Linux

Operating and File Systems

Christy Reuter
Jammie Dugas

First Edition, 2012

ISBN 978-81-323-1286-4

Published by:
College Publishing House
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1 - Introduction to Linux

Chapter 2 - Linux Kernel

Chapter 3 - History of Linux

Chapter 4 - Linux Adoption

Chapter 5 - Linux Distribution

Chapter 6 - Btrfs

Chapter 7 - Ext2

Chapter 8 - Ext3

Chapter 9 - Ext4

Chapter 10 - FAT Filesystem & Linux

Chapter 11 - Global File System

Chapter 12 - JFFS and JFFS2

Chapter 13 - LogFS and Lustre (File System)

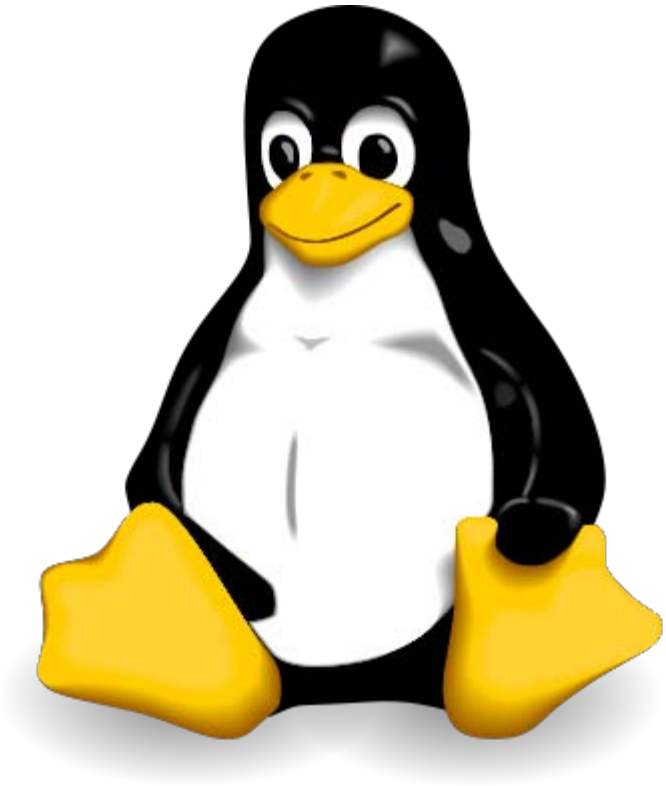
Chapter 14 - NILFS and ReiserFS

Chapter 15 - XFS

Chapter 1

Introduction to Linux

Linux



Company / developer	GNU Project, Linus Torvalds and many others
Programmed in	Assembly language, C
OS family	Unix-like
Working state	Current
Source model	Free and open source software
Latest stable	2.6.36.2 (December 9, 2010; 21 days ago) [+/−]

release	
Latest unstable release	2.6.37-rc7 (December 21, 2010; 9 days ago) [+/−]
Marketing target	Desktops, servers, embedded devices
Available language(s)	Multi-lingual
Available programming languages(s)	Many
Supported platforms	IA-32, MIPS, x86-64, SPARC, DEC Alpha, Itanium, PowerPC, ARM, m68k, PA-RISC, s390, SuperH, M32R and more
Kernel type	Monolithic
Userland	GNU and others
Default user interface	Graphical (X Window System) and command-line interface
License	Various including GNU General Public License, BSD License, Apache License, MIT License, and others

Linux refers to the family of Unix-like computer operating systems using the Linux kernel. Linux can be installed on a wide variety of computer hardware, ranging from mobile phones, tablet computers and video game consoles, to mainframes and supercomputers. Linux is a leading server operating system, and runs the 10 fastest supercomputers in the world.

The development of Linux is one of the most prominent examples of free and open source software collaboration; typically all the underlying source code can be used, freely modified, and redistributed, both commercially and non-commercially, by anyone under licenses such as the GNU General Public License. Typically Linux is packaged in a format known as a *Linux distribution* for desktop and server use. Some popular mainstream Linux distributions include Debian (and its derivatives such as Ubuntu), Fedora and openSUSE. Linux distributions include the Linux kernel and supporting utilities and libraries to fulfill the distribution's intended use.

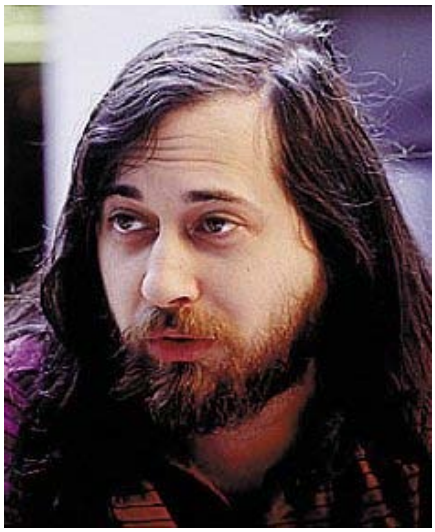
A distribution oriented toward desktop use may include the X Window System, the GNOME and KDE Plasma desktop environments, and the Apache HTTP Server. Other distributions may omit some of this software or substitute others, such as including LXDE instead of a more resource intensive desktop. Because Linux is freely redistributable, it is possible for anyone to create a distribution for any intended use. Commonly used applications with desktop Linux systems include the Mozilla Firefox web-browser, the OpenOffice.org office application suite and the GIMP image editor.

The name "Linux" comes from the Linux kernel, originally written in 1991 by Linus Torvalds. The main supporting user space system tools and libraries from the GNU Project (announced in 1983 by Richard Stallman) are the basis for the Free Software Foundation's preferred name *GNU/Linux*.

History

Unix

The Unix operating system was conceived and implemented in 1969 at AT&T's Bell Laboratories in the United States by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna. It was first released in 1971 and was initially entirely written in assembly language, a common practice at the time. Later, in a key pioneering approach in 1973, Unix was re-written in the programming language C by Dennis Ritchie (with exceptions to the kernel and I/O). The availability of an operating system written in a high-level language allowed easier portability to different computer platforms. With a legal glitch forcing AT&T to license the operating system's source code, Unix quickly grew and became widely adopted by academic institutions and businesses.



Richard Stallman, founder of the GNU project

GNU

The GNU Project, started in 1983 by Richard Stallman, had the goal of creating a "complete Unix-compatible software system" composed entirely of free software. Work began in 1984. Later, in 1985, Stallman started the Free Software Foundation and wrote the GNU General Public License (GNU GPL) in 1989. By the early 1990s, many of the programs required in an operating system (such as libraries, compilers, text editors, a Unix shell, and a windowing system) were completed, although low-level elements such as device drivers, daemons, and the kernel were stalled and incomplete. Linus Torvalds has said that if the GNU kernel had been available at the time (1991), he would not have decided to write his own.

MINIX

Andrew S. Tanenbaum (left), author of the MINIX operating system and Linus Torvalds (right), principal author of the Linux kernel

MINIX is an inexpensive minimal Unix-like operating system, designed for education in computer science, written by Andrew S. Tanenbaum. Starting with version 3, MINIX is free and redesigned also for “serious” use.

In 1991 while attending the University of Helsinki, Torvalds, curious about the operating systems and frustrated by the licensing of MINIX limiting it to educational use only (which prevented any commercial use), began to work on his own operating system which eventually became the Linux kernel.

Torvalds began the development of the Linux kernel on MINIX, and applications written for MINIX were also used on Linux. Later Linux matured and it became possible for Linux to be developed under itself. Also GNU applications replaced all MINIX ones because, with code from the GNU system freely available, it was advantageous if this could be used with the fledgling operating system. Code licensed under the GNU GPL

can be used in other projects, so long as they also are released under the same or a compatible license. In order to make the Linux available for commercial use, Torvalds initiated a switch from his original license (which prohibited commercial redistribution) to the GNU GPL. Developers worked to integrate GNU components with Linux to make a fully functional and free operating system.

Current development

Torvalds continues to direct the development of the kernel. Stallman heads the Free Software Foundation, which in turn supports the GNU components. Finally, individuals and corporations develop third-party non-GNU components. These third-party components comprise a vast body of work and may include both kernel modules and user applications and libraries. Linux vendors and communities combine and distribute the kernel, GNU components, and non-GNU components, with additional package management software in the form of Linux distributions.

Design

A Linux-based system is a modular Unix-like operating system. It derives much of its basic design from principles established in Unix during the 1970s and 1980s. Such a system uses a monolithic kernel, the Linux kernel, which handles process control, networking, and peripheral and file system access. Device drivers are either integrated directly with the kernel or added as modules loaded while the system is running.

Separate projects that interface with the kernel provide much of the system's higher-level functionality. The GNU userland is an important part of most Linux-based systems, providing the most common implementation of the C library, a popular shell, and many of the common Unix tools which carry out many basic operating system tasks. The graphical user interface (or GUI) used by most Linux systems is built on top of an implementation of the X Window System.

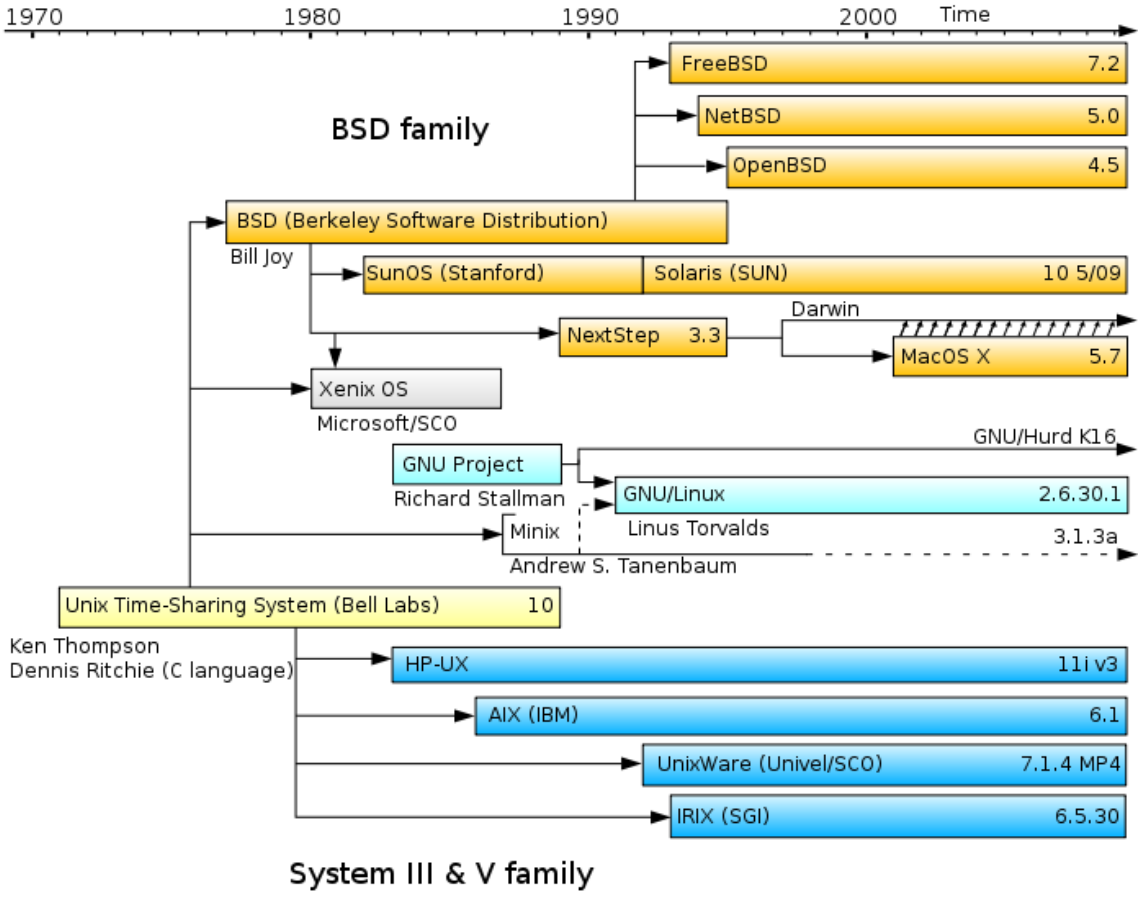
User interface

Users operate a Linux-based system through a command line interface (CLI), a graphical user interface (GUI), or through controls attached to the associated hardware, which is common for embedded systems. For desktop systems, the default mode is usually a graphical user interface, by which the CLI is available through terminal emulator windows or on a separate virtual console. Most low-level Linux components, including the GNU userland, use the CLI exclusively. The CLI is particularly suited for automation of repetitive or delayed tasks, and provides very simple inter-process communication. A graphical terminal emulator program is often used to access the CLI from a Linux desktop. A Linux system typically implements a CLI by a shell, which is also the traditional way of interacting with a Unix system. A Linux distribution specialized for servers may use the CLI as its only interface.

On desktop systems, the most popular user interfaces are the extensive desktop environments KDE Plasma Desktop, GNOME, and Xfce, though a variety of additional user interfaces exist. Most popular user interfaces are based on the X Window System, often simply called "X". It provides network transparency and permits a graphical application running on one system to be displayed on another where a user may interact with the application.

Other GUIs may be classified as simple X window managers, such as FVWM, Enlightenment, and Window Maker, which provide a minimalist functionality with respect to the desktop environments. A window manager provides a means to control the placement and appearance of individual application windows, and interacts with the X Window System. The desktop environments include window managers as part of their standard installatations (Metacity for GNOME, Kwin for KDE, Xfwm for XFCE as of 2010) although users may choose to use a different window manager if preferred.

Development



A summarized history of Unix-like operating systems showing Linux's origins. Of note, Linux shares similar architectural designs and concepts (as part of the POSIX standard) but does not share non-free source code with the original Unix or MINIX.

The primary difference between Linux and many other popular contemporary operating systems is that the Linux kernel and other components are free and open source software. Linux is not the only such operating system, although it is by far the most widely used. Some free and open source software licenses are based on the principle of copyleft, a kind of reciprocity: any work derived from a copyleft piece of software must also be copyleft itself. The most common free software license, the GNU GPL, is a form of copyleft, and is used for the Linux kernel and many of the components from the GNU project.

Linux based distributions are intended by developers for interoperability with other operating systems and established computing standards. Linux systems adhere to POSIX, SUS, ISO, and ANSI standards where possible, although to date only one Linux distribution has been POSIX.1 certified, Linux-FT.

Free software projects, although developed in a collaborative fashion, are often produced independently of each other. The fact that the software licenses explicitly permit redistribution, however, provides a basis for larger scale projects that collect the software produced by stand-alone projects and make it available all at once in the form of a Linux distribution.

A Linux distribution, commonly called a "distro", is a project that manages a remote collection of system software and application software packages available for download and installation through a network connection. This allows the user to adapt the operating system to his/her specific needs. Distributions are maintained by individuals, loose-knit teams, volunteer organizations, and commercial entities. A distribution is responsible for the default configuration of the installed Linux kernel, general system security, and more generally integration of the different software packages into a coherent whole. Distributions typically use a package manager such as dpkg, Synaptic, YAST, or Portage to install, remove and update all of a system's software from one central location.

Community

A distribution is largely driven by its developer and user communities. Some vendors develop and fund their distributions on a volunteer basis, Debian being a well-known example. Others maintain a community version of their commercial distributions, as Red Hat does with Fedora.

