 사용하지 않을때)

root=/dev/rd/0 (devfs 사용할 때)

initrd 는 루트로 마운트되고, RAM 디스크가 여전히 루트에 마운트된 채, 보통의 부트 과정이 계속된다.

제 4 절 설치

첫째, initrd 파일 시스템을 위한 디렉토리가 "보통의" 루트 파일 시스템에 생성되어야 한다. 예를 들면

mkdir /initrd

이름은 상관없다. 보다 자세한 것은 pivot_root(2) 맨 페이지에서 찾아볼 수 있다.

만약 루트 파일 시스템이 부트 과정에서 생성된다면 (즉, 만약 설치 플로피를 제작하고 있다면), 루트 파일 시스템 생성 과정은 /initrd 디렉토리를 생성해야 한다.

만약 initrd 가 어떤 경우에 마운트되지 않는다면, 그것의 내용은 다음의 장치가 만들어졌을 경우엔 계속해서 접근 가능하다 (이것이 devfs 를 사용중일 경우엔 작동하지 않음에 유의):

mknod /dev/initrd b 1 250

chmod 400 /dev/initrd

둘째, 커널은 RAM 디스크 지원과 초기 RAM 디스크 지원이 활성화되어 컴파일 되어야 한다. 또한, 적어도 initrd 로부터 프로그램을 실행하기 위한 모든 요소들(예, 실행 가능한 형식과 파일 시스템) 이 컴파일 되어 커널 안으로 들어가야 한다.

셋째, 램 디스크 이미지를 만들어야 한다. 이것은 블럭 장치에 파일 시스템을 만들고, 여기에 필요한 만큼 파일을 복사한 후, 블럭 장치의 내용을 initrd 파일 에 추가하면 된다. 최근의 커널에서는, 최소한 세 종류의 장치가 다음에 적적할다.

- 플로피 디스크 (어느곳에서나 작동하지만 괴로울정도로 느리다)
- 램 디스크 (빠르지만, 물리적 메모리를 할당한다)
- 루프백 장치 (가장 훌륭한 방법)

루프백 장치를 이용하는 방법을 설명하도록 하겠다.

1. 루프백 블럭 장치가 커널 안에 설정되었는지 확인한다.
2. 적절한 크기의 빈 파일 시스템을 생성한다. 예를 들면, # dd if=/dev/zero of=initrd bs=300k count=1

   # mke2fs -F -m0 initrd

   (만약 공간이 중요하다면, Ext2 대신에 Minix FS 를 사용할수도 있다.)
3. 파일 시스템을 마운트한다. 예를 들면, # mount -t ext2 -o loop initrd /mnt
4. 콘솔 장치를 생성한다(devfs를 사용한다면 꼭 필요하지는 않으나, 어쨌든 문제 를 문제를 일으키지는 않는다) : # mkdir /mnt/dev

   # mknod /mnt/dev/console c 5 1
5. initdrd 환경을 적절하게 사용하기 위해 필요한파일들을 복사한다. 가장 중요한 파일인 /linuxrc 를 잊지 말라. /linuxrc 의 퍼미션은 x (실행) 을 포함해야 함에 주의하라.
6. initrd 환경이 할 수 있는 정확한 작동은 리부팅 하지 않고도 다음의 명령을 이용하여 자주 테스트 할 수 있다. # chroot /mnt /linuxrc

   물론 이것은 일반적인 시스템 상태를 방해하지 않는 initrd에만 한정된다 (예, 네트웍 인터페이스를 재설정, 마운트된 디바이스에 덮어쓰기, 이미 실행중인 데몬을 다시 띄우기 등에 의해서. 그러나 그러한 chroot 된 initrd 환경에서도 pivot_root 를 사용하다는 것에 주목하라.)
7. 파일 시스템을 언마운트한다. # umount /mnt
8. 이제 initrd 는 파일 "initrd" 안에 있다. 필요하다면 압축할수도 있다. # gzip -9 initrd

   initrd 를 실험하기 위해서, 복구용 플로피 디스켓과 /linuxrc 에서 /bin/sh 로 심볼릭 링크만 추가하면 된다. 이 대신에, 작은 initrd 를 생성하기 위해서, 실험적인 newlib 환경 [2] 를 사용할 수도 있다.
9. 마지막으로, 커널을 부트하고 initrd 를 읽어들인다. 대부분 모든 리눅스 부트 로더는 initrd 를 지원한다. 부트 과정이 구식의 메커니즘과 여전히 호환되므로, 다음의 부트 명령행 파라미터들이 주어져야 만 한다:

   만약 devfs 를 사용한다면,

   root=/dev/ram0 init=/linuxrc rw

   사용하지 않는다면(rw 는 initrd 파일 시스템에 쓰기를 할때만 필요하다)

   root=/dev/rd/0 init=/linuxrc rw

   LOADLIN 의 경우에는, 단지 이것만 실행하면 된다.

