A Little Bit about Linux And a Lot about Boot Up

Version 0.7.0

Orr Dunkelman

orrd@vipe.technion.ac.il

A Little Bit about Linux

Linux is an open source UNIX operating system.

Open source is an old concept.

During the 1980's Richard Stallman has founded the Free Software Foundation. The FSF started a project called GNU (GNU's Not Unix), which tried to create an open source UNIX.

Open source programs are distributed with their source code. This has many practical advantages, and many important companies nowadays support this idea like IBM, SUN, etc.

Linux itself was first distributed in 1991 by a Finnish student named Linus Torvalds. It was a kernel and some of Richard Stallman's GNU project utilities.

A Little Bit about Linux (cont.)

Since 1991 Linux has evolved in many fields. For example, back in 1996 if you needed some device driver, you had to write it down yourself, or to find a fellow Linux-user who did that. Nowadays, some of the big hardware vendors release also Linux drivers, and it is easier to find such drivers.

Another field Linux has greatly evolved in is the distributions field. Several years ago, you had to download most of the stuff you wanted on your own, compile it on your own, install on your own, etc. Today, you can enter most of the computer stores and buy a *Linux Distribution*. A distribution is a Linux system packed together with several thousands of utilities, already compiled, easy to install.

Linux Today – What Can You Do With Linux?

Linux is a very stable operating system. Thus, many routers use embedded Linux systems. Many firewalls (computer security elements) are based on Linux, even if the only way to operate them is using some Explorer-only Java applet!

Linux is used whenever stability/performance ratio is a crucial requirement. For example, many web servers (including www.google.com, and many other big internet servers) use Linux as their operating system (usually with the open source Apache web server).

Linux Today – What Can You Do With Linux? (cont.)

Another interesting use for Linux is as a file server. In organizations where you have many users, and you want all the files to be kept in one place so everyone could access them, you use a file server which stores the files. Interesting enough, in many places, even the windows users access a Linux file server without even noticing the file server is a Linux system.

The most important use for the “home” user is a stable, functional, Hebrew supported, works on old hardware as well, requires less hard-disk and memory size, fully graphical interface, UNIX operating system.

Linux Today – Where Can I Meet Linux?

Today, you can find Linux systems in many work places.

For example, Intel's networking unit is using Linux as their development platform. Intel is not alone, and many other big companies have also chose Linux as their development platform.

Those of you who happen to wander into some Internet Service Provider server room would probably meet a Linux box which controls the routing and the service each user gets (like in ACTCOM, Netvision, etc.).

In several places where high performance computing is needed there are places were many regular PCs are used. Each of the PCs run Linux with some extension to make all the PCs one big cluster (a virtual Supercomputer).

A Bit on the Boot Up Sequence

When you power on the computer it starts a procedure which ends in a login window. Actually the login window is a process that the operating system has spawned.

The process of loading the operating system to the memory is very complicated and tricky. We start with some hardware boot handler, which passes the control to a software handler and end in a fully functional operating system.

Why Do We Need a Software Boot Handler?

If the hardware boot handler is good enough to pass control to the software handler, why it is not used to pass control directly to the operating system?

Boot Process – Theory Before Practice

Let us assume that we have a copy of the operating system ready to be loaded into the main memory. In Linux we call this file *kernel image*, in DOS it is the *MSDOS.SYS* and *IO.SYS*, and each and every operating system has this kind of file(s).

When the computer is off, this file is stored on the hard-drive. The hard-drive stores the files according to some file system. To read this file system we need the operating system's "read from hard-disk" functions.

These functions are available in the kernel image file, which can be accessed as soon as the image is loaded into memory...

To solve this endless loop, we use the concept of *bootstrapping*.

Bootstrapping

Bootstrapping is the art of using oneself to pull oneself out of the mud.

Let us assume that we start with a small place where all data is stored in pre-determined universally known format. We put in this small place a small program which knows how to access the hard-disk itself and load some additional program. This additional program can read the file-system and load files into the memory. And so it does: it reads the kernel image and writes it into the memory. Then this program hands over the computer to the kernel image which is a functional operating system.

Boot Process for a Intel Processor

For x86 Intel processors, powering them on (or pressing RESET) will cause the BIOS (Basic Input/Output System) to perform the POST (see later). After the POST is finished the operation *jmp ADDR* is loaded into the CPU, where *ADDR* is a constant address, which is usually '0x07C000'.

The BIOS can control keyboards, screens, disk drivers and several hardware interfaces (parallel and serial ports or nowadays also USB).

The BIOS search in all the partitions it can access (according to pre-determined order) a boot sector (boot sector has a special value in a specific address).

The Role of the BIOS

The BIOS has 3 (very) important and major purposes:

- POST - Power On Self Test. This step is just to make sure that everything is fine. Some of the errors can cause the system to halt, and some of them are only reported.
- OS Loader - The BIOS is programmed to scan all possible boot devices (floppy drives, hard-disks, CD-Rom's and even network cards) to look for a boot program.
- Basic I/O Services - the BIOS can be used to access the system's basic hardware. Linux usually use the BIOS during the initialization, but afterwards prefers to use its own handlers.

The OS Loading – First Phase

After the BIOS finds boot record (the place where the boot sector contains a OS-loader) it reads the boot sector and places it at 0x07C00. Boot records has 512 bytes. 446 of which are the loader itself, 64 bytes are the partition table, and the last 2 bytes are the signature of a boot record (0x55AA).

We shall focus on one of the Linux loaders - the *LILO* program. Once LILO is placed at 0x07C00 and executed it copies itself to the address 0x96A00 and sets a stack (0x96A00 downto 0x96800). It also loads the secondary boot loader at address 0x96C00 and transfers the control to it.