In many cities and regions, local associations known as Linux Users Groups (LUGs) seek to promote their preferred distribution and by extension free software. They hold meetings and provide free demonstrations, training, technical support, and operating system installation to new users. Many Internet communities also provide support to Linux users and developers. Most distributions and free software / open source projects have IRC chatrooms or newsgroups. Online forums are another means for support, with notable examples being LinuxQuestions.org and the Gentoo forums. Linux distributions host mailing lists; commonly there will be a specific topic such as usage or development for a given list.

There are several technology websites with a Linux focus. Print magazines on Linux often include cover disks including software or even complete Linux distributions.

Although Linux distributions are generally available without charge, several large corporations sell, support, and contribute to the development of the components of the system and of free software. An analysis of the Linux kernel showed 75 percent of the code from December 2008 to January 2010 was developed by programmers working for corporations, leaving about 18 percent to the traditional, open source community. Some of the major corporations that contribute include Dell, IBM, HP, Oracle, Sun Microsystems, Novell, and Nokia. A number of corporations, notably Red Hat, have built their entire business around Linux distributions.

The free software licenses, on which the various software packages of a distribution built on the Linux kernel are based, explicitly accommodate and encourage commercialization; the relationship between a Linux distribution as a whole and individual vendors may be seen as symbiotic. One common business model of commercial suppliers is charging for support, especially for business users. A number of companies also offer a specialized business version of their distribution, which adds proprietary support packages and tools to administer higher numbers of installations or to simplify administrative tasks.

Another business model is to give away the software in order to sell hardware. This used to be the norm in the computer industry, with operating systems such as CP/M, Apple DOS and versions of MacOS prior to 7.5 freely copyable (but not modifiable). As computer hardware standardized throughout the 1980s, it became more difficult for hardware manufacturers to employ this tactic, as their OS would run on any computer that shared the same architecture.

Programming on Linux

Linux distributions support dozens of programming languages. The most common collection of utilities for building both Linux applications and operating system programs is found within the GNU toolchain, which includes the GNU Compiler Collection (GCC) and the GNU build system. Amongst others, GCC provides compilers for Ada, C, C++, Java, and Fortran. Proprietary compilers for Linux include the Intel C++ Compiler, Sun Studio, and IBM XL C/C++ Compiler. BASIC is supported in such forms as Gambas, FreeBASIC, and XBasic.

Most distributions also include support for PHP, Perl, Ruby, Python and other dynamic languages. While not as common, Linux also supports C# (via Mono), Vala, and Scheme. A number of Java Virtual Machines and development kits run on Linux, including the original Sun Microsystems JVM (HotSpot), and IBM's J2SE RE, as well as many open-source projects like Kaffe.

The two main frameworks for developing graphical applications are those of GNOME and KDE. These projects are based on the GTK+ and Qt widget toolkits, respectively, which can also be used independently of the larger framework. Both support a wide

variety of languages. There are a number of Integrated development environments available including Anjuta, Code::Blocks, Eclipse, Geany, ActiveState Komodo, KDevelop, Lazarus, MonoDevelop, NetBeans, Qt Creator and Omnis Studio while the long-established editors Vim and Emacs remain popular.

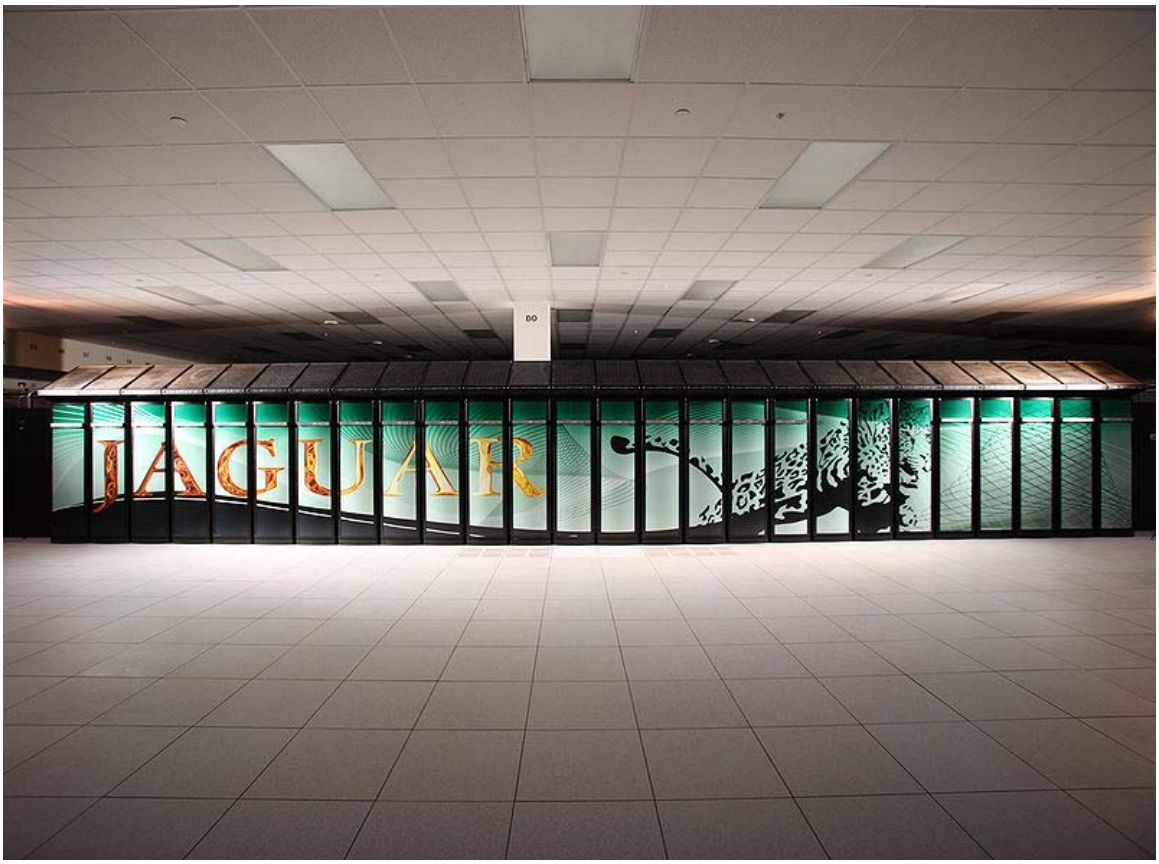
Uses

As well as those designed for general purpose use on desktops and servers, distributions may be specialized for different purposes including: computer architecture support, embedded systems, stability, security, localization to a specific region or language, targeting of specific user groups, support for real-time applications, or commitment to a given desktop environment. Furthermore, some distributions deliberately include only free software. Currently, over three hundred distributions are actively developed, with about a dozen distributions being most popular for general-purpose use.

Linux is a widely ported operating system kernel. The Linux kernel runs on a highly diverse range of computer architectures: in the hand-held ARM-based iPAQ and the mainframe IBM System z9, System z10 in devices ranging from mobile phones to supercomputers. Specialized distributions exist for less mainstream architectures. The ELKS kernel fork can run on Intel 8086 or Intel 80286 16-bit microprocessors, while the μClinux kernel fork may run on systems without a memory management unit. The kernel also runs on architectures that were only ever intended to use a manufacturer-created operating system, such as Macintosh computers (with both PowerPC and Intel processors), PDAs, video game consoles, portable music players, and mobile phones.

There are several industry associations and hardware conferences devoted to maintaining and improving support for diverse hardware under Linux, such as FreedomHEC.

Servers, mainframes and supercomputers



The Oak Ridge National Laboratory's Jaguar supercomputer, until recently the world's fastest supercomputer. It uses the Cray Linux Environment as its operating system.

Servers designed for Linux

Linux distributions have long been used as server operating systems, and have risen to prominence in that area; Netcraft reported in September 2006 that eight of the ten most reliable internet hosting companies ran Linux distributions on their web servers. (since June 2008, Linux distributions represented five of the top ten, FreeBSD three of ten, and Microsoft two of ten; since February 2010, Linux distributions represented six of the top ten, FreeBSD two of ten, and Microsoft one of ten.)

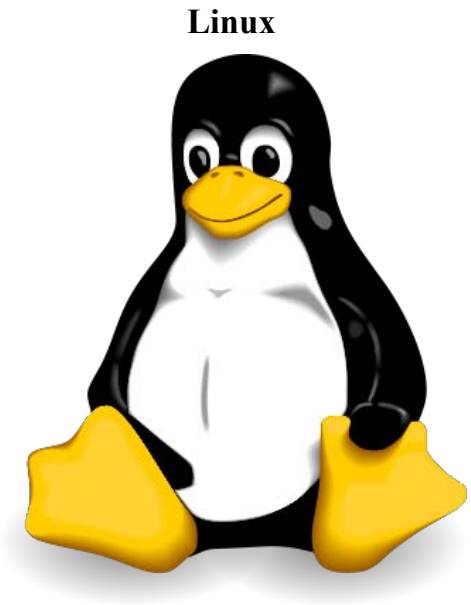
Linux distributions are the cornerstone of the LAMP server-software combination (Linux, Apache, MySQL, Perl/PHP/Python) which has achieved popularity among developers, and which is one of the more common platforms for website hosting.

Linux distributions have become increasingly popular on mainframes in the last decade due to pricing, compared to other mainframe operating systems. In December 2009, computer giant IBM reported that it would predominantly market and sell mainframe-based Enterprise Linux Server.

Linux distributions are also commonly used as operating systems for supercomputers: since June 2010, out of the top 500 systems, 455 (91%) run a Linux distribution. Linux was also selected as the operating system for the world's most powerful supercomputer, IBM's Sequoia which will become operational in 2011.

Chapter 2

Linux Kernel




```
Kernel command line: block2mtd.block2mtd=/dev/hda2,131072,rootfs root=/dev/mtdbl
ock0 rootfstype=jffs2 init=/etc/preinit  noinitrd console=tty0 console=ttyS0,384
00n8 reboot=bios
Found and enabled local APIC!
Enabling fast FPU save and restore... done.
Enabling unmasked SIMD FPU exception support... done.
Initializing CPU#0
PID hash table entries: 32 (order: 5, 128 bytes)
Detected 1991.657 MHz processor.
Console: colour VGA+ 80x25
console [tty0] enabled
console [ttyS0] enabled
Dentry cache hash table entries: 1024 (order: 0, 4096 bytes)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
Memory: 5112k/8128k available (1497k kernel code, 2624k reserved, 597k data, 196
k init, 0k highmem)
virtual kernel memory layout:
    fixmap  : 0xfffb9000 - 0xfffff000   ( 280 kB)
    vmalloc : 0xc1000000 - 0xfffb7000   (1007 MB)
    lowmem  : 0xc0000000 - 0xc07f0000   (   7 MB)
      .init : 0xc0313000 - 0xc0344000   ( 196 kB)
      .data : 0xc027653c - 0xc030bcfc   ( 597 kB)
      .text : 0xc0100000 - 0xc027653c   (1497 kB)
Checking if this processor honours the WP bit even in supervisor mode...Ok.
Calibrating delay using timer specific routine.. 4047.64 BogoMIPS (lpj=20238210)
```

Linux kernel 2.6.25.17 booting

Original author(s)	Linus Torvalds
Developer(s)	Linus Torvalds and thousands of collaborators
Initial release	1991
Stable release	2.6.36.2 (December 9, 2010; 20 days ago) [+/−]
Preview release	2.6.37-rc7 (December 21, 2010; 8 days ago) [+/−]
Written in	C
Operating system	Unix-like
Available in	English
Type	Kernel
License	GNU General Public License version 2 (only) plus variously licensed binary blobs

The **Linux kernel** is an operating system kernel used by the Linux family of Unix-like operating systems. It is one of the most prominent examples of free and open source software.

The Linux kernel is released under the GNU General Public License version 2 (GPLv2), (plus some firmware images with various licenses), and is developed by contributors worldwide. Day-to-day development takes place on the Linux kernel mailing list.

The Linux kernel was initially conceived and created by Finnish computer science student Linus Torvalds in 1991. Linux rapidly accumulated developers and users who adopted code from other free software projects for use with the new operating system. The Linux kernel has received contributions from thousands of programmers. Many Linux distributions have been released based upon the Linux kernel.

History

In April 1991, Linus Torvalds, a 21-year-old student at the University of Helsinki, Finland started working on some simple ideas for an operating system. He started with a task switcher in Intel 80386 assembly language and a terminal driver. On 26 August 1991, Torvalds posted the following to *comp.os.minix*, a newsgroup on Usenet:

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since April, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months [...] Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

[...] It's mostly in C, but most people wouldn't call what I write C. It uses every conceivable feature of the 386 I could find, as it was also a project to teach me about the 386. As already mentioned, it uses a MMU, for both paging (not to disk yet) and segmentation. It's the segmentation that makes it REALLY 386 dependent (every task has a 64Mb segment for code & data - max 64 tasks in 4Gb. Anybody who needs more than 64Mb/task - tough cookies). [...] Some of my "C"-files (specifically mm.c) are almost as much assembler as C. [...] Unlike minix, I also happen to LIKE interrupts, so interrupts are handled without trying to hide the reason behind them.

After that, many people contributed code to the project. Early on, the MINIX community contributed code and ideas to the Linux kernel. At the time, the GNU Project had created many of the components required for a free operating system, but its own kernel, GNU Hurd, was incomplete and unavailable. The BSD operating system had not yet freed itself from legal encumbrances. Despite the limited functionality of the early versions, Linux rapidly accumulated developers and users.

By September 1991, Linux version 0.01 was released, uploading it to the FTP server (ftp.funet.fi) of the Helsinki University of Technology (HUT). It had 10,239 lines of code. In October 1991, Linux version 0.02 was released.

In December 1991, Linux 0.11 was released. This version was the first to be self-hosted - Linux 0.11 could be compiled by a computer running Linux 0.11. When he released version 0.12 in February 1992, Torvalds adopted the GNU General Public License (GPL) over his previous self-drafted license, which had not permitted commercial redistribution.

A newsgroup known as *alt.os.linux* was started, and on 19 January 1992, the first post to alt.os.linux was made. On 31 March 1992, alt.os.linux became *comp.os.linux*.

The X Window System was soon ported to Linux. In March 1992, Linux version 0.95 was the first to be capable of running X. This large version number jump (from 0.1x to 0.9x) was due to a feeling that a version 1.0 with no major missing pieces was imminent. However, this proved to be somewhat overoptimistic, and from 1993 to early 1994, 15 development versions of version 0.99 appeared.

On 14 March 1994, Linux 1.0.0 was released, with 176,250 lines of code. In March 1995, Linux 1.2.0 was released (310,950 lines of code).

Version 2 of Linux, released on 9 June 1996, was followed by additional major versions under the version 2 header:

- 25 January 1999 - Linux 2.2.0 was released (1,800,847 lines of code).

- 18 December 1999 - IBM mainframe patches for 2.2.13 were published, allowing Linux to be used on enterprise-class machines.
- 4 January 2001 - Linux 2.4.0 was released (3,377,902 lines of code).
- 17 December 2003 - Linux 2.6.0 was released (5,929,913 lines of code).
- 20 October 2010 - Linux 2.6.36 was released (13,499,457 lines of code).

