MBR, Bootloaders and Booting

Andy Pepperdine

The process of getting your PC from a state of inertness without power, to one where it actively does something is known as “booting”, triggered by the act of switching on the power. I hope this paper will remove some of the mystery of what happens, and enable you to make some changes where they are necessary to do what you wish to accomplish, or maintain what you have after an important change. The earliest computers called the process “Initial Program Load”, or other similarly unimaginative name; but the idea of pulling oneself up by ones bootstraps was such a good image, that it became known as “bootstrapping” and finally “booting”.

But first, a minor warning. Almost every system seems to be different from every other. So the descriptions here will have to be read more as examples to give a feel for the theory. Your own machine will almost certainly have its own methods of doing things in detail. Look around in your own directories to see what applies.

The overall picture

Computers, these days, consist of various bits of hardware that closely interact under the influence of a program that is interpreted by the central processing unit (cpu). For the cpu to get at the program, it must reside in the local memory, or random access memory (RAM). The cpu does not see what is on disks or other external devices. It can read and write only what is in RAM. Other parts of the system access the outside world of disks, memory sticks, optical drives, networks, etc. but all under the direction of the cpu acting on behalf of the instructions in the program.

So the question is “How does the program get into memory at the very beginning?”

When the power is first applied, a part of the BIOS is triggered into action. We will assume for the purposes of this software oriented paper that the way in which the BIOS itself gets going is a piece of magic that works, apart from saying that the process is remarkably similar to the boot process but on a “lower” level. The BIOS will then run down the list of possible devices it knows about, asking each in turn to read the first data block off and into a specific position in RAM. When one of them succeeds in doing so, the BIOS kicks the cpu, pointing it at the start of the data just read in. The cpu can then read its first instruction and start its standard cycle of “read – execute – move to next instruction”.

This first data block must contain a suitable set of instructions that are sufficient to pull into memory all the other stuff that will be needed to run the whole operating system that you need for all your applications. There is typically not very much room available in that first block, and the only thing it does is to load a larger lot of code and transfer control to that code. This second stage will have enough to be able to see what alternatives there may be for what is needed, and will load the final lot of code and pass control over to that.

On a standard PC running Linux, the first data block is known as the Master Boot record (MBR), the second stage is the called the “bootloader” (for example Grub or Lilo), and the final one is the Linux kernel itself.

When the kernel starts up, it also must bootstrap its own data areas and file systems before it can ask you what you want to do. After some simple initialisation of its own data, it creates the first process, known as “init” in Unix systems. This “init” process will then control the whole of the execution of the machine from that point on. It will read the various scripts from the file systems to determine what other tasks need to be readied and start individual processes to handle each of these in turn.

On Windows systems, the MBR will pass control directly to the Windows kernel and there is no equivalent to the bootloader. It is not required in that case. But equally, the MBR just loads the Windows initialisation code directly and gives no further chance to intervene.

On systems that can boot into a number of different operating systems, there will be variations on the theme. For example, to keep things well separated, it may be reasonable to “chain” bootloaders together so they execute one after the other before reaching the one that will present your desktop to you.

Master Boot Record

The MBR contains a series of instructions that are very specific. You will not need to know what is in them except to be aware that it cannot be modified. Once it is set up for what you have asked, it must be left alone or you risk losing the ability to boot your machine. The real work is done in the next stage, the one which the MBR loads to execute.

However, since new machines will have Windows installed, the MBR has been tuned to just load up Windows. To set up a machine with the option of running Linux, the MBR must be overwritten with one which will load the bootloader, which will in turn decide which one to select.

The MBR contains the very first code to be executed, known as the primary bootloader. It also contains the partition table when it is on a hard disk. Since the standard size of sectors (the basic unit for reading and writing) is (or was) 512 bytes, the amount of code available is only 446 bytes. This code searches the partition table for a partition marked with the boot flag, and then looks in there to find the next stage to load.

GRUB

The GRand Unified Bootloader (GRUB) is the one which most modern Linux distros provide as the default when installing their system.

There are in fact two versions. The older one is simpler to understand, but is more dangerous to change as it requires directing editing by a human of the relevant files. If you get it wrong, you can lose the ability to boot and have to re-install GRUB. This version is known as just GRUB, or nowadays as GRUB legacy. To get a handle on what is happening, I will describe its workings as it leads to a clearer view of the later version.

The modern one is called GRUB 2, and is what you will get in Ubuntu, for example. In this version, the crucial files are held as editable files in the system, and not in the bootloader space. So, after editing, you must run another process to transfer this information into the bootloader proper, but in doing so sanity checks can be made and so it should be safer. The actual bootloader scripts are more complex, which is why it is a good idea to let software work out what is needed.

GRUB Legacy

There are still reasons why this should be used. For example, if you have a machine with two separate Linux distros installed, and you wish to keep them apart, then by using GRUB legacy to decide which of the two systems to load, it can transfer to the relevant GRUB 2 for each in turn. GRUB 2 can then decide whether to load in safety mode, or run diagnostics, independently of any other system installed, reducing the chances of messing up the other system.

