Oliver Garraux

5/5/2009

CIT 286

Linux History and Architecture

History

In 1991, Linus Torvalds began work on a clone of MINIX, a version of UNIX that was frequently used by students in the early nineties (Jones). Originally, Torvalds called the new operating system kernel FreakX, however a friend who helped host a copy of it convinced Linus Torvalds to name it Linux, combining his first name and the “X” that is traditionally at the end of UNIX related operating system (Torvalds and Diamond). Originally, it was just a hobby – Torvalds was using it as a way to learn more about the then-new Intel i386 architecture.

Torvalds licensed Linux under an open source license, the GNU General Public License (GPL), which allows everyone to access and modify the Linux source code. It requires that any changes you make to the software must also be licensed under the GPL (assuming that you are distributing your modified version of the software). During the early nineties, there was some skepticism over Linux, and if it was suitable for business use. By the late nineties, that question was definitively answered when major vendors like Oracle and IBM began releasing Linux versions of their enterprise database software. Throughout the 2000’s, Linux’s use has grown, especially in embedded systems. The flexibility that Linux provides (because it is so modular, and open for anyone to modify) makes it ideal for embedded environments. Linux has grown at the other extreme as well, with many of the newest super-computers running on a version of Linux (Torvalds and Diamond).

Architecture

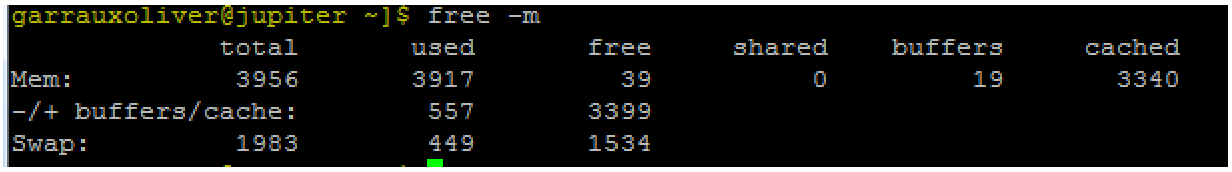
Linux itself technically is not an entire operating system; it is an operating system **kernel**. The kernel, or core of the operating system, handles memory and process management, in addition to all interactions with hardware. Linux is not an operating system **shell**, however it is almost always accompanied by the open source GNU utilities. Some call the combination of the Linux kernel and GNU shell GNU/Linux. Most people use Linux by using distributions, which are prepackaged collections of Linux related software that are easy to install and update. Presently, some of the most common Linux distributions are Red Hat Enterprise Linux/Fedora and Ubuntu.

Memory Management

The Linux kernel uses virtual memory. Each process in Linux has a separate virtual address space for memory; however, the actual memory addresses that the programs running in userland see are the same for all processes. So the applications themselves do not know the real memory address of the memory they use. Instead, the Linux kernel translates their virtual memory addresses to real memory addresses that the kernel then uses to access the data stored in memory. One of the advantages of this approach is that Linux and the applications that run on it are independent of the memory-addressing scheme used by the hardware it is running on (Bovet and Cesati 35-36).

Linux also implements paging by splitting the memory space up into pages. Typically, these pages are 4KB in length. Linux uses demand paging which means that the all of the virtual memory that a process uses may not actually be stored in memory. Parts of the virtual memory can be stored on disk and swapped in to physical memory when needed. This makes it easier to run large applications, and makes the operating system more reliable. Otherwise, if Linux did not support a paging system that allowed virtual memory to be stored on disk, the system or applications might crash when it runs out of memory. This way, there will just be a performance hit.

When a child process is created on Linux, it will share the same memory as the parent process that created it. The memory will only be copied to a separate place in memory when one of the processes needs to change their copy of the memory (by writing to it). This is called “Copy on Write”, and can save memory by letting processes share memory when the contents of the memory are the same. Another interesting memory feature that Linux has is that it will use free memory to cache or buffer data that is being read from or written to the disk. The philosophy behind using the memory as a cache is that empty memory is memory that could be doing something useful. Linux can quickly clear the memory that is being used as a cache, so there is little downside to using memory as a cache; it can improve I/O speeds by buffering writes or keeping frequently read data in memory. On many Linux systems, it will appear that the system is almost out of memory because Linux is using a great deal of the memory for caching. The screenshot below demonstrates an example of this. While the "free" command shows only 39 MB free out of a total of 4 GB, a closer inspection reveals that over 3 GB of the memory that is being used is being used for caching.



```
garrauxoliver@jupiter ~]$ free -m
             total       used       free     shared    buffers     cached
Mem:          3956       3917         39          0         19       3340
-/+ buffers/cache:        557       3399
Swap:         1983        449       1534
```

Both Copy on Write and caching must be taken into consideration when looking at memory usage on Linux. Because of Copy on Write, the sum of the memory that each process is using will be more than the amount of memory that is actually being used on the system, because some of it is shared between multiple processes.