In July 2009 Microsoft submitted Hyper-V drivers to the kernel, which improve the performance of virtual Linux guest systems in a Windows hosted environment. Microsoft was forced to submit the code when it was discovered that Microsoft had incorporated a Hyper-V network driver with GPL-licensed components statically linked to closed-source binaries.

Legal aspects

Licensing terms

Initially, Torvalds released Linux under a license which forbade any commercial exploitation. This was soon changed to the GNU General Public License (GPL), as of version 0.12. This license allows distribution and sale of possibly modified and unmodified versions of Linux but requires that all those copies be released under the same license and be accompanied by the complete corresponding source code.

Torvalds has described licensing Linux under the GPL as the "best thing I ever did."

GPL version 3

Currently, Linux is licensed only under version 2 of the GPL, with (unlike much other GPL software) no option to use a later version, and there is some controversy over how easily it could be changed to use later GPL versions such as the new version 3 (and whether this is even desirable). Torvalds himself specifically indicated upon the release of version 2.4.0 that his own code is only under version 2. However, the terms of the GPL state that if no version is specified, then any version may be used, and Alan Cox pointed out that very few other Linux contributors have specified a particular version of the GPL. In September 2006, a survey of 29 key kernel programmers indicated 28 preferred GPLv2 to the then-current GPLv3 draft. Torvalds commented, "I think a number of outsiders... believed that I personally was just the odd man out, because I've been so publicly not a huge fan of the GPLv3."

Loadable kernel modules and firmware

It is debated whether Loadable Kernel Modules (LKMs) should be considered derivative works under copyright law, and thereby fall under the terms of the GPL. Torvalds has stated his belief that LKMs using only a limited, "public" subset of the kernel interfaces can sometimes be non-derived works, thus allowing some binary-only drivers and other LKMs that are not licensed under the GPL. Not all Linux contributors agree with this interpretation, however, and even Torvalds agrees that many LKMs are clearly derived

works, and indeed he writes that "kernel modules ARE derivative 'by default'". On the other hand Torvalds has also said that "one gray area in particular is something like a driver that was originally written for another operating system (ie. clearly not a derived work of Linux in origin). [...] THAT is a gray area, and _that_ is the area where I personally believe that some modules may be considered to not be derived works simply because they weren't designed for Linux and don't depend on any special Linux behaviour." Proprietary graphics drivers, in particular, are heavily discussed. Ultimately, it is likely that such questions can only be resolved by a court.

One point of licensing controversy is Linux's use of firmware "binary blobs" to support some hardware devices. These files are under a variety of licences, many of them restrictive and their exact underlying source code is usually unknown. Richard Stallman claims that these blobs make Linux partially non-free software, and that distributing Linux may even be violating the GPL (which requires "complete corresponding source code" to be available). In response, the FSFLA started a project, Linux-libre, to create a completely free kernel without proprietary objects, which is used by some completely free distributions.

Trademark

Linux is a registered trademark of Linus Torvalds in the United States and some other countries. This is the result of an incident in which William Della Croce, Jr., who was not involved in the Linux project, trademarked the name and subsequently demanded royalties for its use. Several Linux backers retained legal counsel and filed suit against Della Croce. The issue was settled in August 1997 when the trademark was assigned to Linus Torvalds.

SCO litigation

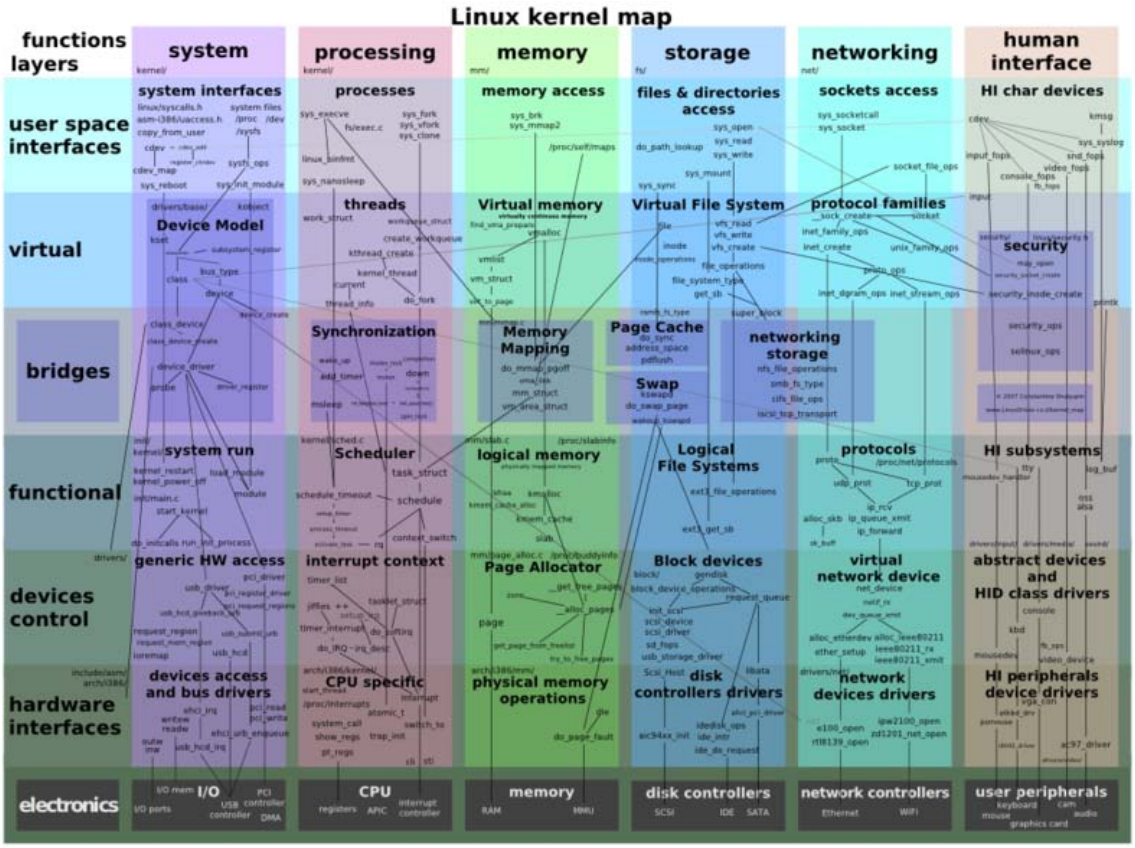
In March 2003, the SCO Group (SCO) filed a lawsuit against IBM claiming that IBM had violated copyrights that SCO claimed to hold over the Unix source code, by contributing portions of that code to Linux. Additionally, SCO sent letters to a number of companies warning that their use of Linux without a license from SCO may be a violation of copyright law, and claimed in the press that they would be suing individual Linux users. IBM then promised to defend its Linux customers on their behalf. This controversy has generated lawsuits by SCO against Novell, DaimlerChrysler (partially dismissed in July, 2004), and AutoZone, and retaliatory lawsuits by Red Hat and others against SCO.

In early 2007 SCO filed the specific details of the purported copyright infringement. Despite previous claims that SCO was the rightful owner of 1 million lines of code, they specified 326 lines of code, most of which were uncopyrightable. In August 2007, the court in the Novell case ruled that SCO did not actually own the Unix copyrights to begin with, though the Tenth Circuit Court of Appeals ruled in August 2009 that the question of who owned the copyright properly remained for a jury to answer. The jury case was decided on 30 March 2010 in Novell's favour.

Technical features

Linux supports true preemptive multitasking (both in user mode and kernel mode), virtual memory, shared libraries, demand loading, shared copy-on-write executables, memory management, the Internet protocol suite, and threading.

Architecture



Linux kernel map, bigger Interactive map and poster

Linux is a monolithic kernel. Device drivers and kernel extensions run in kernel space (ring 0 in many CPU architectures), with full access to the hardware, although some exceptions run in user space. The graphics system most people use with Linux doesn't run in the kernel, in contrast to that found in Microsoft Windows.

Kernel mode preemption allows device drivers to be preempted under certain conditions. This feature was added to handle hardware interrupts correctly and improve support for symmetric multiprocessing (SMP). Preemption also improves latency, increasing responsiveness and making Linux more suitable for real-time applications.

Kernel panic

Kernel panic

In Linux, a "panic" is an unrecoverable system error detected by the kernel contrary to similar errors detected by user space code. It is possible for kernel code to indicate such a condition by calling the `panic` function located in the header file `sys/system.h`. However, most panics are the result of unhandled processor exceptions in kernel code, such as references to invalid memory addresses. These are typically indicative of a bug somewhere in the call chain leading to the panic. They can also indicate a failure of hardware, such as a failed RAM cell or errors in arithmetic functions in the processor caused by a processor bug, overheating/damaged processor, or a soft error.

Kernel Oops

A report of a bug in the kernel is called an "OOPS". It is automatically collected by the kerneloops software or the abrt kernel oops plugin. KernelOops.org collects these reports and publishes statistics on their website.

Tanenbaum–Torvalds debate

The fact that Linux is a monolithic kernel rather than a microkernel was the topic of the Tanenbaum–Torvalds debate between Andrew S. Tanenbaum and Linus Torvalds. The debate started in 1992 about Linux and kernel architecture in general on the Usenet discussion group `comp.os.minix`. Tanenbaum argued that microkernels are superior to monolithic kernels and that therefore Linux is obsolete. Unlike traditional monolithic kernels, device drivers in Linux are easily configured as Loadable Kernel Modules and

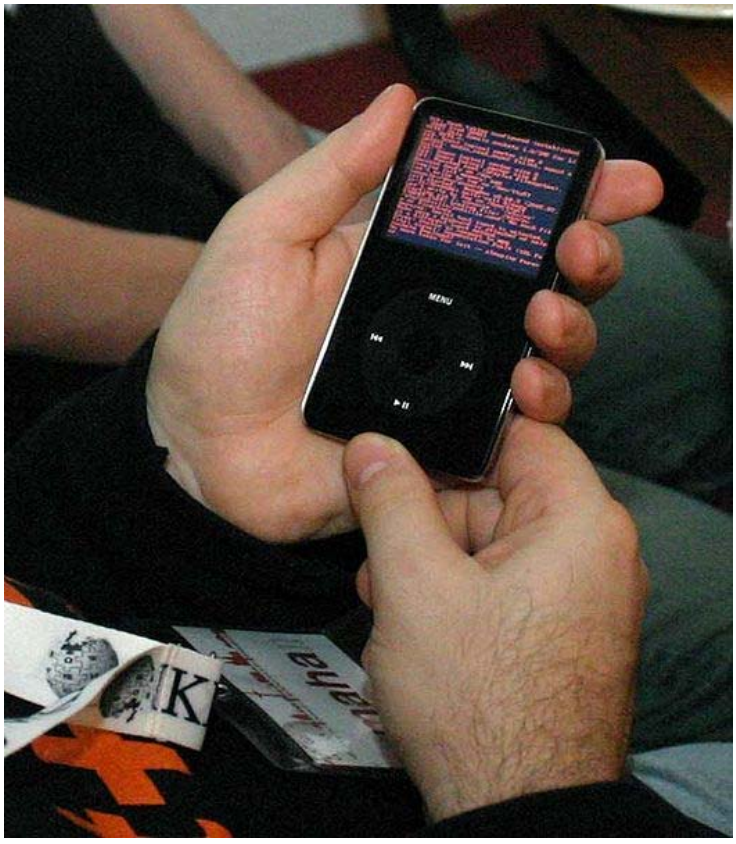
are loaded or unloaded while running the system. This subject was revisited on 9 May 2006, and on 12 May 2006 Tanenbaum wrote a position statement.

Programming languages

Linux is written in the version of the C programming language supported by GCC (which has introduced a number of extensions and changes to standard C), together with a number of short sections of code written in the assembly language (in GCC's "AT&T-style" syntax) of the target architecture. Because of the extensions to C it supports, GCC was for a long time the only compiler capable of correctly building Linux. In 2004, Intel claimed to have modified the kernel so that its C compiler also was capable of compiling it. There was another such reported success in 2009 with a modified 2.6.22.

Many other languages are used in some way, primarily in connection with the kernel build process (the methods whereby the bootable image is created from the sources). These include Perl, Python, and various shell scripting languages. Some drivers may also be written in C++, Fortran, or other languages, but this is strongly discouraged. Linux's build system only officially supports GCC as a kernel and driver compiler.

Portability



iPodLinux booting Linux

While not originally designed to be portable, Linux is now one of the most widely ported operating system kernels, running on a diverse range of systems from the iPAQ (a handheld computer) to the IBM Z/Architecture (a massive mainframe server that can run hundreds or even thousands of concurrent Linux instances). Linux runs as the main operating system on IBM's Blue Gene supercomputers. As of June 2009, Linux is the OS on 91% of systems on the Top 500 supercomputers list. Also, Linux has been ported to various handheld devices such as TuxPhone, Apple's iPod and iPhone. The Google Android, HP webOS, and Nokia Maemo operating systems, developed for mobile phone devices, all use modified versions of the Linux kernel.

Virtual machine architectures

The Linux kernel has extensive support for and runs on many virtual machine architectures both as the *host* operating system and as a *guest* operating system. The virtual machines usually emulate Intel x86 family of processors, though in a few cases PowerPC or ARM processors are also emulated.

Estimated cost to redevelop

The cost to redevelop the Linux kernel version 2.6.0 in a traditional proprietary development setting has been estimated to be $612 million USD (€467 million euro) in 2004 prices using the COCOMO man-month estimation model. In 2006, a study funded by the European Union put the redevelopment cost of kernel version 2.6.8 higher, at €882 million euro ($1.14 billion USD).

This topic was revisited in October 2008 by Amanda McPherson, Brian Proffitt and Ron Hale-Evans. Using David A. Wheeler's methodology, they estimated redevelopment of the 2.6.25 kernel now costs $1.3 billion (part of a total $10.8 billion to redevelop Fedora 9). Again, Garcia-Garcia and Alonso de Magdaleno from University of Oviedo (Spain) estimate that the value annually added to kernel was about 100 million EUR between 2005 and 2007 and 225 million EUR in 2008, it would cost also more than one billion EUR (about 1.4 billion USD) to develop in the European Union.

Feature history

Version 1.0 of the Linux kernel was released on 14 March 1994. This release of the Linux kernel only supported single-processor i386-based computer systems. Portability became a concern, and so version 1.2 (released 7 March 1995) gained support for computer systems using processors based on the Alpha, SPARC, and MIPS architectures.

Version 2.0 was released 9 June 1996. There were 41 releases in the series. The major feature of 2.0 was SMP support (that is, support for multiple processors in a single system) and support for more types of processors.