The OS Loading – Second Phase

The secondary boot loader loads the descriptor table at 0x98800 (which kernels are available), and waits for the user's input. LILO supports up to 16 different images. The user has to choose which of the possible images he wants to load.

Actually these 16 images might contain a direction to a secondary loader of a non-Linux system like Solaris, MacOS, FreeBSD, or even Microsoft operating systems.

When choosing the image to be used, we can also choose to pass parameters to the kernel.

The OS Loading – Second Phase (part II)

After choosing which kernel (and operating system) to load, the loader loads some setup information at 0x90000 and 0x90200. Then after everything is ready for the kernel itself, it uncompress it to address 0x10000. This address might change to 0x100000 if the kernel is compiled as “high”.

If the loaded image is a Linux kernel image then the control is transfered to the setup code (0x90200). For other OSes the boot sequence is a bit more complex, but is essentially the same.

OS Loading – Second Phase – Reporting Errors

Murphy's Law: Anything that can go wrong will go wrong.

Like in any good program, errors will occur. The main problem which LILO faces is how to report the errors. Long error messages are out of the question. Even outputting an error code that was not returned by the BIOS is very limited.

The solution: LILO only announces success. If some stage fails, you don't get the success message.

If the first loader is not loaded, then the system halts, and no output is shown. After the successful loading, the first loader outputs L.

OS Loading – Second Phase – Reporting Errors (2)

After the first loader is loaded, it tries to load the second loader. If this step fails, it is usually problem with reading the media (either a media failure or bad geometry), and the error code returned by the BIOS is shown.

When the second loader is loaded, the first loader who is still in charge outputs I. If the system is stuck at this point, it means that the first loader did not succeed to pass the control to the second loader.

When the second loader starts to run it outputs L. If the system halts at this point then the descriptor table is unreachable. If the second loader outputs L? it means that the loader was not loaded into the right place. L- means a corrupted descriptor table.

OS Loading – Second Phase – SUCCESS!

If everything went just fine then the second loader outputs the final O and waits to the user's selection of the kernel.

As mentioned before, once the kernel is chosen, the system loads it to the memory while uncompressing it. Then the control is given to the uncompressed kernel.

Question: Why does Linux compress its kernel?
The boot sequence is very complicated as is. Adding complexities by using compression seems very weird.

Compressing Kernel – A Big (small) Reason

In the system this lecture was written (Linux 2.4.18, 256 MB memory, dual Pentium 300 Mhz) the size of the compressed image file is 850 KB. The fully functional system (which supports SCSI RAID hard-drives, fully functional TCP connections and more) can be loaded onto a diskette!

This means you can get a fully equipped operating system that can be carried around on a diskette.

Another important reason is x86 design. When the processor starts to run it is in *real mode*. Thus, it can access only 1 MB of the memory (bits 20–31 of the addresses are all set to 0). If the kernel image is bigger than 1 MB, it requires special dealing.

Linux uses additional tricks to load itself (like using RAM disks), some of them are very useful to bypass these restrictions.

OS Loading – Third Phase

After the kernel gets the control it starts its initialization.

Basically this phase is quite similar for all OSes. The kernel sets its real-memory handling functions (which differ from the virtual memory handlers), the interrupt handlers (starting with the basic ones).

Then the kernel starts to load drivers. Some of them are part of the kernel and some can be used as modules (and thus can be replaced without compiling the entire kernel again).

Now the kernel launches various “helpers” equivalent to XINU’s butler and then launches the init process.

The INIT Process – Creator of all Processes

In the beginning there was INIT, and the process table was empty, but the INIT. And the INIT said, Let there be daemons: and the daemons were loaded. And INIT saw the daemons, that they were good. And the INIT said, Let there be user front-ends. And INIT called the front-ends tty and pts.

INIT is the first process in all UNIX systems. As UNIX system requires that each process would have a father, INIT is the first in this tree of processes.

INIT loads all the services that the OS is supplying to the user (through several interfaces), and then loads the login screens[a].

[a] This can be avoided, if you don't really like it

Linux & Open Source vs. Propriety OSes

There are many advantages to the open source environment Linux is being developed in.

- More programmers. Each user can suggest modifications and improvements.
- More flexibility. You can alter the code to your needs.
- Security. No Viruses! Once a security hole is found – you can close it yourself!
- Transparency. You know what you run.
- Cheaper. Open Source program are usually cheaper.

And I'm not counting Linux's advantages as is – portability, faster, best functionality-stability ratio, and much more.

Linux in Israel & Haifa

Linux had a slow start in Israel. The Hebrew support was not good till several months ago. Hence, Linux was used where Linux Gurus have walked the earth.

Nowadays, when the Hebrew support is over the ”functional, annoying but functional” level, and the fact more people were exposed to it, Linux is a valid option for many places.

The first resource about Linux in Israel is the IGLU website *http://www.iglu.org.il*. There you can find information about various Linux events in Israel, Hebrew in Linux, and more.

Another place to look for material is the HAIFUX website *http://linuxclub.il.eu.org*.

WARNING: Linux users in Israel have several mailing lists. Sending the right question to the wrong list might cause your flaming!

References

[1] HAIFUX Lectures: 1 (Introduction to Linux) by Guy Keren, 12 (Linux Startup Process) by Guy Keren.

[2] Linux source code: init/main.c.

[3] LILO Technical Overview, version 21.

[4] Linux Documentation Project (http://www.ldp.org): From-PowerUP-To-Bash-Prompt HOWTO, LILO mini-HOWTO, Linux-Init HOWTO.