   LOADLIN < kernel > initrd= <disk_image>

예) LOADLIN C:\LINUX\BZIMAGE initrd=C:\LINUX\INITRD.GZ root=/dev/ram0

init=/linuxrc rw

LILO 의 경우, /etc/lilo.conf 안에 전체 섹션 이나 혹은 해당하는 섹션에 INITRD=<path> 옵션을 추가한다. 그리고 APPEND 를 이용하여 옵션을 넘겨준다. 예를 들면,

image = /bzImage

initrd = /boot/initrd.gz

append = "root=/dev/ram0 init=/linuxrc rw"

그리고 /sbin/lilo 를 실행시킨다.

다른 부트 로더는 해당하는 문서를 참조하라.

이제 부팅을 할 수 있고 initrd 를 사용하는 것을 즐길 수 있다.

제 5 절 루트 장치 바꾸기

linuxrc 가 자신의 할일을 끝내면, 루트 장치를를 변경하여, "진짜" 루트 장치에서 리눅스 시스템을 시작함으로써 계속 진행시킨다.

그 과정은 다음의 과정을 포함한다.

- 새로운 루트 파일 시스템 마운트하기
- 그것을 루트 파일 시스템으로 바꾸고
- 이전의 (initrd) 루트 파일 시스템으로의 모든 억세스를 제거하고
- initrd 파일 시스템을 언마운트하고 RAM 디스크를 해제한다.(de-allocate)

새로운 루트 파일 시스템을 마운트 하는 것은 쉽다: 단지 현재의 루트 아래의 디렉토리에 마운트만 되면 된다. 예를 들면:

mkdir /new-root

mount -o ro /dev/hda1 /new-root

루트 변경은 pivot_root 시스템 콜을 이용하여 이뤄진다. 이 시스템 콜은 pivot_root 유틸리티 (pivot_root(8) 맨 페이지를 참조; pivot_root 는 util-linux 버전 2.10h 이상으로 배포된다 [3]) 를 통해서도 사용할 수 있다. pivot_root 는 현재의 루트를 새로운 루트 아래의 디렉토리로 변경하고, 새로운 루트를 그 자리에 위치시킨다. 이전의 루트를 위한 디렉토리는 pivot_root 를 호출하기 전에 존재해야 한다. 예를 들면:

cd /new-root

mkdir initrd

pivot_root . initrd

이제 linuxrc 프로세스는 여전히 이전의 루트를 접근할 지도 모르는데, 이 경로는 실행 파일, 공유 라이브러리, 표준 입력/출력/에러, 그리고 그것의 현재 루트 디렉토리이다. 이러한 모든 참조는 다음의 명령을 이용해 제거된다.

exec chroot . what-follows <dev/console >dev/console 2>&1

what-follow 는 새로운 루트 아래에 있는 프로그램이다. 예를 들면 /sbin/init 만약 새로운 루트 파일 시스템이 devfs 와 함께 이용될 것이고 유효한 /dev 디렉토리를 갖고 있지 않다면, devfs 는 /dev/console 을 제공하기 위해 chroot 가 불려지기 전에 마운트 되어야 한다.

주의: pivot_root 에 대한 자세한 구현 내용은 시간에 따라 바뀔수 있다. 호환성을 확실하게 하기 위해, 다음의 요소들이 관찰되어야 한다.

- pivot_root 를 호출하기 전에, 호출하는 프로세스의 현재 디렉토리가 새로운 루트 디렉토리를 가리켜야 한다
- 첫째 인자로 . 을 사용하고, 이전의 루트를 위한 _상대적인_ 경로를 두번째 인자로 사용한다
- chroot 프로그램은 이전과 새로운 루트에서 사용 가능해야 한다
- 결국 새로운 루트로 chroot 한다
- exec 명령에서 dev/console을 위해 상대적 경로를 사용한다

이제, initrd 는 언마운트되어 RAM 디스크에 할당된 메모리가 해제될 수 있다.

umount /initrd

blockdev –flushbufs /dev/ram0 # devfs 를 사용할 경우

NFS 마운트된 루트에 initrd 를 사용할 수도 있으며, 자세한 사항은 pivot_root(8) 맨 페이지를 참조하라.

주의: 만약 linuxrc 나 그로부터 exec 되는 어떤 프로그램이든지 어떤 이유에서 종료되면, 구식의 change_root 메커니즘이 불려진다("구식 구트 변경 메커니즘" 항목을 참조).

제 6 절 사용 시나리오

initrd 구현에 대한 주요 동기는 시스템 설치시에 모듈식의 커널 설정을 가능하게 하기 위해서였다. 그 과정은 다음과 같이 동작한다.