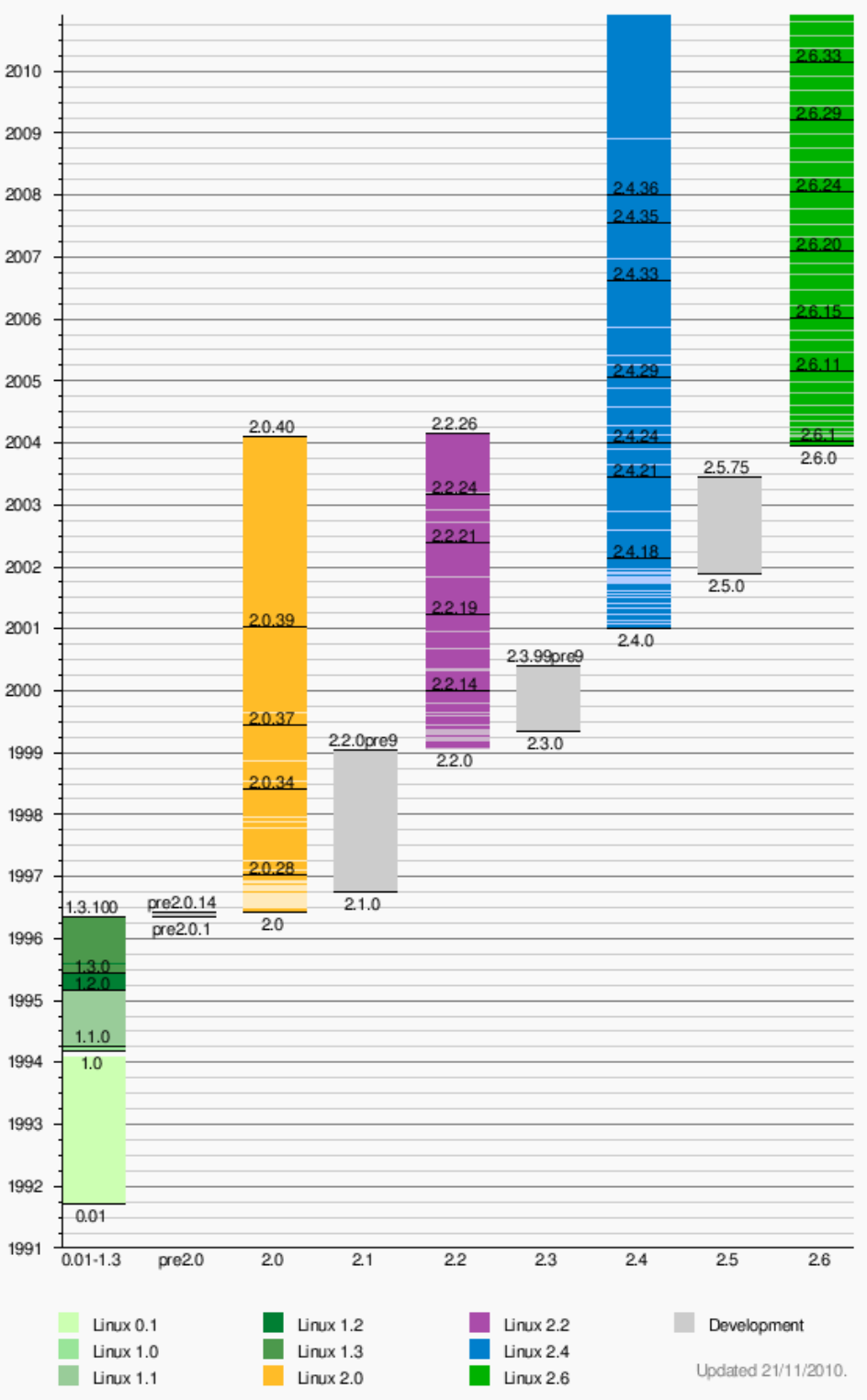
Version 2.2 (released 26 January 1999) removed the global spinlock and provided improved SMP support, and added support for the m68k and PowerPC architectures as well as new filesystems (including read-only support for Microsoft's NTFS filesystem).

Version 2.4.0, released on 4 January 2001, contained support for ISA Plug and Play, USB, and PC Cards. It also included support for the PA-RISC processor from Hewlett-Packard. Development for 2.4.*x* changed a bit in that more features were made available throughout the duration of the series, including: support for Bluetooth, Logical Volume Manager (LVM) version 1, RAID support, InterMezzo and ext3 filesystems.

Version 2.6.0 was released on 18 December 2003. The 2.6 series of kernels is still the active series of stable kernels as of November 2010. The development for 2.6.*x* changed further towards including new features throughout the duration of the series. Among the changes that have been made in the 2.6 series are: integration of μClinux into the mainline kernel sources, PAE support, support for several new lines of CPUs, integration of ALSA into the mainline kernel sources, support for up to 2^{32} users (up from 2^{16}), support for up to 2^{29} process IDs (up from 2^{15}), substantially increased the number of device types and the number of devices of each type, improved 64-bit support, support for filesystems of up to 16 terabytes, in-kernel preemption, support for the Native POSIX Thread Library, User-mode Linux integration into the mainline kernel sources, SELinux integration into the mainline kernel sources, Infiniband support, and considerably more. Also notable are the addition of several filesystems throughout the 2.6.*x* releases: FUSE, JFS, XFS, ext4 and more. Details on the history of the 2.6 kernel series can be found in the ChangeLog files on the 2.6 kernel series source code release area of kernel.org.

Development

Timeline



Development model

The current development model of the Linux kernel is such that Linus Torvalds makes the releases of new versions, also called the "vanilla" or "mainline" kernels, meaning that they contain the main, generic branch of development. This branch is officially released as a new version approximately every three months, after Torvalds does an initial round of integrating major changes made by all other programmers, and several rounds of bug-fix pre-releases.

In the current scheme, the main branch of development is not a traditional "stable" branch, instead it incorporates all kinds of changes, both the latest features as well as security and bug fixes. For users who do not want to risk updating to new versions containing code that may not be well tested, a separate set of "stable" branches exist, one for each released version, which are meant for people who just want the security and bug fixes, but not a whole new version. These branches are maintained by the *stable team* (Greg Kroah-Hartman, Chris Wright, maybe others).

Most Linux users use a kernel supplied by their Linux distribution. Some distributions ship the "vanilla" and/or "stable" kernels. However, several Linux distribution vendors (such as Red Hat and Debian) maintain another set of Linux kernel branches which are integrated into their products. These are by and large updated at a slower pace compared to the "vanilla" branch, and they usually include all fixes from the relevant "stable" branch, but at the same time they can also add support for drivers or features which had not been released in the "vanilla" version the distribution vendor started basing their branch from.

The development model for Linux 2.6 was a significant change from the development model for Linux 2.5. Previously there was a stable branch (2.4) where only relatively minor and safe changes were merged, and an unstable branch (2.5), where bigger changes and cleanups were allowed. Both of these branches had been maintained by the same set of people, led by Torvalds. This meant that users would always have a well-tested 2.4 version with the latest security and bug fixes to use, though they would have to wait for the features which went into the 2.5 branch. The downside of this was that the "stable" kernel ended up so far behind that it no longer supported recent hardware and lacked needed features. In the late 2.5.x series kernel some maintainers elected to try and back port their changes to the stable series kernel which resulted in bugs being introduced into the 2.4.x series kernel. The 2.5 branch was then eventually declared stable and renamed to 2.6. But instead of opening an unstable 2.7 branch, the kernel developers decided to continue putting major changes into the 2.6 branch, which would then be released at a pace faster than 2.4.x but slower than 2.5.x. This had the desirable effect of making new features more quickly available and getting more testing of the new code, which was added in smaller batches and easier to test.

As a response to the lack of a *stable* kernel tree where people could coordinate the collection of bug fixes as such, in December 2005 Adrian Bunk announced that he would keep releasing 2.6.16.y kernels when the stable team moved on to 2.6.17. He also

included some driver updates, making the maintenance of the 2.6.16 series very similar to the old rules for maintenance of a stable series such as 2.4. Since then, the "stable team" had been formed, and it would keep updating kernel versions with bug fixes. In October 2008 Adrian Bunk announced that he will maintain 2.6.27 for a few years as a replacement of 2.6.16. The stable team picked up on the idea and as of 2010 they continue to maintain that version and release bug fixes for it, in addition to others.

After the change of the development model with 2.6.x, developers continued to want what one might call an *unstable* kernel tree, one that changes as rapidly as new patches come in. Andrew Morton decided to repurpose his -mm tree from memory management to serve as the destination for all new and experimental code. In September 2007 Morton decided to stop maintaining this tree. In February 2008, Stephen Rothwell created the *linux-next* tree to serve as a place where patches aimed to be merged during the next development cycle are gathered. Several subsystem maintainers also adopted the suffix -*next* for trees containing code which is meant to be submitted for inclusion in the next release cycle.

Maintenance

While Linus Torvalds supervises code changes and releases to the latest kernel versions, he has delegated the maintenance of older versions to other programmers. Major releases as old as 2.0 (officially made obsolete with the kernel 2.2.0 release in January 1999) are maintained as needed, although at a very slow pace.

Kernel series	Current version	Maintainer	Support Model
2.0	2.0.40	David Weinehall	**EOL** (Officially made obsolete with the kernel 2.2.0 release)
2.2	2.2.26	Marc-Christian Petersen (former maintainer Alan Cox)	**EOL** (Unofficially obsolete with the 2.2.27-rc2)
2.4	2.4.37.11	Willy Tarreau (former maintainer Marcelo Tosatti)	Will be EOL, if no major bugs reported, by September 2011
2.6.16	2.6.16.62	Adrian Bunk	**EOL** (was the first **long-term** stable release, replaced by 2.6.27.xx)
2.6.27	2.6.27.54	Greg Kroah-Hartman	**long-term** stable release. From 9 October 2008 to July/September 2010
2.6.32	2.6.32.23	Greg Kroah-Hartman	**long-term** stable release. From 3 December 2009 to 2011 /2012
2.6.35	2.6.35.10	Andi Kleen	**long-term** stable release. From ? 2010 to 20?? (will be last **long-term** stable release)
2.6	2.6.36	Linus Torvalds	(current only) (releases every three

		months)
2.6-linux-next	next-20100820	latest development version

Other Linux kernel programmers who maintain subsystems inside the kernel include:

- Robert Love: preemptible kernel, inotify
- Ingo Molnár: x86 architecture, scheduler, locking
- David S. Miller: networking, sparc architecture
- Hans Peter Anvin: x86 architecture, kernel automounter

Revision control

The Linux kernel source code used to be maintained without the help of an automated source code management system, mostly because of Linus Torvalds' dislike of centralized SCM systems.

In 2002, Linux kernel development switched to BitKeeper, a SCM system which satisfied Linus Torvalds' technical requirements. BitKeeper was made available to Linus and several others free of charge, but was not free software, which was a source of controversy. The system did provide some interoperability with free SCM systems such as CVS and Subversion.

In April 2005, however, efforts to reverse-engineer the BitKeeper system by Andrew Tridgell led BitMover, the company which maintained BitKeeper, to stop supporting the Linux development community. In response, Linus Torvalds and others wrote a new source code control system for the purpose, called Git. The new system was written within weeks, and in two months the first official kernel release was made using git. Git soon developed into a separate project in its own right and gained wider adoption in the free software community.

Version numbering

The Linux kernel has had three different numbering schemes.

The first version of the kernel was 0.01. This was followed by 0.02, 0.03, 0.10, 0.11, 0.12 (the first GPL version), 0.95, 0.96, 0.97, 0.98, 0.99 and then 1.0. From 0.95 on there were many patch releases between versions.

After the 1.0 release and prior to version 2.6, the version was composed as "**A.B.C**", where the number A denoted the kernel version, the number B denoted the major revision of the kernel, and the number C indicated the minor revision of the kernel. The version was changed only when major changes in the code and the concept of the kernel occurred, twice in the history of the kernel: In 1994 (version 1.0) and in 1996 (version 2.0). The major revision was used according to the traditional even-odd system version

numbering system. The minor revision had been changed whenever security patches, bug fixes, new features or drivers were implemented in the kernel.

Since 2004, after version 2.6.0 was released, the kernel developers held several discussions regarding the release and version scheme and ultimately Linus Torvalds and others decided that a much shorter release cycle would be beneficial. Since then, the version has been composed of three or four numbers. The first two numbers became largely irrelevant, and the third number is the actual version of the kernel. The fourth number accounts for bug and security fixes (only) to the kernel version.

The first use of the fourth number occurred when a grave error, which required immediate fixing, was encountered in 2.6.8's NFS code. However, there were not enough other changes to legitimize the release of a new minor revision (which would have been 2.6.9). So, 2.6.8.1 was released, with the only change being the fix of that error. With 2.6.11, this was adopted as the new official versioning policy. Later it became customary to continuously back-port major bug-fixes and security patches to released kernels and indicate that by updating the fourth number.

Regular development pre-releases are titled release candidates, which is indicated by appending the suffix 'rc' to the kernel version, followed by an ordinal number.

Also, sometimes the version will have a suffix such as 'tip', indicating another development branch, usually (but not always) the initials of a person who made it. For example, 'ck' stands for Con Kolivas, 'ac' stands for Alan Cox, etc. Sometimes, the letters are related to the primary development area of the branch the kernel is built from, for example, 'wl' indicates a wireless networking test build. Also, distributors may have their own suffixes with different numbering systems and for back-ports to their "Enterprise" (i.e. stable but older) distribution versions.

Chapter 3

History of Linux

The **History of Linux** began in 1991 with the commencement of a personal project by a Finnish student, Linus Torvalds, to create a new operating system kernel.

Since then the resulting Linux kernel has been marked by constant growth throughout its history. Since the initial release of its source code in 1991, it has grown from a small number of C files under a license prohibiting commercial distribution to its state in 2009 of over 370 megabytes of source under the GNU General Public License.

Events leading to creation

The Unix operating system was conceived and implemented in the 1960s and first released in 1970. Its availability and portability caused it to be widely adopted, copied and modified by academic institutions and businesses. Its design became influential to authors of other systems.

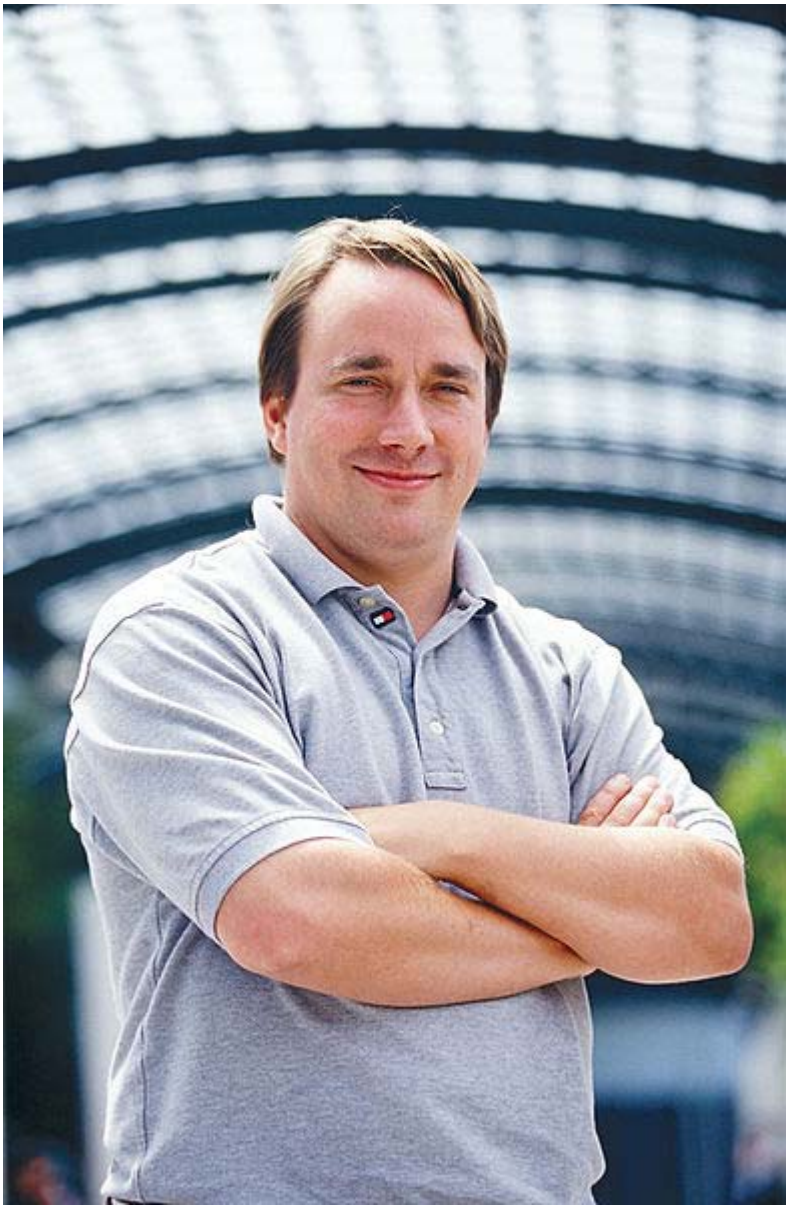
In 1983, Richard Stallman started the GNU project with the goal of creating a free UNIX-like operating system. As part of this work, he wrote the GNU General Public License (GPL). By the early 1990s there was almost enough available software to create a full operating system. However, the GNU kernel, called Hurd, failed to attract enough attention from developers leaving GNU incomplete.

