1. Linux Introduction

1.3 Linux Evolution and Popular Operating Systems

The definition of the word *Linux* depends on the context in which it is used. Linux means the *kernel* of the system, which is the central controller of everything that happens on the computer (more on this later). People that say their computer "runs Linux" usually refer to the kernel and suite of tools that come with it (called the *distribution*). If you have "Linux experience", you are most likely talking about the programs themselves, though depending on the context, you might be talking about knowing how to fine-tune the kernel. Each of these components will be investigated so that you understand exactly what roles each plays.

Further complicating things is the term *UNIX*. UNIX was originally an operating system developed at AT&T Bell Labs in the 1970's. It was modified and *forked* (that is, people modified it and those modifications served as the basis for other systems) such that at the present time there are many different variants of UNIX. However, UNIX is now both a trademark and a specification, owned by an industry consortium called the Open Group. Only software that has been certified by the Open Group may call itself UNIX. Despite adopting all the requirements of the UNIX specification, Linux has not been certified, so Linux really isn't UNIX! It's just… UNIX-like.

1.3.1 Role of the Kernel

The kernel of the operating system is like an air traffic controller at an airport. The kernel dictates which program gets which pieces of memory, it starts and kills programs, and it handles displaying text on a monitor. When an application needs to write to disk, it must ask the operating system to do it. If two applications ask for the same resource, the kernel decides who gets it, and in some cases, kills off one of the applications in order to save the rest of the system.

The kernel also handles switching of applications. A computer will have a small number of CPUs and a finite amount of memory. The kernel takes care of unloading one task and loading a new task if there are more tasks than CPUs. When the current task has run a sufficient amount of time, the CPU pauses the task so that another may run. This is called *pre-emptive multitasking*. Multitasking means that the computer is doing several tasks at once, and pre-emptive means that the kernel is deciding when to switch focus between tasks. With the tasks rapidly switching, it appears that the computer is doing many things at once.

Each application may think it has a large block of memory on the system, but it is the kernel that maintains this illusion, remapping smaller blocks of memory, sharing blocks of memory with other applications, or even swapping out blocks that haven't been touched to disk.

When the computer starts up it loads a small piece of code called a *boot loader*. The boot loader's job is to load the kernel and get it started. If you are more familiar with operating systems such as Microsoft Windows or Apple's OS X, you probably never see the boot loader, but in the UNIX world it's usually visible so that you can tweak the way your computer boots.

The boot loader loads the Linux kernel, and then transfers control. Linux then continues with running the programs necessary to make the computer useful, such as connecting to the network or starting a web server.

1.3.2 Applications

Like an air traffic controller, the kernel is not useful without something to control. If the kernel is the tower, the applications are the airplanes. Applications make requests to the kernel and receive resources, such as memory, CPU, and disk, in return. The kernel also abstracts the complicated details away from the application. The application doesn't know if a block of disk is on a solid-state drive from manufacturer A, a spinning metal hard drive from manufacturer B, or even a network file share. Applications just follow the kernel's *Application Programming Interface (API)* and in return don't have to worry about the implementation details.

When we, as users, think of applications, we tend to think of word processors, web browsers, and email clients. The kernel doesn't care if it is running something that's user facing, a network service that talks to a remote computer, or an internal task. So, from this we get an abstraction called a *process*. A process is just one task that is loaded and tracked by the kernel. An application may even need multiple processes to function, so the kernel takes care of running the processes, starting and stopping them as requested, and handing out system resources.

1.3.3 Role of Open Source

Linux started out in 1991 as a hobby project by Linus Torvalds. He made the source freely available and others joined in to shape this fledgling operating system. His was not the first system to be developed by a group, but since it was a built-from-scratch project, early adopters had the ability to influence the project's direction and to make sure mistakes from other UNIX-es were not repeated.

Software projects take the form of *source code*, which is a human readable set of computer instructions. The source code may be written in any of hundreds of different languages, Linux just happens to be written in C, which is a language that shares history with the original UNIX.

Source code is not understood directly by the computer, so it must be compiled into machine instructions by a *compiler*. The compiler gathers all of the source files and generates something that can be run on the computer, such as the Linux kernel.

Historically, most software has been issued under a *closed-source license*, meaning that you get the right to use the machine code, but cannot see the source code. Often the license specifically says that you will not attempt to reverse engineer the machine code back to source code to figure out what it does!

Open source takes a source-centric view of software. The open source philosophy is that you have a right to obtain the software, and to modify it for your own use. Linux adopted this philosophy to great success. People took the source, made changes, and shared them back with the rest of the group.

Alongside this, was the *GNU project* (GNU's, not UNIX). While GNU was building their own operating system, they were far more effective at building the tools that go along with a UNIX operating system, such as the compilers and user interfaces. The source was all freely available, so Linux was able to target their tools and provide a complete system. As such, most of the tools that are part of the Linux system come from these GNU tools.

There are many different variants on open source, and those will be examined in a later chapter. All agree that you should have access to the source code, but they differ in how you can, or in some cases, must, redistribute changes.