1. 시스템은 플로피나 다른 매체에서 최소화된 커널로 부팅한다(예, RAM 디스크, initrd, a.out, Ext2 FS 지원) 그리고 initrd 를 로드한다
2. /linurc 는 (1) "진짜" 루트 FS (즉, 디바이스 종류, 디바이스 드라이버, 파일 시스템) 를 마운트 하는데, (2) 배포 매체(즉, CD-ROM, 네트웍, 테잎 등) 에 필요한 것이 무엇인지 결정한다. 이것은 사용자에게 묻거나, 자동으로 조사하거나, 둘을 합친 방법에 의해 수행된다.
3. linuxrc 는 필요한 커널 모듈을 로드한다
4. linuxrc 는 루트 파일 시스템을 생성하고 정착시킨다 (이것은 아직까지는 최적의 사용할만한 시스템일 필요는 없다)
5. linuxrc 는 pivot_root 를 호출하여 루트 파일 시스템을 바꾸고 chroot 를 이용해 설치를 계속할 프로그램을 exec 한다
6. 부트 로더가 설치된다
7. 부트 로더는 시스템을 가동하기 위해 사용된 모듈의 집합을 가진 initrd 를 로드하도록 설정된다(예, /initrd 는 수정되고, 나중엔 언마운트되고, 마지막으로 이미지는 /dev/ram0 나 /dev/rd/0 에서 파일로 쓰여질 수 있다)
8. 이제 시스템은 부트 가능하고 추가적인 설치 작업이 수행된다

여기서 initrd 의 주 역할은 (쓸데없이) 부풀려진 "범용" 커널을 사용하지 않고 커널을 다시 컴파일하거나 링크하지 않고도 보통의 시스템 작업 동안 설정 자료를 재사용하기 위함이다.

두번째 시나리오는 단일 관리적인 범위안에 있는 서로 다른 하드웨어 설정을 가진 시스템에 리눅스를 돌리는 경우의 설치이다. 그러한 경우에, 커널의 최소 집합 (이상적으로는 오직 하나) 만을 만들고 설정 정보의 시스템 특정 부분은 가능한 작게 유지하는 것이 이상적이다. 이 경우, 공통의 initrd 모듈이 모든 필요한 모듈을 가지고 생성될 수 있다. 그후, 오직 그것에 의해 읽혀지는 /linuxrc 나 파일만이 다르게 될 것이다.

세번째 시나리오는 보다 편리한 복구 디스크이다. 왜냐하면 부팅시에 루트 FS 파티션의 위치같은 정보들이 제공될 필요가 없기 때문이다. 그러나 initrd 에서 로드된 시스템은 사용자 친화적인 대화창을 통해 몇몇의 정확도 검사를 수행할 수 있다(혹은 자동 탐지의 어떤 형태로서도)

마지막으로, CD-ROM 배포자들은 보다 나은 CD 에서의 설치를 위해 사용할 수 있다. 그 예로 부트 플로피를 사용하고 보다 큰 RAM 디스크를 CD로부터 initrd를 통해 부트스트래핑 하거나, LOADLIN 같은 로더를 통해서나 직접 CD-ROM 에서 부팅하고, 플로피가 필요 없이 CD 에서 RAM 디스크를 읽어들이는 것이다.

제 7 절 구식 루트 변경 메커니즘

다음의 메커니즘은 pivot_root 가 도입되기 전에 사용되었다. 현재의 커널은 여전히 그것을 지원하지만 그것의 지속적인 사용가능성에 대해서 의존해서는 _안_된다.

그것은 "진짜" 루트 장치 (즉, 커널 이미지 안에 rdev 로 설정되거나 부트 명령행에서 root=... 로 설정된 것) 를 linuxrc 가 종료할때 루트 파일 시스템으로 마운트함으로서 동작한다. initrd 파일 시스템은 그리고 나서, 언마운트되던가, 여전히 사용중이라면 새로운 루트 파일 시스템에 /initrd 같은 디렉토리가 존재할 경우 여기로 옮겨진다.

이 메커니즘은 사용하기 위해, 부트 명령 옵션인 root, init, rw 를 지정할 필요는 없다. (만약 지정되었다면, initrd 환경이 아닌 실제 루트 파일 시스템에 영향을 미칠 것이다.)

만약 /proc 가 마운트된다면, "진짜" 루트 장치는 linuxrc 안에서 새로운 루트 FS 장치의 번호를 특수 파일인 /proc/sys/kernel/real-root-dev에 기록함으로써 바뀔 수 있다. 예를 들면

echo 0x301 >/proc/sys/kernel/real-root-dev

이 메커니즘은 NFS 와 이와 유사한 파일 시스템과 호환되지 않음에 주의하라.

이 구식의 비난받는 메커니즘은 보통 "change_root" 라고 불리는 반면, 새롭고 지지받는 메커니즘은 "pivot_root" 라고 불린다.

제 8 절 참고자료

[1] *Almesberger, Werner; "Booting Linux: The History and the Future"* <ftp://icaftp.epfl.ch/pub/people/almesber/booting/bootinglinux-current.ps.g>

[2] *newlib package (experimental), with initrd example* <ftp://icaftp.epfl.ch/pub/people/almesber/misc/newlib-linux/>

[3] *Brouwer, Andries; "util-linux: Miscellaneous utilities for Linux"* <ftp://ftp.win.tue.nl/pub/linux-local/utils/util-linux/>