Another free operating system project in the 1980s was the Berkeley Software Distribution (BSD). This was developed by UC Berkeley from the 6th edition of Unix from AT&T. Since BSD contained Unix code that AT&T owned, AT&T filed a lawsuit (USL v. BSDi) in the early 1990s against the University of California. This strongly limited the development and adoption of BSD.

MINIX, a Unix-like system intended for academic use, was released by Andrew S. Tanenbaum in 1987. While source code for the system was available, modification and redistribution were restricted. In addition, MINIX's 16-bit design was not well adapted to the 32-bit features of the increasingly cheap and popular Intel 386 architecture for personal computers.

These factors and the lack of a widely-adopted, free kernel provided the impetus for Torvalds's starting his project. He has stated that if either the GNU or 386BSD kernels were available at the time, he likely would not have written his own.

The creation of Linux



Linus Torvalds in 2002

In 1991, in Helsinki, Linus Torvalds began a project that later became the Linux kernel. It was initially a terminal emulator, which Torvalds used to access the large UNIX servers of the university. He wrote the program specifically for the hardware he was using and independent of an operating system because he wanted to use the functions of his new PC with an 80386 processor. Development was done on MINIX using the GNU C compiler,

which is still the main choice for compiling Linux today (although the code can be built with other compilers, such as the Intel C Compiler).

As Torvalds wrote in his book *Just for Fun*, he eventually realized that he had written an operating system kernel. On 25 August 1991, he announced this system in a Usenet posting to the newsgroup "comp.os.minix.":

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

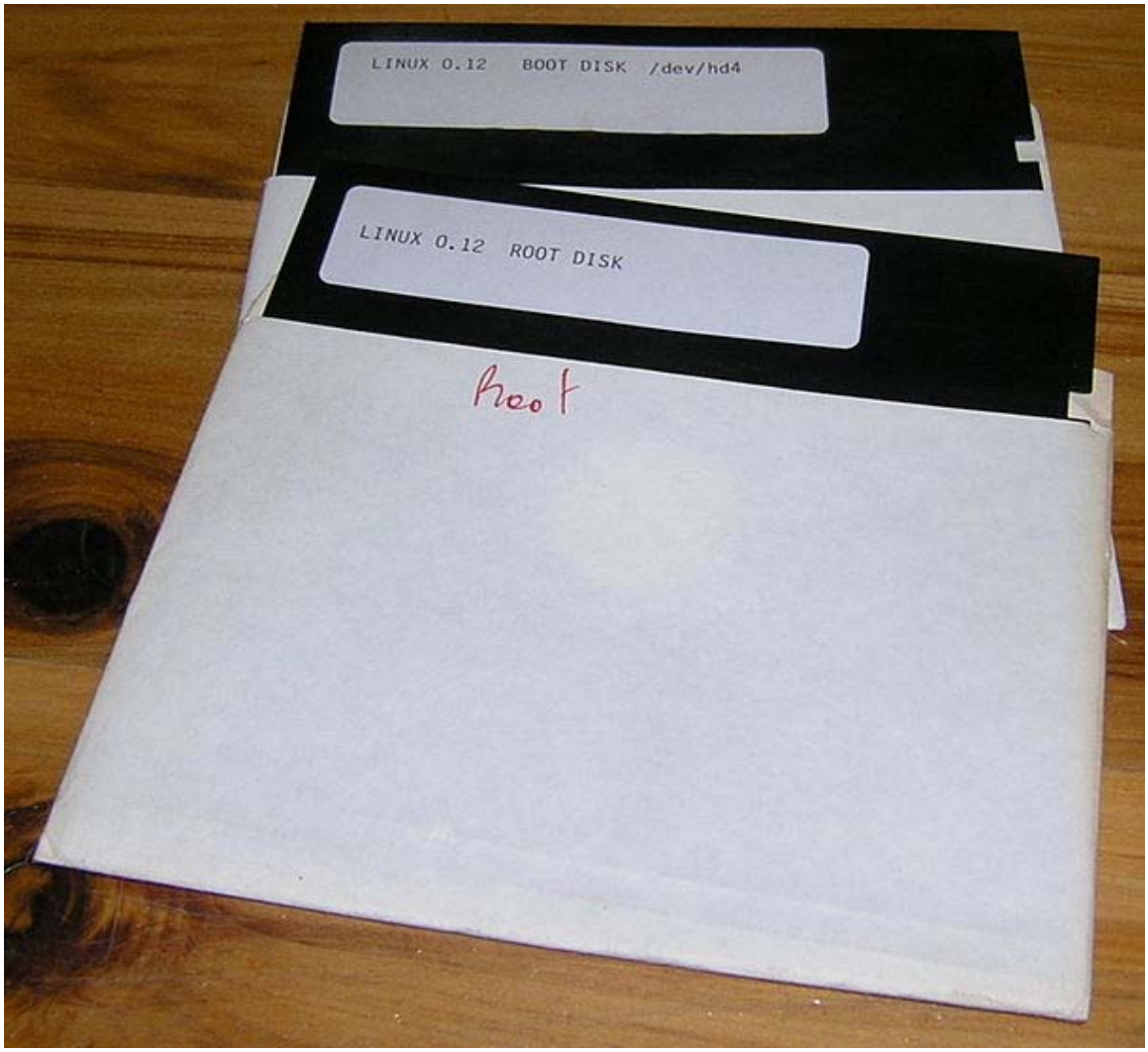
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

—Linus Torvalds

The name



Floppy discs holding a very early version of Linux

Linus Torvalds had wanted to call his invention Freax, a portmanteau of "freak", "free", and "x" (as an allusion to Unix). During the start of his work on the system, he stored the files under the name "Freax" for about half of a year. Torvalds had already considered the name "Linux," but initially dismissed it as too egotistical.

In order to facilitate development, the files were uploaded to the FTP server (ftp.funet.fi) of FUNET in September 1991. Ari Lemmke, Torvald's coworker at the University of Helsinki who was one of the volunteer administrators for the FTP server at the time, did not think that "Freax" was a good name. So, he named the project "Linux" on the server without consulting Torvalds. Later, however, Torvalds consented to "Linux".

To demonstrate how the word "Linux" should be pronounced, Torvalds included an audio guide with the kernel source code.

Linux under the GNU GPL

Torvalds first published the Linux kernel under its own licence, which had a restriction on commercial activity.

The software to use with the kernel was software developed as part of the GNU project licensed under the GNU General Public License, a free software license. The first release of the Linux kernel, Linux 0.01, included a binary of GNU's Bash shell.

In the "Notes for linux release 0.01", Torvalds lists the GNU software that is required to run Linux:

Sadly, a kernel by itself gets you nowhere. To get a working system you need a shell, compilers, a library etc. These are separate parts and may be under a stricter (or even looser) copyright. Most of the tools used with linux are GNU software and are under the GNU copyleft.

In 1992, he suggested releasing the kernel under the GNU General Public License. He first announced this decision in the release notes of version 0.12. In the middle of December 1992 he published version 0.99 using the GNU GPL.

Linux and GNU developers worked to integrate GNU components with Linux to make a fully-functional and free operating system.

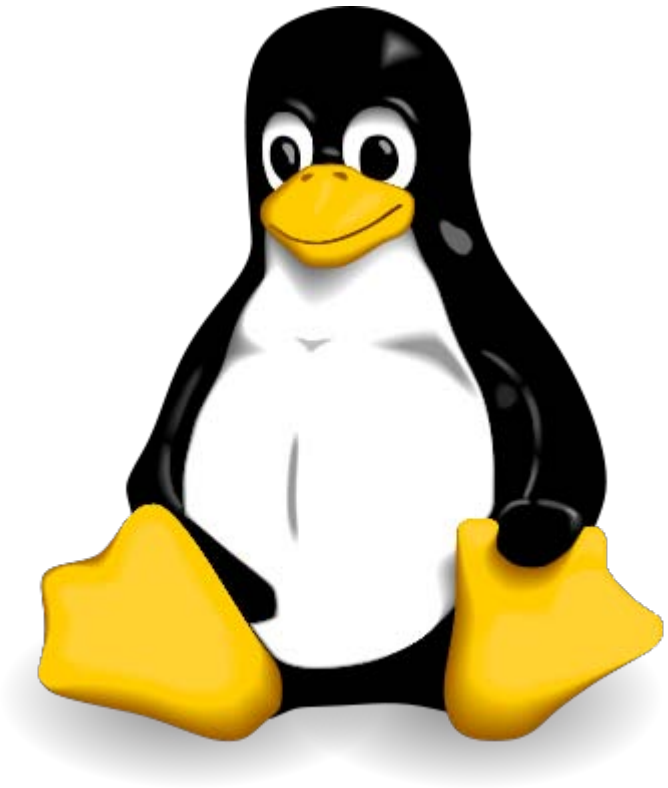
Torvalds has stated, “making Linux GPL'd was definitely the best thing I ever did.”

GNU/Linux naming controversy

The designation "Linux" was initially used by Torvalds only for the Linux kernel. The kernel was, however, frequently used together with other software, especially that of the GNU project. This quickly became the most popular adoption of GNU software. In June 1994 in GNU's bulletin, Linux was referred to as a "free UNIX clone", and the Debian project began calling its product *Debian GNU/Linux*. In May 1996, Richard Stallman published the editor Emacs 19.31, in which the type of system was renamed from Linux to Lignux. This spelling was intended to refer specifically to the combination of GNU and Linux, but this was soon abandoned in favor of "GNU/Linux".

This name garnered varying reactions. The GNU and Debian projects use the name, although most developers simply use the term "Linux" to refer to the combination.

Official mascot



Tux

Torvalds announced in 1996 that there would be a mascot for Linux, a penguin. Larry Ewing provided the original draft of today's well known mascot based on this description. The name Tux was suggested by James Hughes as derivative of *Torvalds' UniX*.

New development

Kernel

There are many other well-known maintainers for the Linux kernel beside Torvalds such as Alan Cox and Marcelo Tosatti. Cox maintained version 2.2 of the kernel until it was discontinued at the end of 2003. Likewise, Tosatti maintained version 2.4 of the kernel until the middle of 2006. Andrew Morton steers the development and administration of the 2.6 kernel, which was released on 18 December 2003 in its first stable incarnation. Also the older branches are still constantly improved.

Community

The largest part of the work on Linux is performed by the community: the thousands of programmers around the world that use Linux and send their suggested improvements to the maintainers. Various companies have also helped not only with the development of

the Kernels, but also with the writing of the body of auxiliary software, which is distributed with Linux.

It is released both by organized projects such as Debian, and by projects connected directly with companies such as Fedora and openSUSE. The members of the respective projects meet at various conferences and fairs, in order to exchange ideas. One of the largest of these fairs is the LinuxTag in Germany (currently in Berlin), where about 10,000 people assemble annually, in order to discuss Linux and the projects associated with it.

Open Source Development Lab and Linux Foundation

The Open Source Development Lab (OSDL) was created in the year 2000, and is an independent nonprofit organization which pursues the goal of optimizing Linux for employment in data centers and in the carrier range. It served as sponsored working premises for Linus Torvalds and also for Andrew Morton (until the middle of 2006 when Morton transferred to Google). Torvalds works full-time on behalf of OSDL, developing the Linux Kernels.

On January 22, 2007, OSDL and the Free Standards Group merged to form The Linux Foundation, narrowing their respective focuses to that of promoting GNU/Linux in competition with Microsoft Windows.

Companies

Despite being open-source, a few companies profit from Linux. These companies, most of which are also members of the Open Source Development Lab, invest substantial resources into the advancement and development of Linux, in order to make it suited for various application areas. This includes hardware donations for driver developers, cash donations for people who develop Linux software, and the employment of Linux programmers at the company. Some examples are IBM and HP, which use Linux on their own servers, and Red Hat, which maintains its own distribution. Likewise Nokia supports Linux by the development and LGPL licensing of Qt, which makes the development of KDE possible, and by employing some of the X and KDE developers.

Controversy over Linux

Linux has been surrounded by controversy repeatedly since its inception.

"Linux is obsolete"

In 1992 Andrew S. Tanenbaum, recognized computer scientist and author of the Minix microkernel system, wrote a Usenet article on the newsgroup comp.os.minix with the title "Linux is obsolete", which marked the beginning of a famous debate about the structure of the then-recent Linux kernel. Among the most significant criticisms were that:

- The kernel was monolithic and thus old-fashioned.
- The lack of portability, due to the use of exclusive features of the Intel 386 processor. "Writing a new operating system that is closely tied to any particular piece of hardware, especially a weird one like the Intel line, is basically wrong."
- There was no strict control of the source code by any individual person.
- Linux employed a set of features which were useless (Tanenbaum believed that multithreaded file systems were simply a "performance hack").

Tanenbaum's prediction that Linux would become outdated within a few years and replaced by GNU Hurd (which he considered to be more modern) proved incorrect. Linux has been ported to all major platforms and its open development model has led to an exemplary pace of development. In contrast, GNU Hurd has not yet reached the level of stability that would allow it to be used on a production server.

Samizdat

In his unpublished book *Samizdat*, Ken Brown claims that Torvalds illegally copied code from MINIX. These claims have been refuted by Tanenbaum:

He [Ken Brown] wanted to go on about the ownership issue, but he was also trying to avoid telling me what his real purpose was, so he didn't phrase his questions very well. Finally he asked me if I thought Linus wrote Linux. I said that to the best of my knowledge, Linus wrote the whole kernel himself, but after it was released, other people began improving the kernel, which was very primitive initially, and adding new software to the system--essentially the same development model as MINIX. Then he began to focus on this, with questions like: "Didn't he steal pieces of MINIX without permission." I told him that MINIX had clearly had a huge influence on Linux in many ways, from the layout of the file system to the names in the source tree, but I didn't think Linus had used any of my code.

The book's claims, methodology and references were seriously questioned and in the end it was never released and was delisted from the distributor's site.

