Demystifying Boot Disks

Eli Billauer

eli@billauer.co.il

Lecture's outline

- Introduction
- The RAM disk and the loop device
- What's in a rescue disk
- What you need to know to successfully modify a rescue disk
- What's in a boot disk, and how to modify it

General remark: A large part of this lecture is example-oriented. Where an example is given the explanations are omitted from the slides

Why boot/rescue disks?

The general idea: Running Linux without mounting hard disks

- Repairing a damaged system ("rescue" disks)
- Running Linux for a single ad-hoc application
- Running on a diskless computer
- Trying a newly compiled kernel

Where to find what

- Documentation
  - Bootdisk-HOWTO
  - BootPrompt-HOWTO
  - The LILO mini-HOWTO
  - `/usr/src/linux/Documentation/ramdisk.txt`
  - `/usr/src/linux/Documentation/initrd.txt`
  - `/usr/src/linux/init/main.c` (Source code of boot process)
- Boot images
  - On installation CD's, typically under `images/`
  - All over the web

What happens when you boot

- The kernel is loaded, uncompressed and run
- The kernel *might* do some Hocus-Pocus (explained later) to run a `linuxrc` initialization executable
- The kernel mounts a root directory on a device, usually known in advance
- The First Process is launched, trying in the following order: `/etc/init`, `/bin/init`, `/sbin/init` or other defaults.
- The boot is now in the hands of this process.

As we shall see, there are endless variations for this

Different ways to boot up

- From hard disk
- Boot kernel from floppy, root on hard disk
- Boot kernel from floppy, root on floppy disk
- Boot kernel from floppy, root on **RAM disk**.
  - The root image comes from boot disk
  - The root image comes from a second disk
  - Both...
- Booting from CD-ROM is actually booting a floppy disk image written on CD. (See `mkisofs` documentation)
- `loadlin.exe` (used by linux4win)

The RAM disk

This will create a 4 Mbyte RAM disk, and mount it.

```
# dd if=/dev/zero of=/dev/ram0 bs=1k count=4096
4096+0 records in
4096+0 records out
# mke2fs /dev/ram0
mke2fs 1.10, 24-Apr-97 for EXT2 FS 0.5b, 95/08/09
(...)
# mkdir /mnt/ram
# mount /dev/ram0 /mnt/ram
```

The RAM disk (cont.)

- After these instructions ,we can use `/mnt/ram` like any usual disk.
- The memory taken by a RAM disk is never returned (reboot...)
- This is implicitly done by the kernel when we boot into a RAM disk but:
  - The kernel copies the boot image from floppy to RAM disk. Something like:

    ```
    cat /dev/fd0 | zcat > /dev/ram0
    ```

  - Then the kernel mounts the image as root directory.

The loop device

How to create a 1 MB loop disk

```
# dd if=/dev/zero of=loopfile bs=1024 count=1024
1024+0 records in
1024+0 records out
# ls -l loopfile
-rw-rw-r--   1 root root  1048576 Dec 14 22:32 loopfile
# mke2fs loopfile
mke2fs 1.10, 24-Apr-97 for EXT2 FS 0.5b, 95/08/09
loopfile is not a block special device.
Proceed anyway? (y,n) y
(...)
# mkdir mountdir
# mount -o loop loopfile mountdir
```

The loop device (cont.)

- Note that `mount -o loop` is actually an abbreviation for

```
# losetup /dev/loop0 loopfile
# mount /dev/loop0 mountdir
```

- ...except that `mount` looks for an available loop device (`/dev/loop0, /dev/loop1, ...`)
- Loop devices depend on the `loop.o` kernel module

Exploring the rescue disk

- If we want to see what's in a certain diskette, we first raw-copy it to a file:

```
# cat /dev/fd0 > from-diskette.gz
# gunzip from-diskette.gz

gunzip: from-diskette.gz: decompression OK,
trailing garbage ignored
```

- Using `cat` to raw-read diskettes is "politically incorrect", so it's more common to see is the equivalent

```
# dd if=/dev/fd0 of=from-diskette.gz bs=1k
```

- We next look at the rescue disk image which comes with the installation CD.

Exploring the rescue disk (cont.)

```
# cp /mnt/cdrom/images/rescue.img rescue.gz
# ls -l rescue.gz
-r--r--r--   1 root  root  1401934 Dec 15 01:11 rescue.gz
# gunzip rescue.gz
# ls -l rescue
-r--r--r--   1 root  root  4194304 Dec 15 01:11 rescue
# mkdir temp
# mount -o loop rescue temp
# cd temp
# ls -F
bin/         lib/         proc/        usr/
dev/         lost+found/  sbin@
etc/         mnt/         tmp/
```