On Linux (as with most Unix-based systems), the pages that are swapped out to the disk are stored in a separate partition known as swap. Another feature in Linux related to paging is the support for Physical Address Extensions (PAE), which allows a 32-bit system to address more than 4 GB of memory. Linux uses multiple levels of paging to accomplish this.

Linux technically does use memory segmentation as well as paging; however, its use of segmentation is extremely limited. Linux uses only four segments: one each for kernel code, kernel data, user code, and user data. Segmentation is not as portable as paging (Freie Universität Berlin); considering that portability is one of the design goals of the Linux kernel, it is no surprise that segmentation is not a major feature of the Linux memory manager.

Process Management

Any Linux process must be running in either kernel mode or user mode. Kernel mode processes include the core of the kernel itself, along with all of the kernel modules/drivers that are running.   Kernel mode processes run are privileged and have direct access to the memory and hardware. Processes running in user mode interact with the kernel by using system calls. There is a wide variety of system calls, some like fork() and exec() are related to process management. Others, like malloc() relate to memory management (Bovet and Cesati 398).

The Linux kernel is a reentrant, meaning that it is capable of preemptable multitasking. The execution of processes can be interrupted by the kernel. When this happens, a context switch occurs and the details regarding the process's state are saved in the process descriptor (Bovet and Cesati 105).

Scheduling

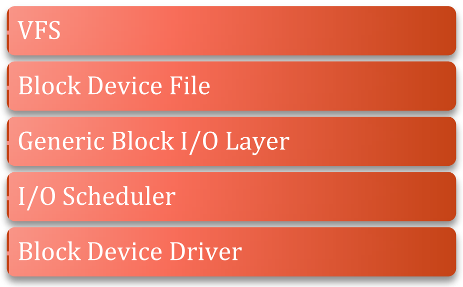
Because the Linux kernel is preemptable, the processor must control the execution of processes and stop them from taking up more time than they should. The kernel splits time up into slices to regulate a process's activities. There are two typical types of processes in Linux: conventional processes and real time processes. Real time processes are ones that demand more immediate attention and will execute before any conventional processes have a chance to run. Typically, most real time processes are ones that interact with hardware or are related to the kernel. The kernel will first run the real time processes with the highest priority and work its way down to the lower priority real time processes.

To schedule conventional processes, the kernel uses two different sets of priorities. Static priorities are assigned to the process either when it is started, or manually set by the system administrator by using the renice command. The lower the nice value is, the higher priority the process has when executing. To give the process a higher priority Linux increases the length of the time slice that the process gets. The other type of priorities are dynamic priorities and they are used automatically to prevent one process from monopolizing the CPU time. The dynamic priority of a process varies depending on how much the process has been sleeping recently. If a process has had to sleep for a long time, then its dynamic priority may be increased to give it a better chance of running.

Recently, in 2007 a new scheduler was introduced into the Linux kernel version 2.6.23. Called the Completely Fair Scheduler, it uses what Ingo Molnar, its creator, calls a “timeline” instead of time slices (Molnar). The new algorithm appears to operate with the same general principles that the older scheduler does.

Device and File Management

Linux has two basic kinds of devices: character devices and block devices. Character devices are much simpler, and usually only allow sequential I/O. Examples of typical character devices are a text terminal, tape drive, or serial interface (Bovet and Cesati 552). Block devices are much more complicated, so Linux relies on a layered system. The VFS (Virtual File System) layer that sits at the top is part of the Linux File Manager. All file systems eventually point back to a block device. Then a generic abstracted block I/O layer handles the I/O request. The Generic Block I/O layer is not aware of what type of hardware the block device is, it just forwards the request down to the lower layers of the kernel that handling the scheduling and actually interfacing with the hardware (Bovet and Cesati 560).