Competition from Microsoft

Although Torvalds has said that Microsoft's feeling threatened by Linux in the past was of no consequence to him, the Microsoft and Linux camps had a number of antagonistic interactions between 1997 and 2001. This became quite clear for the first time in 1998, when the first Halloween document was brought to light by Eric S. Raymond. This was a short essay by a Microsoft developer that sought to lay out the threats posed to Microsoft by free software and identified strategies to counter these perceived threats. However the Free Software Foundation issued a statement that Microsoft's production of proprietary software is bad for software users because it denies users "their rightful freedom."

Competition entered a new phase in the beginning of 2004, when Microsoft published results from customer case studies evaluating the use of Windows vs. Linux under the

name “Get the Facts” on its own web page. Based on inquiries, research analysts, and some Microsoft sponsored investigations, the case studies claimed that enterprise use of Linux on servers compared unfavorably to the use of Windows in terms of reliability, security, and total cost of ownership.

In response, commercial Linux distributors produced their own studies, surveys and testimonials to counter Microsoft's campaign. Novell's web-based campaign at the end of 2004 was entitled “Unbending the truth” and sought to outline the advantages as well as dispelling the widely publicized legal liabilities of Linux deployment (particularly in light of the *SCO v IBM* case). Novell particularly referenced the Microsoft studies in many points. IBM also published a series of studies under the title “The Linux at IBM competitive advantage” to again parry Microsoft's campaign. Red Hat had a campaign called “Truth Happens” aimed at letting the performance of the product speak for itself, rather than advertising the product by studies.

In the autumn of 2006, Novell and Microsoft announced an agreement to co-operate on software interoperability and patent protection. This included an agreement that customers of either Novell or Microsoft may not be sued by the other company for patent infringement. This patent protection was also expanded to non-free software developers. The last part was criticized because it only included non-commercial developers.

In July 2009, Microsoft submitted 22,000 lines of source code to the Linux kernel under the GPLV2 license, which were subsequently accepted. Although this has been referred to as "a historic move" and as a possible bellwether of an improvement in Microsoft's corporate attitudes toward Linux and open-source software, the decision was not altogether altruistic, as it promised to lead to significant competitive advantages for Microsoft and avoided legal action against Microsoft. Microsoft was actually compelled to make the code contribution when Vyatta principal engineer and Linux contributor Stephen Hemminger discovered that Microsoft had incorporated a Hyper-V network driver, with GPL-licenced open source components, statically linked to closed-source binaries in contravention of the GPL licence. Microsoft contributed the drivers to rectify the licence violation, although the company attempted to portray it as a charitable act, rather than one to avoid legal action against it. In the past Microsoft had termed Linux a "cancer" and "communist"

SCO

In March 2003, the SCO Group accused IBM of violating their copyright on UNIX by transferring code from UNIX to Linux. SCO claims ownership of the copyrights on UNIX and a lawsuit was filed against IBM. Red Hat has countersued and SCO has since filed other related lawsuits. At the same time as their lawsuit, SCO began selling Linux licenses to users who did not want to risk a possible complaint on the part of SCO. Since Novell also claims the copyrights to UNIX, it filed suit against SCO.

SCO has since filed for bankruptcy.

Trademark rights

In 1994 and 1995, several people from different countries attempted to register the name "Linux" as a trademark. Thereupon requests for royalty payments were issued to several Linux companies, a step with which many developers and users of Linux did not agree. Linus Torvalds clamped down on these companies with help from Linux International and was granted the trademark to the name, which he transferred to Linux International. Protection of the trademark was later administered by a dedicated foundation, the non-profit Linux Mark Institute. In 2000, Linus Torvalds specified the basic rules for the assignment of the licenses. This means that anyone who offers a product or a service with the name *Linux* must possess a license for it, which can be obtained through a unique purchase.

In June 2005, a new controversy developed over the use of royalties generated from the use of the Linux trademark. The Linux Mark Institute, which represents Linus Torvalds' rights, announced a price increase from 500 to 5,000 dollars for the use of the name. This step was justified as being needed to cover the rising costs of trademark protection.

In response to this increase, the community became displeased, which is why Linus Torvalds made an announcement on 21 August 2005, in order to dissolve the misunderstandings. In an e-mail he described the current situation as well as the background in detail and also dealt with the question of who had to pay license costs:

[...] And let’s repeat: somebody who doesn’t want to _protect_ that name would never do this. You can call anything "MyLinux", but the downside is that you may have somebody else who _did_ protect himself come along and send you a cease-and-desist letter. Or, if the name ends up showing up in a trademark search that LMI needs to do every once in a while just to protect the trademark (another legal requirement for trademarks), LMI itself might have to send you a cease-and-desist-or-sublicense it letter.

At which point you either rename it to something else, or you sublicense it. See? It’s all about whether _you_ need the protection or not, not about whether LMI wants the money or not.

[...] Finally, just to make it clear: not only do I not get a cent of the trademark money, but even LMI (who actually administers the mark) has so far historically always lost money on it. That’s not a way to sustain a trademark, so they’re trying to at least become self-sufficient, but so far I can tell that lawyers fees to _give_ that protection that commercial companies want have been higher than the license fees. Even pro bono lawyers charge for the time of their costs and paralegals etc.

—Linus Torvalds

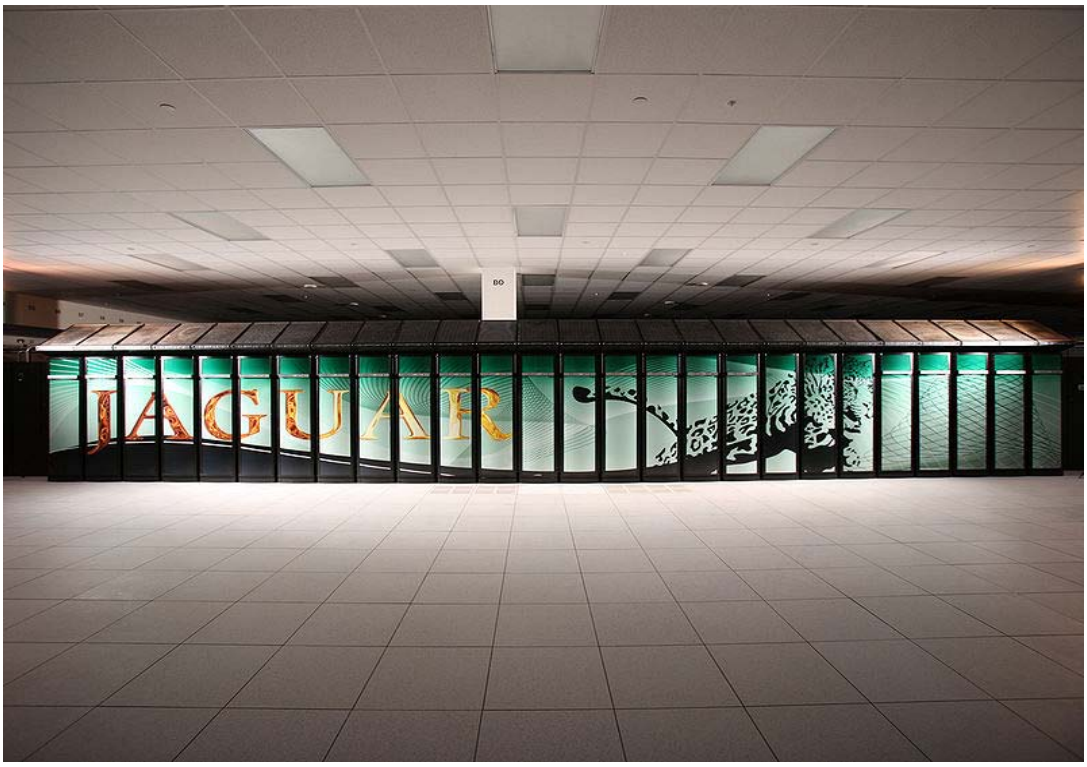
The Linux Mark Institute has since begun to offer a free, perpetual worldwide sublicense.

Chronology

- 1983: Richard Stallman creates the GNU project with the goal of creating a free operating system.
- 1989: Richard Stallman writes the first version of the GNU General Public License.
- 1991: The Linux kernel is publicly announced on 25 August by the 21 year old Finnish student Linus Benedict Torvalds.
- 1992: The Linux kernel is relicensed under the GNU GPL. The first so called "Linux distributions" are created.
- 1993: Over 100 developers work on the Linux kernel. With their assistance the kernel is adapted to the GNU environment, which creates a large spectrum of application types for Linux. The oldest currently existing Linux distribution, Slackware, is released for the first time. Later in the same year, the Debian project is established. Today it is the largest community distribution.
- 1994: In March Torvalds judges all components of the kernel to be fully matured: he releases version 1.0 of Linux. The XFree86 project contributes a graphic user interface (GUI). In this year the companies Red Hat and SUSE publish version 1.0 of their Linux distributions.
- 1995: Linux is ported to the DEC Alpha and to the Sun SPARC. Over the following years it is ported to an ever greater number of platforms.
- 1996: Version 2.0 of the Linux kernel is released. The kernel can now serve several processors at the same time, and thereby becomes a serious alternative for many companies.
- 1998: Many major companies such as IBM, Compaq and Oracle announce their support for Linux. In addition a group of programmers begins developing the graphic user interface KDE.
- 1999: A group of developers begin work on the graphic environment GNOME, which should become a free replacement for KDE, which depended on the then proprietary Qt toolkit. During the year IBM announces an extensive project for the support of Linux.
- 2004: The XFree86 team splits up and joins with the existing X Window standards body to form the X.Org Foundation, which results in a substantially faster development of the X Window Server for Linux.
- 2005: The project openSUSE begins a free distribution from Novell's community. Also the project OpenOffice.org introduces version 2.0 that now supports OASIS OpenDocument standards in October.
- 2006: Oracle releases its own distribution of Red Hat. Novell and Microsoft announce a cooperation for a better interoperability.
- 2007: Dell starts distributing laptops with Ubuntu pre-installed in them.

Chapter 4

Linux Adoption



The Oak Ridge National Laboratory's Jaguar supercomputer, from July 2009, until surpassed in October 2010 by the Chinese Tianhe-I, was the world's fastest supercomputer. It uses the Cray Linux Environment as its operating system.

Linux adoption refers to new use of the Linux computer operating system by homes, organizations, companies, and governments, while **Linux migration** refers to the change from using other operating systems to using Linux.

Many factors have resulted in increased use of Linux systems by traditional desktop users as well as operators of server systems, including desire for decreased operating system cost, increased security and support for open-source principles. Several national governments have passed policies moving governmental computers to Linux from proprietary systems in the last few years.

In August 2010 Jeffrey Hammond, the principal analyst at Forrester Research pronounced: "Linux has crossed the chasm to mainstream adoption." His declaration was based on the huge number of enterprises that had moved to Linux during the late-2000s recession. In a company survey completed in the third quarter of 2009, 48% of companies surveyed reported using an open source operating system.

History

Gartner claimed that Linux-powered personal computers accounted for 4% of unit sales in 2008. However, it is common for users to install Linux in addition to (as a dual boot arrangement) or in place of the pre-existing Microsoft Operating platform.

Timeline

- 1983 (September): GNU project was announced publicly
- 1991 (September): first version of the Linux kernel was released to the Internet
- 1999 EmperorLinux started shipping specially configured laptops running modified Linux distributions to ensure usability
- 2001 (second quarter): Linux server unit shipments recorded a 15% annual growth rate
- 2004: Linux shipped on approximately 50% of the worldwide server blade units, and 20% of all rack-optimized servers

2007

- Dell announced it would ship select models with Ubuntu Linux pre-installed
- Lenovo announced it would ship select models with SUSE Linux Enterprise Desktop pre-installed
- HP announced that it would begin shipping computers preinstalled with Red Hat Linux in Australia
- ASUS launched the Linux-based ASUS Eee PC

2008

- Dell announced it would begin shipping Ubuntu-based computers to Canada and Latin America
- Dell began shipping systems with Ubuntu pre-installed in China
- Acer launched the Linux-based Acer Aspire One
- In June 2008 the Electronics Corporation of Tamil Nadu (ELCOT) a bulk computer buyer for students in the Indian state of Tamil Nadu decided to switch entirely to supplying Linux after Microsoft attempted to use its monopoly position to sell the organization Windows bundled with Microsoft Office. ELCOT declined the offer stating "Any such bundling could result in serious exploitation of the consumer."
- In August 2008 IBM cited market disillusionment with Microsoft Vista in announcing a new partnership arrangement with Red Hat, Novell and Canonical

to offer "Microsoft-free" personal computers with IBM application software, including Lotus Notes and Lotus Symphony.
- Microsoft denied paying a Nigerian contractor $400,000 to replace Linux on school computers with its own software

2009

- In January 2009 the New York Times stated: "More than 10 million people are estimated to run Ubuntu today".

- In mid-2009 Asus stopped offering Linux as part of its *It's better with Windows* campaign, receiving strong criticism for it. The company claimed that competition from other netbook makers drove them to offer only Windows XP. Writing in May 2010 ComputerWorld columnist Steven J. Vaughan-Nichols said "I'm sure that the real reason is Microsoft has pressured Asus into abandoning Linux. On ASUS' site, you'll now see the slogan "ASUS recommends Windows 7" proudly shown. Never mind that, while Windows 7 is a good operating system, Windows 7 is awful on netbooks."

- In May 2009, Fedora developer Jef Spaleta estimated on the basis of IP addresses of update downloads and statistics from the voluntary user hardware registration service Smolt that there are 16 million Fedora systems in use. No effort was made to estimate how much the Fedora installed base overlaps with other Linux distributions (enthusiasts installing many distributions on the same system).

- In June 2009 ZDNet reported "Worldwide, there are 13 million active Ubuntu users with use growing faster than any other distribution."

2010

- In April 2010 Chris Kenyon, vice president for OEM at Canonical Ltd. estimated that there were 12 million Ubuntu users.

- In June 2010 a Quebec Superior Court Judge Denis Jacques ruled that the provincial government broke the law when it spent Cdn$720,000, starting in the fall of 2006 to migrate 800 government workstations to Microsoft Windows Vista and Office 2007 without carrying out a "serious and documented search" for alternatives. The search for alternatives was legally required for any expenditures over Cdn$25,000. The court case was brought by *Savoir Faire Linux*, a small Montreal-based company that had hoped to bid Linux software to replace the government's aging Windows XP. The judge dismissed the government's contention that Microsoft software was chosen because employees were already familiar with Windows and that switching to a different operating system would have cost more.

Types of adopters

KDE Linux desktop

Although Linux's status as mainstream operating system is relatively recent, it is in use in many different environments, including government, education, home, business and scientific institutions.

Government

As local governments come under pressure from institutions such as the World Trade Organization and the International Intellectual Property Alliance, some have turned to Linux and other Free Software as an affordable, legal alternative to both pirated software and expensive proprietary computer products from Microsoft, Apple and other commercial companies. The spread of Linux affords some leverage for these countries when companies from the developed world bid for government contracts (since a low-cost option exists), while furnishing an alternative path to development for countries like India and Pakistan that have many citizens skilled in computer applications but cannot afford technological investment at "First World" prices.