1.3.4 Linux Distributions

Take Linux and the GNU tools, add some more user facing applications like an email client, and you have a full Linux system. People started bundling all this software into a *distribution* almost as soon as Linux became usable. The distribution takes care of setting up the storage, installing the kernel, and installing the rest of the software. The full featured distributions also include tools to manage the system and a *package manager* to help you add and remove software after the installation is complete.

Like UNIX, there are many different flavors of distributions. These days, there are distributions that focus on running servers, desktops, or even industry specific tools like electronics design or statistical computing. The major players in the market can be traced back to either **Red Hat** or **Debian**. The most visible difference is the package manager, though you will find other differences on everything from file locations to political philosophies.

Red Hat started out as a simple distribution that introduced the Red Hat Package Manager (RPM). The developer eventually formed a company around it which tried to commercialize a Linux desktop for business. Over time, Red Hat started to focus more on the server applications such as web and file serving, and released Red Hat Enterprise Linux, which was a paid service on a long *release cycle*. The release cycle dictates how often software is upgraded. A business may value stability and want long release cycles, a hobbyist or a startup may want the latest software and opt for a shorter release cycle. To satisfy the latter group, Red Hat sponsors the **Fedora Project** which makes a personal desktop comprising the latest software, but still built on the same foundations as the enterprise version.

Because everything in Red Hat Enterprise Linux is open source, a project called **CentOS** came to be, that recompiled all the RHEL packages and gave them away for free. CentOS and others like it (such as **Scientific Linux**) are largely compatible with RHEL and integrate some newer software, but do not offer the paid support that Red Hat does.

Debian is more of a community effort, and as such, also promotes the use of open source software and adherence to standards. Debian came up with its own package management system based on the .deb file format. While Red Hat leaves non Intel and AMD platform support to derivative projects, Debian supports many of these platforms directly.

Ubuntu is the most popular Debian derived distribution. It is the creation of **Canonical**, a company that was made to further the growth of Ubuntu and make money by providing support.

1.3.5 Hardware Platforms

Linux started out as something that would only run on a computer like Linus': a 386 with a specific hard drive controller. The range of support grew, as people built support for other hardware. Eventually, Linux started supporting other chips, including hardware that was made to run competitive operating systems!

The types of hardware grew from the humble Intel chip up to supercomputers. Later, smaller-size, Linux supported, chips were developed to fit in consumer devices, called embedded devices. The support for Linux became ubiquitous such that it is often easier to build hardware to support Linux and then use Linux as a springboard for your custom software, than it is to build the custom hardware and software from scratch.

Eventually, cellular phones and tablets started running Linux. A company, later bought by Google, came up with the Android platform which is a bundle of Linux and the software necessary to run a phone or tablet. This means that the effort to get a phone to market is significantly less, and companies can spend their time innovating on the user facing software rather than reinventing the wheel each time. Android is now one of the market leaders in the space.

Aside from phones and tablets, Linux can be found in many consumer devices. Wireless routers often run Linux because it has a rich set of network features. The TiVo is a consumer digital video recorder built on Linux. Even though these devices have Linux at the core, the end users don't have to know. The custom software interacts with the user and Linux provides the stable platform.

1.4 Choosing an Operating System

You have learned that Linux is a UNIX-like operating system, which means that it has not undergone formal certification and therefore can't use the official UNIX trademark. There are many other alternatives; some which are also UNIX-like and some that are certified as UNIX. There are also non-Unix operating systems such as Microsoft Windows.

The most important question to ask when determining the configuration of a machine is "what will this machine do?" If you need to run specialized software that only runs on Oracle Solaris, then that's what you'll need. If you need to be able to read and write Microsoft Office documents, then you'll either need Windows or something capable of running LibreOffice or OpenOffice.

1.4.1 Decision Points

The first thing you need to decide is the machine's role. Will you be sitting at the console running productivity applications or web browsing? If so, you have a desktop. Will the machine be used as a Web server or otherwise sitting in a server rack somewhere? You're looking at a server.

Servers usually sit in a rack and share a keyboard and monitor with many other computers, since console access is only used to set up and troubleshoot the server. The server will run in non-graphical mode, which frees up resources for the real purpose of the computer. A desktop will primarily run a GUI.

Next, determine the functions of the machine. Is there specific software it needs to run, or specific functions it needs to do? Do you need to be able to manage hundreds or thousands of these machines at the same time? What is the skill set of the team managing the computer and software?

You must also determine the lifetime and risk tolerance of the server. Operating systems and software upgrades come on a periodic basis, called the *release cycle*. Software vendors will only support older versions of software for a certain period of time before not offering any updates, which is called the *maintenance cycle* (or life cycle). For example, major Fedora Linux releases come out approximately every 6 months. Versions are considered *End of Life (EOL)* after 2 major versions plus one month, so you have between 7 and 13 months after installing Fedora before you need to upgrade. Contrast this with the commercial server variant, Red Hat Enterprise Linux, and you can go up to 13 years before needing to upgrade.