Exploring the rescue disk (cont.)

```
bin:
ash*            gunzip@         modprobe*       sh@
badblocks*      gzip*           mount*          swapoff@
cat*            head*           mt*             swapon*
chmod*          ifconfig*       mv*             sync*
chroot*         init*           open*           tac*
cp*             insmod*         pico*           tail*
cpio*           ln*             ping*           tar*
dd*             ls*             ps*             traceroute*
df*             lsmod*          rm*             umount*
e2fsck*         mkdir*          route*
fdisk*          mke2fs*         rpm*
grep*           mknod*          sed*
```

Exploring the rescue disk (cont.)

```
dev:
console   hdb1    hdd15   sda11   sdc5
fd@       hdb10   hdd16   sda12   sdc6
fd0       hdb11   hdd2    sda13   sdc7
fd0D360   hdb12   hdd3    sda14   sdc8
(...)
fd0h360   hdb4    null    sda7    sde5
fd0h720   hdb5    ram     sda8    sde6
fd1       hdb6    ram0    sda9    sde7
fd1D360   hdb7    ram1    sdb     sde8
(...)
```

Not all devices are shown here. If you'll need a certain device, make sure it's in the list!

Exploring the rescue disk (cont.)

```
etc:
fstab      mtab        protocols  services   termcap

lost+found:

mnt:
floppy/  image/

proc:

tmp:

usr:
bin@    lib/    sbin@   share/
```

Exploring the rescue disk (cont.)

```
lib:
ld-2.1.1.so*            libnss_dns-2.1.1.so*
ld-linux.so.2@          libnss_dns.so.1@
libc-2.1.1.so*          libnss_dns.so.2@
libc.so.6@              libnss_files-2.1.1.so*
libcom_err.so.2@        libnss_files.so.1@
libcom_err.so.2.0*      libnss_files.so.2@
libdb.so.2@             libproc.so.2.0.0*
libdb1-2.1.1.so*        libtermcap.so.2@
libe2p.so.2@            libtermcap.so.2.0.8*
libe2p.so.2.3*          libuuid.so.1@
libext2fs.so.2@         libuuid.so.1.2*
libext2fs.so.2.4*
(...)
```

Exploring the rescue disk (cont.)

```
usr/lib:
rpm/

usr/lib/rpm:
macros   rpmpopt  rpmrc

usr/share:
terminfo/

usr/share/terminfo:
l/

usr/share/terminfo/l:
linux
```

Changing the rescue disk

What we'll talk about:

- Creating a initrd disk image
- Dynamic linking
- Kernel Modules
- Adding a `/dev` device

Creating an initrd RAM disk image

After adding and/or removing the files, wrap it up by:

```
# umount temp
# gzip rescue
# cat rescue.gz > /dev/fd0
```

Size considerations:

- If you modify an existing RAM disk image, you can't exceed the allocated file system size.
- You may create a bigger filesystem of your own, and tell the kernel to prepare a larger RAM disk with the `ramdisk_size=` kernel argument.
- The bigger problem is to fit the gzipped file into a floppy disk.

Dynamic Linking

- If you want to add an executable to your rescue disk, make sure the libraries are there too!

```
# ldd /sbin/ifconfig
    libc.so.6 => /lib/libc.so.6 (0x40006000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x00000000)
```

- We then need to verify that the libraries exist, for example:

```
# ls -l /lib/libc*
(...)  root  root  /lib/libc-2.0.7.so*
(...)  root  root  /lib/libc.so.6 -> libc-2.0.7.so*
(...)
```

Dynamic Linking (cont.)

- Dynamic linking can be bypassed by compiling the source with `gcc -static ...`
- This creates an independent executable

Kernel Modules

- If you add a new executable, it might require a specific kernel module. `lsmod` might help to find out which.
- Kernel modules might be implicitly needed to access file systems, hardware, etc.
- There's no automated way to know which modules you'll need
- There will (probably) be no `kerneld` running in rescue mode.
- There are two ways to get the modules in place, then:
  - Copy the modules into the rescue disk and `insmod` as needed.
  - Have all possibly needed modules loaded from the **boot disk**. (You won't waste your precisous rescue disk space on this!)

Kernel Modules (cont.)

- The kernel modules are organized under `/lib/modules`. The directory structure goes deep
- Some modules depend on others. Attempting to `insmod` a module before a module it depends on will result in an error message regarding unresolved symbols.
- A quite easy-to-understand makefile-type list of module dependencies appears in the `modules.dep` file, typically `/lib/modules/preferred/modules.dep`. This file is created by the `depmod` command.

Adding a /dev device

```
# ls -l /dev/hda*
brw-rw----   1 root  disk  3,  0 May  5  1998 /dev/hda
brw-rw----   1 root  disk  3,  1 May  5  1998 /dev/hda1
brw-rw----   1 root  disk  3,  2 May  5  1998 /dev/hda2
...

# ls -l /dev/ttyS*
crw-r--r--   1 root  root  4, 64 May  5  1998 /dev/ttyS0
crw-r--r--   1 root  root  4, 65 May  5  1998 /dev/ttyS1
crw-r--r--   1 root  root  4  66 May  5  1998 /dev/ttyS2
crw-r--r--   1 root  root  4, 67 May  5  1998 /dev/ttyS3
```

Adding a /dev device (cont.)