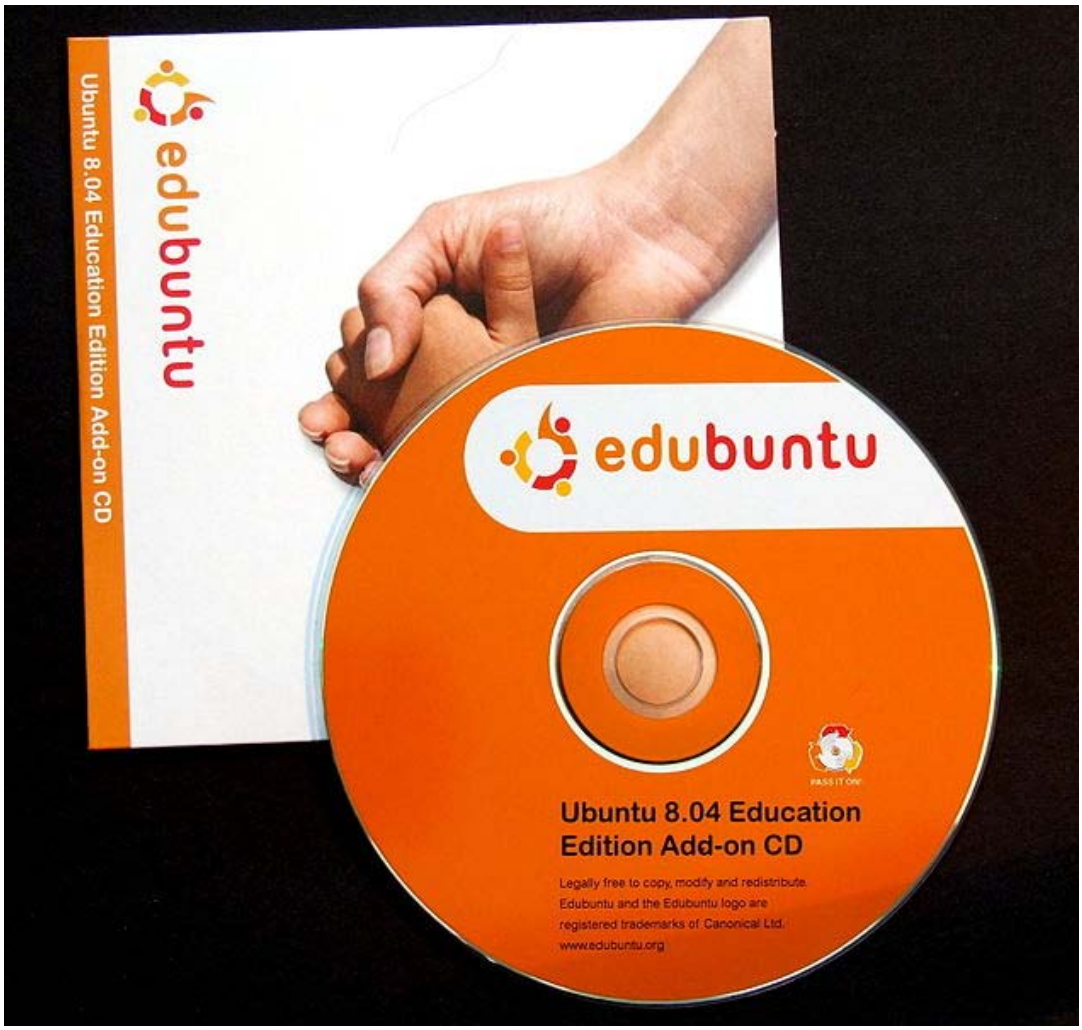
- Brazil's PC Conectado program utilizing Linux
- City of Munich has chosen 2003 to migrate its 14,000 desktops to Debian-based LiMux, but until 2010 an adoption rate of only 20% was achieved.
- The United States Department of Defense uses Linux - "the U.S. Army is “the” single largest install base for Red Hat Linux" and the US Navy nuclear submarine fleet runs on Linux.

- The city of Vienna has chosen to start migrating its desktop PCs to Debian-based Wienux . However, the idea was largely abandoned, because the necessary software was incompatible with Linux.
- Spain was noted as the furthest along the road to Linux adoption in 2003, for example with Linux distribution LinEx
- State owned Industrial and Commercial Bank of China (ICBC) is installing Linux in all of its 20,000 retail branches as the basis for its web server and a new terminal platform. (2005)
- In April 2006, the US Federal Aviation Administration announced that it had completed a migration to Red Hat Enterprise Linux in one third of the scheduled time and saved 15 million dollars.
- The Government of Pakistan established a Technology Resource Mobilization Unit in 2002 to enable groups of professionals to exchange views and coordinate activities in their sectors and to educate users about free software alternatives. Linux is an option for poor countries which have little revenue for public investment; Pakistan is using open source software in public schools and colleges, and hopes to run all government services on Linux eventually.
- The French Parliament has switched to using Ubuntu on desktop PCs.
- The Federal Employment Office of Germany (Bundesagentur für Arbeit) has migrated 13,000 public workstations from Windows NT to OpenSuse.
- Czech Post migrated 4000 servers and 12,000 clients to Novell Linux in 2005
- German Foreign ministry is migrating all of its 11,000 desktops to Linux and other Open source applications.
- Cuba - Students from the Cuban University of Information Science launched its own distribution of Linux called "Nova" to promote the replace of Microsoft Windows on civilian and government computers, a project that is now supported by the Cuban Government.
- The Canton of Solothurn in Switzerland decided 2001 to migrate its computers to Linux, but 2010 the Swiss authority has made a U-turn by deciding to use Windows 7 for desktop clients.
- France's national police force, the National Gendarmerie started moving their 90,000 desktops from Windows XP to Ubuntu in 2007 over concerns about the additional training costs of moving to Windows Vista. The migration should be completed by 2015. The force has saved about €50 million on software licensing between 2004 and 2008.
- France's Ministry of Agriculture uses Mandriva Linux.
- Macedonia's Ministry of Education and Science deployed more than 180,000 Ubuntu based classroom desktops, and has encouraged every student in the Republic of Macedonia to use Ubuntu computer workstations.
- The People's Republic of China exclusively uses Linux as the operating system for its Loongson processor family, with the aim of technology independence.
- The US National Nuclear Security Administration operates the world's third fastest supercomputer, the IBM Roadrunner, which uses Red Hat Enterprise Linux along with Fedora as its operating systems.
- The regional Andalusian Autonomous Government of Andalucía in Spain developed its own Linux distribution, called Guadalinex in 2004.

- The South African Social Security Agency (SASSA) deployed Multi-station Linux Desktops to address budget and infrastructure constraints in 50 rural sites.
- In 2003, the Turkish government decided to create its own Linux distribution, Pardus, developed by UEKAE (National Research Institute of Electronics and Cryptology). The first version, Pardus 1.0, was officially announced in 27 December 2005.
- In 2010 The Philippines fielded an Ubuntu-powered national voting system.

- In July 2010 Malaysia had switched 703 of the state's 724 agencies to Free and Open Source software with a Linux based operating system used. The Chief Secretary to the Government cited, "(the) general acceptance of its promise of better quality, higher reliability, more flexibility and lower cost".
- In late 2010 Vladimir Putin signed a plan to move the Russian Federation government towards free software including Linux in the second quarter of 2012.

Education



Edubuntu CD kit

Linux is often used in technical disciplines at universities and research centres. This is due to several factors, including that Linux is available free of charge and includes a large body of free/open source software. To some extent, technical competence of computer science and software engineering academics is also a contributor, as is stability, maintainability, and upgradability. IBM ran an advertising campaign entitled "Linux is Education" featuring a young boy who was supposed to be "Linux".

Examples of large scale adoption of Linux in education include the following:

- The OLPC XO-1 (previously called the MIT $100 laptop and The Children's Machine), is an inexpensive laptop running Linux, which will be distributed to millions of children as part of the One Laptop Per Child project, especially in developing countries.
- Republic of Macedonia deployed 5,000 Linux desktops running Ubuntu across all 468 public schools and 182 computer labs (December 2005). Later in 2007, another 180,000 Ubuntu thin client computers were deployed.
- Schools in Bolzano, Italy, with a student population of 16,000, switched to a custom distribution of Linux, (FUSS Soledad GNU/Linux), in September 2005.
- Brazil has around 20,000 Linux desktops running in elementary and secondary public schools.
- Government officials of Kerala, India announced they will use only free software, running on the Linux platform, for computer education, starting with the 2,650 government and government-aided high schools.
- 22,000 students in the US state of Indiana had access to Linux Workstations at their high schools in 2006.
- Germany has announced that 560,000 students in 33 universities will migrate to Linux.
- The Philippines has deployed 13,000 desktops running on Fedora, the first 10,000 where delivered in December 2007 by ASI. Another 10,000 desktops of Edubuntu and Kubuntu are planned.
- Russia announced in October 2007 that all its school computers will run on Linux. This is to avoid cost of licensing currently unlicensed software.
- In 2004 Georgia began running all its school computers and LTSP thin clients on Linux, mainly using Kubuntu, Ubuntu and stripped Fedora-based distros.
- 9,000 computers to be converted to Linux and OpenOffice.org in school district Geneva, Switzerland by September 2008
- The Indian state of Tamil Nadu plans to distribute 100,000 Linux laptops to its students.
- The Chinese government is buying 1.5 million Linux Loongson PCs as part of its plans to support its domestic industry. In addition the province of Jiangsu will install as many as 150,000 Linux PCs, using Loongson processors, in rural schools starting in 2009.
- The Indian government's tablet computer initiative for student use employs Linux as the operating system as part of its drive to produce a tablet PC for under 1,500 rupees (US$35).

Home

- Sony's PlayStation 3 came with a hard disk (20GB, 60GB or 80GB) and was specially designed to allow easy installation of Linux on the system. However, Linux was prevented from accessing certain functions of the PlayStation such as 3D graphics. Sony also released a Linux kit for its PlayStation 2 console. Playstation hardware running Linux has been occasionally used in small scale distributed computing experiments, due to the ease of installation and the relatively low price of a PS3 compared to other hardware choices offering similar performance. As of April 1, 2010, Sony disabled the ability to install Linux "due to security concerns" starting with firmware version 3.21.
- In 2008 many netbook models were shipped with Linux installed, usually with a lightweight distribution, such as Xandros or Linpus, to reduce resource consumption on their limited resources.
- Through 2007 and 2008 Linux distributions with an emphasis on ease of use such as Ubuntu became increasingly popular as home desktop operating systems, with some OEMs, such as Dell, offering models with Ubuntu or other Linux distributions on desktop systems.

Business

Linux is also used in some corporate environments as the desktop platform for its employees, with commercially available solutions including Red Hat Enterprise Linux, SUSE Linux Enterprise Desktop, and Linspire.

- Burlington Coat Factory has used Linux exclusively since 1999.
- Ernie Ball, known for its famous *Super Slinky* guitar strings, has used Linux as its desktop operating system since 2000.
- Novell is undergoing a migration from Windows to Linux. Of its 5500 employees, 50% were successfully migrated as of April, 2006. This was expected to rise to 80% by November.
- Wotif, the Australian hotel booking website, migrated from Windows to Linux servers to keep up with the growth of its business.
- Union Bank of California announced in January 2007 that it would standardize its IT infrastructure on Red Hat Enterprise Linux in order to lower costs.
- Peugeot, the European car maker, announced plans to deploy up to 20,000 copies of Novell's Linux desktop, SUSE Linux Enterprise Desktop, and 2,500 copies of SUSE Linux Enterprise Server, in 2007.
- Mindbridge, a software company, announced in September, 2007 that it had migrated a large number of Windows servers onto a smaller number of Linux servers and a few BSD servers. It claims to have saved "bunches of money."
- Virgin America, the low cost U.S. airline, uses Linux to power its in-flight entertainment system, RED.
- Amazon.com, the US based mail-order retailer, uses Linux "in nearly every corner of its business".
- Google uses a version of Ubuntu internally nicknamed Goobuntu.

- IBM does extensive development work for Linux and also uses it on desktops and servers internally. The company also created a TV advertising campaign: *IBM supports Linux 100%.*
- DreamWorks Animation adopted the use of Linux since 2001, and uses more than 1,000 Linux desktops and more than 3,000 Linux servers.
- The Chicago Mercantile Exchange employs an all-Linux computing infrastructure and has used it to process over a quadrillion dollars worth of financial transactions
- The Chi-X pan-European equity exchange runs its MarketPrizm trading platform software on Linux.
- The London Stock Exchange uses the Linux based MillenniumIT Millennium Exchange software for its trading platform and predicts that moving to Linux from Windows will give it an annual cost savings of at least £10 million ($14.7 million) from 2011-12
- The New York Stock Exchange uses Linux to run its trading applications.
- American electronic music composer Kim Cascone migrated from Apple Mac to Ubuntu for his music studio, performance use and administration in 2009.
- McDonald's uses the Ubuntu operating system at McCafes.
- *Laughing Boy Records* under the direction of owner Geoff Beasley switched from doing audio recording on Windows to Linux in 2004 as a result of Windows spyware problems.
- Nav Canada's new Internet Flight Planning System for roll-out in 2011, is written in Python and runs on Red Hat Linux.
- Electrolux Frigidaire Infinity i-kitchen is a "smart appliance" refrigerator that uses a Linux operating system, running on an embedded 400MHz Freescale i.MX25 processor with 128 MB of RAM and a 480×800 touch panel.

Scientific institutions

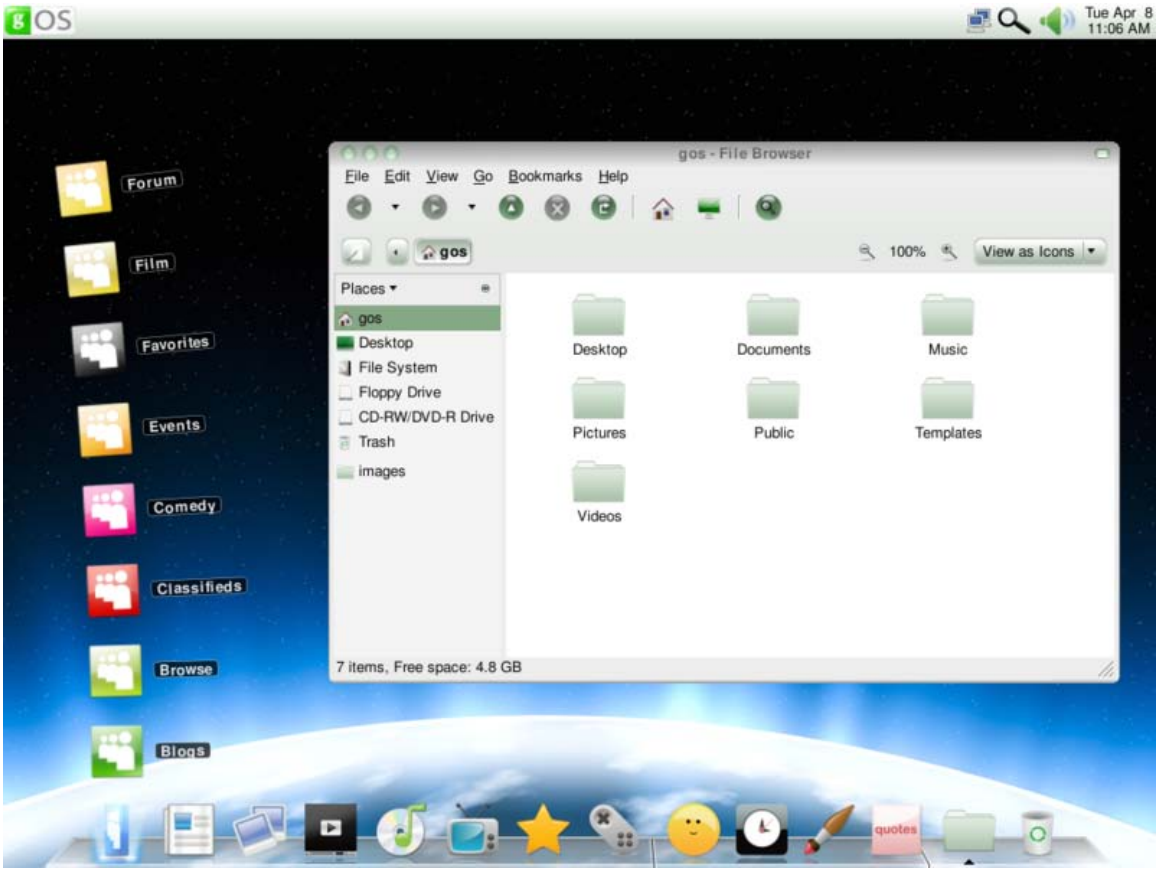
The IBM Roadrunner, the world's third fastest supercomputer operated by the US National Nuclear Security Administration, uses Red Hat Enterprise Linux and Fedora as its operating systems.