The active files during the boot process are all found in the directory /boot/grub/ but the key one is called menu.lst. In there is the list of all the possible systems that can be loaded by this bootloader. A simple example of such a file contains:

```
root (hd0,0)
default = saved
timeout = 10

title Ubuntu 11.04 (Natty Narwhal)
  root (hd0,1)
  chainloader +1
  savedefault

title Ubuntu 10.10 (Maverick Meerkat)
  root (hd0,2)
  chainloader +1
  savedefault
```

In this almost trivial case, the GRUB legacy list is used merely to pass on the boot process to other booting.

The “title” lines provide text to display in a menu for the user to select; they also serve to start the description of a menu item to be presented to the user. When the user does select one, GRUB looks at the corresponding description, finds the appropriate disk and partition (the “root” parameter), and performs the actions defined. Here it is just to treat it as a chain of booting, and transfer control to another GRUB bootloader without further analysis or checking. The final lines just save the default menu so that the next time the machine is booted, it will default to what was used last time (“default = saved”). The timeout is the number of seconds that GRUB will wait before assuming the default.

The chainloader is what you use when invoking a non-Linux system, as GRUB cannot then do any checks on validity and sanity. For example, to boot a Windows system, the following lines might be used:

```
rootnoverify (hd0,0)
makeactive
chainloader +1
boot
```

We use “rootnoverify” here to stop GRUB looking for a valid file system, as it will not know what to expect. Instead, it will just transfer to the initial record on the partition described.

GRUB Legacy is also used by Puppy as being simple, and here is an example they provide:

```
# GRUB configuration file '/boot/grub/menu.lst'.
# generated by 'grubconfig'.  Sun Mar  6 11:20:09 2011
#
# Start GRUB global section
#timeout 30
color light-gray/blue black/light-gray
# End GRUB global section
# Linux bootable partition config begins
```

```
  title Puppy Linux 520 (on /dev/sda1)
  root (hd0,0)
  kernel /boot/vmlinuz root=/dev/sda1 pmedia=atahd
#  kernel /boot/vmlinuz root=/dev/sda1 ro vga=normal
# Linux bootable partition config ends
title Install GRUB to floppy disk (on /dev/fd0)
pause Insert a formatted floppy disk and press enter.
root (hd0,0)
setup (fd0)
pause Press enter to continue.
title Install GRUB to Linux partition (on /dev/sda1)
root (hd0,0)
setup (hd0,0)
pause Press enter to continue.
title -     For help press 'c', then type: 'help'
root (hd0)
title -     For usage examples, type: 'cat /boot/grub/usage.txt'
root (hd0)
```

Here we see that the first character in the line can be a '#' sign and that means the line is a comment line, not a relevant line for execution. The important description in this case is under the title “Puppy Linux 520...”, where it states which device and partition to look at (root (hd0,0)), and where to find the kernel (/boot/vmlinuz) in that partition.

GRUB 2

The follow-on version of GRUB boots systems in much the same way, but the files are all generated for another set of files that you can edit. Do not edit the contents of /boot when using GRUB 2.

The start process is the same, but the configuration file that does the same job as menu.lst used to do, is in /boot/grub/grub.cfg but it now contains more and is a more complex script. Its contents are derived by the program update-grub from the contents of the file /etc/default/grub using templates found in the directory /etc/grub.d. This indirect structure gives a little more safety against inadvertent mis-editing of the grub control files, although not much.

An example of a grub file in the /etc/default directory is:

```
# If you change this file, run 'update-grub' afterwards to
update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""

# Uncomment to enable BadRAM filtering, modify to suit your
needs
```

```
# This works with Linux (no patch required) and with any kernel
that obtains
# the memory map information from GRUB (GNU Mach, kernel of
FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card
supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx"
parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"

# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"
```

As you can see, the file contains largely values of keywords. The syntax is the same as that for a shell. Comments are palced on lines beginning with a '#' symbol and they are useful to identify what to change. More information is available in the info pages accessible from the command listed in the initial comment of the file.

If you are having trouble with the resolution on the display during the boot process, this is the place to look and change.

From this file, update-grub collects the templates from the directory /etc/grub.d and for each template it generates a piece of GRUB shell script, and puts them all in the /boot/grub/grub.cfg file. If you look there, you will see the pieces surrounded by comments indicating which file each part came from. Various parameters set by the file /etc/default/grub are used to govern the actual output these templates produce.

The grub.d directory contains a files with a particular naming convention. Here is a list of the files on the system I'm using right now:

```
00_header
05_debian_theme
10_linux
20_linux_xen
20_memtest86+
30_os-prober
40_custom
41_custom
README
```

The README file is not a template, but all the others are. They are processed in the order listed above, and only those starting with two digits are used to build up grub.cfg to control the booting process. Each file is executable, in fact they are usually shell scripts, and when executed they produce output that is to be inserted in the grub.cfg file.

You will very rarely, if ever, need to change these scripts, but if you are doing something more advanced, then the facility is there to enable you to add your own menu items by adding or editing them.

When GRUB executes during the boot process, it will perform the code in the order it is found, and this will be the order in which the templates are processed.

Further reading and help

http://www.dedoimedo.com/computers/grub.html gives an excellent introduction to the whole process of booting from the user's perspective. The URL is to the GRUB legacy description, from which there is a link to one for GRUB 2.

If things go wrong, rescue is at http://www.supergrubdisk.org/super-grub2-disk/

If GRUB gives error messages during the boot process, http://www.linuxselfhelp.com/gnu/grub/html_chapter/grub_13.html#SEC101 may help determine the place and possible cause of the error.