Now we'll demonstrate how to add a loop device to our rescue disk

- First we find out the type, major and minor of the device:

```
# ls -l /dev/loop0
brw-rw----   1 root  disk  7, 0 May  5  1998 /dev/loop0
```

- Aha! It's a block device with major 7 and minor 0. We can now create one on the boot disk.

```
# mknod /mnt/my-rescue-disk/dev/loop0 b 7 0
```

- A list of majors and minors can be found in `/usr/src/linux/Documentation/devices.txt` (try also `/usr/src/linux/include/linux/major.h`)

Don't say I didn't warn you

Dealing with boot & rescue disks should be harmless, but there are plenty of ways to screw up your system while playing with them:

- Write to `/dev/hda` instead of `/dev/fd0`
- Tamper with your system files instead of your image files
- You are root most of the time. Plenty of chances to do stupid things.

Exploring the boot disk

There are two kinds of boot disks:

- Kernel and boot image raw-written on sectors (yuck!)
  - The old way to create boot disks
  - Disk is setup with `rdev`
  - We won't get into this further
- Disk contains an ext2 file system and is LILO'ed
  - The kernel and boot image appear as files
  - The file system is there only so that the `lilo` executable will have a correct environment.
  - Kernel arguments may be sent during bootup, as usual

Exploring the boot disk (cont.)

We now look at a **customized** boot disk

```
# mount /dev/fd0 /mnt/floppy
# cd /mnt/floppy
# ls -RF
.:
boot/  etc/        lost+found/
dev/   initrd.img  vmlinuz-2.2.14-15mdk
boot:
boot.0200  boot.b  map  message
dev:
fd0  hda7
etc:
lilo.conf
```

Exploring the boot disk (cont.)

```
# cat etc/lilo.conf
boot=/dev/fd0
timeout=100
message=/boot/message
prompt
image=/vmlinuz-2.2.14-15mdk
     label=linux
     root=/dev/hda7
     append=" hdc=ide-scsi "
image=/vmlinuz-2.2.14-15mdk
     label=rescue
     append="load_ramdisk=2 prompt_ramdisk=1 hdc=ide-scsi"
     root=/dev/fd0
     initrd=/initrd.img
```

Exploring the boot disk (cont.)

Did I just count **TWO** RAM disks?

- The kernel is booted from the boot disk
- `initrd.img` is uncompressed and loaded into the RAM disk, which is mounted as root
- `/linuxrc` is executed
- The kernel prompts for a second disk
- The RAM disk is now loaded from `/dev/fd0` (compressed raw data, as we saw before). This is the "rescue disk"
- Root is **remounted** with the new RAM disk.
- `/etc/init`, `/bin/init`, or `/sbin/init` is executed, whichever is found first.

Exploring the boot disk (cont.)

We now look at the RAM disk image on the **boot disk**

```
# cp initrd.img ~/
# cd ~/
# mv initrd.img initrd.gz
# gunzip initrd.gz
# mkdir rd
# mount -o loop initrd rd
```

Exploring the boot disk (cont.)

```
# ls -rF rd
linuxrc*  lib/  etc/  dev/  bin/
# ls -RF rd
rd:
bin/  dev/  etc/  lib/  linuxrc*
rd/bin:
insmod*  sh*
rd/dev:
console  null  ram  systty  tty1  tty2  tty3  tty4
rd/etc:
rd/lib:
fat.o       lockd.o  ntfs.o  slhc.o    supermount.o
ide-scsi.o  msdos.o  ppp.o   sunrpc.o  vfat.o
```

Exploring the boot disk (cont.)

The `linuxrc` file:

```
#!/bin/sh
echo "Loading ide-scsi module"
insmod /lib/ide-scsi.o
echo "Loading fat module"
insmod /lib/fat.o
echo "Loading vfat module"
insmod /lib/vfat.o
...
echo "Loading lockd module"
insmod /lib/lockd.o
```

Exploring the boot disk (cont.)

Hey! I didn't see any link libraries!

```
# ldd rd/bin/sh
        not a dynamic executable
# ldd rd/bin/insmod
        not a dynamic executable
```

- Both `sh` and `insmod` are statically linked, to avoid having the entire libraries.
- These files can probably be found in your filesystem as `/bin/ash.static` and `/sbin/insmod.static`

Changing the boot disk

- You might want to create your own with `mkbootdisk`
  - It's recommended to edit the `mkbootdisk` script so it adds the modules you want, rather than guessing them
  - Most of the script deals with picking the right modules. If you skip that part, and just choose the modules, it's quite easy
- If you want to add modules by hand, it's only a matter of adding the module in the `lib` directory, and modifying `linuxrc` accordingly.
- Be sure to load the modules in the right order!
- When finished, write the new compressed initrd.img to your floppy, **and run LILO**