As previously mentioned, the Linux VFS is part of the Linux File Manager. It allows programs to work with files regardless of what file system is being used. The read or write method that an application call is pointed to depends on what the file system is. The application interfaces the same for any file system though. This is another great example of the modularity of Linux. Because Linux is so modular, a change in one part of the system has a minimal impact on the other pieces of the system, making it more flexible (Bovet and Cesati 457).

Typically, Linux uses the ext3 file system, which is a modern journaling file system that supports transactions. Other file systems that are commonly used with Linux include ReiserFS, ext2 (an older version of ext3 that does not support journaling), XFS, and JFS. Linux can also read most file systems from other operating system as well. Two new file systems are currently being developed just for Linux. The first of these is ext4, a further improved version of ext3. Ext4 is currently included in some bleeding-edge Linux distributions. Much of its improvements relate to journaling; it also allows you to have a larger block size than ext3 does (Corbet). The second file system is called btrfs, and is not based on any of the previous ext2/3/4 file systems. Btrfs is much more advanced than previous Linux file systems and includes modern snapshot technologies as well as the ability to check the filesystem without unmounting it. It is commonly seen as the Linux community's response to Sun's new ZFS available for Solaris (Layton).

Disadvantages of Linux

While Linux is extremely popular in technical circles is has a negligible desktop prescence when compared to Microsoft Windows. This means that many of the common desktop applications that people are used to on Windows (like Microsoft Office and Adobe Creative Suite) are not currently available for Linux. For server software, the situation is much better. A great deal of server software is available for Linux. Accordingly, Linux is much more common in server environments than it is on desktops.

Another disadvantage of Linux, from a technical perspective, is the graphical interface. Linux uses X.org as the basis for its graphical user interface (if you choose to install a graphical interface). X.org is based off the same graphical system that was used by UNIX in the eighties, so there is a lot of archaic code and complexity that is no longer needed by modern systems. Usually, software can be layered on top of X to solve some of these problems, but this still means that there is additional complexity and overhead when compared to Windows and Mac OS X GUI's.

One final problem that may be seen as a disadvantage for some people is that Linux is not in and of itself a single, cohesive, integrated package. Even Linux distributions are just collections of software written by different people that have been put together. Sometimes this can make it more difficult to learn and use because there are more components that you have to be aware of. However, this can also be seen as an advantage – it is easier to add a piece of software on to Linux, or modify part of Linux.

Advantages

The biggest advantage of using Linux is the fact that it is open source. This means that anyone can look at the code to see how it works, inspect it for security vulnerabilities, or make their own changes. Additionally, Linux was designed to be highly portable, so it can run on an extremely diverse set of hardware – from cell phones to mainframes and super-computers. One final advantage of Linux is its modularity. In some sense, this could be the reason that it is so portable. Because Linux is made of many different components, you can easily change one component without necessarily having to change everything else. For example, you can easily add a file system driver to Linux (even without having to make a kernel module in some cases), without having to update all of the applications running on Linux, or the core of the Linux kernel.

Bibliography

Bovet, Daniel P. and Marco Cesati. Understanding the Linux Kernel. 3rd Edition. Sebastopol, CA: O'Reilly, 2006.

Corbet, Jonathan. "A better ext4 filesystem for Linux." 29 January 2008. LinuxWorld. 1 May 2009 <http://www.linuxworld.com/news/2008/012908-kernel.html>.

Freie Universität Berlin. "Freie Universität Berlin Lectures." 12 June 2001. Linux Memory Management. 3 May 2009 <http://www.inf.fu-berlin.de/lehre/SS01/OS/Lectures/Lecture14.pdf>.

Jones, M. Tim. "IBM DeveloperWorks." 06 June 2007. Anatomy of the Linux kernel. 29 April 2009 <http://www.ibm.com/developerworks/linux/library/l-linux-kernel/>.

Layton, Jeff. "Linux Don't Need No Stinkin' ZFS: BTRFS Intro & Benchmarks." 21 April 2009. Linux Magazine. 30 April 2009 <http://www.linux-mag.com/id/7308>.

Molnar, Ingo. "Kernel Trap." 18 April 2007. Linux: The Completely Fair Scheduler. 1 May 2009 <http://kerneltrap.org/node/8059>.

Torvalds, Linus and David Diamond. Just for Fun. HarperCollins, 2001.