The maintenance and release cycles are important because in an enterprise server environment it is time consuming, and therefore rare, to do a major upgrade on a server. Instead, the server itself is replaced when there are major upgrades or replacements to the application that necessitate an operating system upgrade. Similarly, a slow release cycle is important because applications often target the current version of the operating system and you want to avoid the overhead of upgrading servers and operating systems constantly to keep up. There is a fair amount of work involved in upgrading a server, and the server role often has many customizations made that are difficult to port to a new server. This necessitates much more testing than if only the application were upgraded.

If you are doing software development or traditional desktop work, you often want the latest software. Newer software has improvements in both functionality and appearance, which contributes to more enjoyment from the use of the computer. A desktop often stores its work on a remote server, so the desktop can be wiped clean and the newer operating system put on with little interruption.

Individual software releases can be characterized as *beta* or *stable*. One of the great things about being an open source developer is that you can release your new software and quickly get feedback from users. If a software release is in a state that it has many new features that have not been rigorously tested, it is typically referred to as beta. After those features are being tested in the field, the software moves to a stable point. If you need the latest features, then you are looking for a distribution that has a quick release cycle and makes it easy to use beta software. On the server side, you want stable software unless those new features are necessary and you don't mind running code that has not been thoroughly tested.

Another loosely related concept is *backward compatibility*. This refers to the ability for a later operating system to be compatible with software made for earlier versions. This is usually a concern if you need to upgrade your operating system, but aren't in a position to upgrade your application software.

Of course, cost is always a factor. Linux itself might be free, but you may need to pay for support, depending on which options you choose. Microsoft has server license costs and may have additional support costs over the lifetime of the server. Your chosen operating system might only run on a particular selection of hardware, which further affects the cost.

1.4.2 Microsoft Windows

The Microsoft world splits the operating systems according to the machine's purpose: desktop or server? The Windows desktop edition has undergone various naming schemes with the current version (as of this writing) being simply **Windows 10**. New versions of the desktop come out every 3-5 years and tend to be supported for many years. Backward compatibility is also a priority for Microsoft, even going so far as to bundle virtual machine technology so that users can run older software.

In the server realm, there is Windows Server, currently (at this writing) at version 2012 to denote the release date. The server runs a GUI, but largely as a competitive response to Linux, has made amazing strides in command line scripting abilities through PowerShell. You can also make the server look like a desktop with the optional Desktop Experience package.

1.4.3 Apple OS X

Apple makes the OS X operating system, which has undergone UNIX certification. OS X is partially based on software from the FreeBSD project.

At the moment, OS X is primarily a desktop operating system but there are optional packages that help with management of network services that allow many OS X desktops to collaborate, such as to share files or have a network login.

OS X on the desktop is usually a personal decision as many find the system easier to use. The growing popularity of OS X has ensured healthy support from software vendors. OS X is also quite popular in the creative industries such as video production. This is one area where the applications drive the operating system decision, and therefore the hardware choice since OS X runs on Apple hardware.

1.4.4 BSD

There are several open source BSD projects, such as OpenBSD, FreeBSD, and NetBSD. These are alternatives to Linux in many respects as they use a lot of common software. The BSDs usually find themselves in the server role, though also use GNOME and KDE for desktop roles.

1.4.5 Other Commercial UNIXes

Some of the more popular commercial UNIX-es are:

- Oracle Solaris
- IBM AIX
- HP-UX

Each of these runs on hardware from their respective creators. The hardware is usually large and powerful, offering such features as hot-swap CPU and memory, or integration with legacy mainframe systems also offered by the vendor.

Unless the software requires the specific hardware or the needs of the application require some of the redundancy built into the hardware, most people tend to choose these options because they are already users of the company's products. For example, IBM AIX runs on a wide variety of IBM hardware and can share hardware with mainframes. Thus, you find AIX in companies that already have a large IBM footprint, or that make use of IBM software like WebSphere.

1.4.6 Linux

One aspect where Linux is much different than the alternatives is that after you've chosen Linux you still have to choose a distribution. Recall from Topic 1 that the distribution packages the Linux kernel, utilities, and management tools into an installable package and provides a way to install and update packages after the initial installation.

If you choose OS X, Windows, or even OpenBSD, that's what you get. With Linux you have multiple options, from commercial offerings for the server or desktop, to custom distributions made to turn an old computer into a network firewall.

Often application vendors will choose a subset of distributions to support. Different distributions have different versions of key libraries and it is difficult for a company to support all these different versions.

You may also limit your decisions to distributions that offer commercial support. This is common in larger companies where paying for another tier of support is better than risking extensive outages.

1.4.7 Android

Android, sponsored by Google, is the world's most popular Linux distribution. It is fundamentally different from its counterparts. Linux is a kernel, and many of the commands that will be covered in this course are actually part of the **GNU** (GNU's Not Unix) package. That is why some people insist on using the term GNU/Linux instead of Linux alone.

Android uses the **Dalvik** virtual machine with Linux, providing a robust platform for mobile devices such as phones and tablets. However, lacking the traditional packages that are often distributed with Linux (such as GNU and Xorg), Android is generally incompatible with desktop Linux distributions.

This incompatibility means that a RedHat or Ubuntu user can not download software from the Google Play store. Likewise, a terminal emulator in Android lacks many of the commands of its Linux counterparts. It is possible, however, to use BusyBox with Android to enable most commands to work.