- CERN uses Scientific Linux in all its work, including running the Large Hadron Collider and for its 20,000 internal servers.
- Canada's largest super computer, the IBM iDataPlex cluster computer at the University of Toronto uses Linux as its operating system.
- The Internet Archive uses hundreds of x86 servers to catalogue the internet, all of them running Linux.
- ASV Roboat autonomous robotic sailboat runs on Linux
- Tianhe-I, the worlds fastest super computer as of October 2010, located at the National Centre for Supercomputing in Tianjin, China runs Linux.
- FermiLab's Dark Energy Camera and associated 4m telescope, part of the The Dark Energy Survey will be controlled by and store all its data on computers running Linux.

Types of hardware platforms

Linux is used on desktop computers, servers and supercomputers, as well as a wide range of devices.

Desktop computers



gOS desktop

Measuring desktop adoption

Because Linux desktop distributions are not usually distributed by retail sale like other operating systems, there are no sales numbers that indicate the number of users. One downloaded file may be used to create many CDs and each CD may be used to install the operating system on multiple computers. Due to these factors many estimates of current Linux desktop often rely on webpage hits by computers identifying themselves as running Linux. The use of these statistics has been criticized as unreliable and as underestimating Linux use.

Using webpage hits as a measure, until 2008 Linux accounted for only about 1% of desktop market share, while Microsoft Windows operating systems held more than 90%. This might have been because Linux was not seen at that time as a direct replacement for Windows.

According to *W3Counter* webpage hits the Linux desktop market share increased 62% from 1.32% to 2.13% between mid 2007 and the beginning of 2009, while Windows use fell from 95.52% to 88.77% in the same period, a drop of 7%.

The *Linux Counter* uses an alternate method of estimating adoption, asking users to register and then using a mathematical model to estimate the total number of desktop users. In March 2009 this method estimated 29 million Linux users.

In April 2009 Aaron Seigo of KDE indicated that most web-page counter methods produce Linux adoption numbers that are far too low given the system's extensive penetration into non-North American markets, especially China. He stated that the North American-based web-measurement methods produce high Windows numbers and ignore the widespread use of Linux in other parts of the world. In estimating true worldwide desktop adoption and accounting for the Windows-distorted environment in the USA and Canada he indicated that at least 8% of the world desktops run Linux distributions and possibly as high as 10-12% and that the numbers are rising quickly. Other commentators have echoed this same belief, noting that competitors are expending a lot of effort to discredit Linux, which is incongruent with a tiny market share:

I don't believe that the desktop Linux market share is barely 1%. I think it is a lot higher. I have no good data to share; I base my assessment on experience and knowing the industry. There is something else that is even more persuasive, and that is how Microsoft behaves. If Linux is so insignificant, why do they pay so much attention to it?
—Carla Schroder, Linux Today

In May 2009 Preston Gralla, contributing editor to Computerworld.com, in reacting to the Net Applications web hit numbers showing that Linux use was over 1% said that "Linux will never become an important desktop or notebook operating system". He reasons that the upsurge in Linux desktop use recently seen is due to Linux netbooks, a trend he says is already diminishing and will be further eroded when Windows 7 becomes available. He concludes: "As a desktop operating system, Linux isn't important enough to think about. For servers, it's top-notch, but you likely won't use it on your desktop -- even though it did finally manage to crack the 1% barrier after 18 years".

In 2009 Microsoft CEO Steve Ballmer indicated that Linux had a greater desktop market share than Mac, stating that in recent years Linux had "certainly increased its share somewhat". Just under a third of all Dell netbook sales in 2009 had Linux installed. By 2010, as a total of retail sales Linux represented 8% of desktop operating systems.

Caitlyn Martin, researching retail market numbers in the summer of 2010 also concluded that the traditional numbers mentioned for Linux desktop adoption were far too low:

It seems like almost every day someone in the tech press or someone commenting in a technical forum will claim that Linux adoption on the desktop (including laptops) is insignificant. The number that is thrown around is 1%. These claims are even repeated by some who advocate for Linux adoption. Both the idea that Linux market share on the desktop is insignificant and the 1% figure are simply false and have been for many years...Where does the 1% number come from? There are two sources: very old data and web counters. The problem with using web counters to try and ascertain market share is

that they generally only include websites that have paid to be counted. That pretty much guarantees that Windows will be overcounted.
—Caitlyn Martin

Reasons for adoption

Reasons to change from other operating systems to Linux include better system stability, virus, trojan, adware and spyware protection, low or no cost, that most distributions come complete with application software and hardware drivers, simplified updates for all installed software, free software licencing, availability of application repositories and access to the source code. Although Linux fragments files, it fragments less than some other popular operating systems. Linux desktop distributions also offer multiple desktop workspaces, greater customization, free and unlimited support through forums and an operating system that doesn't slow down over time. Environmental reasons are also cited, as Linux operating systems usually do not come in boxes and other retail packaging, but are downloaded via the internet. The lower system specifications also mean that older hardware can be kept in use instead of being recycled or discarded. Linux distributions also get security vulnerabilities patched much more quickly than non-free operating systems and improvements in Linux have been occurring at a faster rate than Windows has been improving.

Investments have been made to improve desktop Linux usability since 2007. A report in The Economist in December 2007 said: "Linux has swiftly become popular in small businesses and the home. That’s largely the doing of Gutsy Gibbon, the code-name for the Ubuntu 7.10 from Canonical. Along with distributions such as Linspire, Mint, Xandros, OpenSUSE and gOS, Ubuntu (and its siblings Kubuntu, Edubuntu and Xubuntu) has smoothed most of Linux’s geeky edges while polishing it for the desktop. No question, Gutsy Gibbon is the sleekest, best integrated and most user-friendly Linux distribution yet. It’s now simpler to set up and configure than Windows."

Indian bulk computer purchaser the Electronics Corporation of Tamil Nadu (ELCOT) started recommending only Linux in June 2008. Following testing they stated: "ELCOT has been using SUSE Linux and Ubuntu Linux operating systems on desktop and laptop computers numbering over 2,000 during the past two years and found them far superior as compared to other operating systems, notably the Microsoft Operating System."

In many developing nations, such as China, where, due to widespread software piracy, Microsoft Windows can be easily obtained for free, Linux distributions are gaining a high level of adoption. In these countries there is essentially no cost barrier to obtaining proprietary operating systems, but users are adopting Linux based on its merit, rather than on price.

In January 2001, Microsoft CEO Bill Gates explained the attraction of adopting Linux in an internal memo that was released in the *Comes vs Microsoft* case. He said:

Our most potent Operating System competitor is Linux and the phenomena around Open Source and free software. The same phenomena fuels competitors to all of our products. The ease of picking up Linux to learn it or to modify some piece of it is very attractive. The academic community, start up companies, foreign governments and many other constituencies are putting their best work into Linux.

Barriers to adoption

The greatest barrier to Linux desktop adoption is probably that few desktop PCs come with it from the factory. A.Y. Siu asserted in 2006 that most people use Windows simply because most PCs come with Windows pre-installed; they didn't choose it. Linux has much lower market penetration because in most cases users have to install it themselves, a task that is beyond the capabilities of the average PC user: "Most users won't even use Windows restore CDs, let alone install Windows from scratch. Why would they install an unfamiliar operating system on their computers?"

TechRepublic writer Jack Wallen expands on this barrier, saying in August 2008:

Why would anyone choose Windows over Linux?...In my seriously biased opinion, I think this question is answered with a simple conspiracy theory: Microsoft is doing everything it can to keep the public blind to Linux. Think about it? Remember the whole Wintel conspiracy where MS and Intel played off of each other to continue their strangle-hold monopoly in the PC industry? That era played a huge part in the blinding of consumers. Top that with the business practices MS forces upon big box shops to insure *(sic)* their operating system is sold on nearly every PC sold and you can see that conspiracy is more of a reality than one might think.

In an openSUSE survey conducted in 2007, 69.5% of respondents said they dual booted a Microsoft Windows operating system in addition to a Linux operating system. In early 2007 Bill Whyman, an analyst at Precursor Advisors, noted that "there still isn't a compelling alternative to the Microsoft infrastructure on the desktop."

Application support, the quality of peripheral support, and end user support were at one time seen as the biggest obstacles to desktop Linux adoption. According to a 2006 survey by The Linux Foundation, these factors were seen as a "major obstacle" for 56%, 49%, and 33% of respondents respectively at that time.

Application support

The November 2006 *Desktop Linux Client Survey* identified the foremost barrier for deploying Linux desktops was that users were accustomed to Windows applications which had not been ported to Linux and which they "just can't live without". These included Microsoft Office, Adobe Photoshop, Autodesk AutoCAD, Microsoft Project, Visio and Intuit QuickBooks. In a DesktopLinux.com survey conducted in 2007, 72% of respondents said they used ways to run Windows applications on Linux.

51% of respondents to the 2006 Linux Foundation survey believed that cross-distribution Linux desktop standards should be the top priority for the Linux desktop community, highlighting the fact that the fragmented Linux market is preventing application vendors from developing, distributing and supporting the operating system. In May 2008, Gartner predicted that "version control and incompatibilities will continue to plague open-source OSs and associated middleware" in the 2013 timeframe.

By 2008 the design of Linux applications and the porting of Windows and Apple applications had progressed to the point where it was difficult to find an application that did not have an equivalent for Linux, providing adequate or better capabilities.

An example of application progress can be seen comparing the main productivity suite for Linux, OpenOffice.org, to Microsoft Office. With the release of OpenOffice.org 3.0 in October 2008 Ars Technica assessed the two:

Although OpenOffice.org has not yet reached full parity with Microsoft Office, it is maturing at a rapid pace and is already capable of meeting the basic needs of many average computer users. It is an ideal choice for schools and is an increasingly viable choice for small businesses and home users that don't rely on the more advanced capabilities of Microsoft's office suite.

Peripheral support

In the past the availability and quality of open source device drivers were issues for Linux desktops. Particular areas which were lacking drivers included printers as well as wireless and audio cards. For example in early 2007, Dell did not sell specific hardware and software with Ubuntu 7.04 computers, including printers, projectors, Bluetooth keyboards and mice, TV tuners and remote controls, desktop modems and Blu-ray disc drives, due to incompatibilities at that time, as well as legal issues.

By 2008 most Linux hardware support and driver issues had been adequately addressed. In September 2008 Jack Wallen's assessment was:

Years ago, if you wanted to install Linux on a machine you had to make sure you hand-picked each piece of hardware or your installation would not work 100 percent...This is not so much the case now. You can grab a PC (or laptop) and most likely get one or more Linux distributions to install and work nearly 100 percent. But there are still some exceptions; for instance, hibernate/suspend remains a problem with many laptops, although it has come a long way.

End-user support

Some critics have stated that compared to Windows, Linux is lacking in end-user support. Linux has traditionally been seen as requiring much more technical expertise. Dell's website describes open source software as requiring intermediate or advanced knowledge to use. In September 2007, the founder of the Ubuntu project, Mark Shuttleworth, commented that "it would be reasonable to say that this is not ready for the mass market."

In October 2004 Chief Technical Officer of Adeptiva Linux, Stephan February, noted at that time that Linux was a very technical software product, and few people outside the technical community were able to support consumers. Windows users are able to rely on friends and family for help, but Linux users generally use discussion boards, which can be uncomfortable for consumers.

In 2005 Dominic Humphries summarized the difference in user tech support:

Windows users are more or less in a customer-supplier relationship: They pay for software, for warranties, for support, and so on. They expect software to have a certain level of usability. They are therefore used to having rights with their software: They have paid for technical support and have every right to demand that they receive it. They are also used to dealing with entities rather than people: Their contracts are with a company, not with a person.

Linux users are in more of a community. They don't have to buy the software, they don't have to pay for technical support. They download software for free & use Instant Messaging and web-based forums to get help. They deal with people, not corporations.

More recently critics have found that the Linux user support model, using community-based forum support, has greatly improved. In 2008 Jack Wallen stated:

With Linux, you have the support of a huge community via forums, online search, and plenty of dedicated websites. And of course, if you feel the need, you can purchase support contracts from some of the bigger Linux companies (Red Hat and Novell, for instance).

However, when you use the peer support inherent in Linux, you do take a chance with time. You could have an issue with something, send out email to a mailing list or post on a forum, and within 10 minutes be flooded with suggestions. Or these suggestions could take hours or days to come in. It seems all up to chance sometimes.

Yet generally speaking, most problems with Linux have been encountered and documented, so the chances are good you'll find your solution fairly quickly.

In addressing the question of user support, Manu Cornet said:

One of the great assets of the Open Source community (and Linux in particular), is that it's a real community. Users and developers really are out there, on web forums, on mailing lists, on IRC channels, helping out new users. They're all happy to see more and more people switch to Linux, and they're happy to help them get a grip on their new system...you'll find literally thousands of places where nice people will answer you and walk you out of your problem most of the time

Other factors

Linux's credibility has also been under attack at times, but as Ron Miller of LinuxPlanet points out:

...the fact that Linux is being criticized is probably a good thing.

First of all, it shows that Linux is making headway in the enterprise and beginning to have an impact on competitors and they are reacting to that. Secondly, it's healthy to take a long look at any solution and analyze its strengths and weaknesses and the economic ramifications of one choice over another.

Ultimately, consumers and decision makers need to look carefully at the data including the sources of the data and the criticism and decide if Linux is the right decision, but as more people choose Linux and it finds its place in the market, it is bound to wear a target. That's simply the price you pay for success in the marketplace.

There is continuing debate about the total cost of ownership of Linux, with Gartner warning in 2005 that the costs of migration may exceed the cost benefits of Linux. Gartner reiterated the warning in 2008, predicting that "by 2013, a majority of Linux deployments will have no real software total cost of ownership (TCO) advantage over other operating systems." Organizations that have moved to Linux have disagreed with these warnings. Sterling Ball, CEO of Ernie Ball, the world's leading maker of premium guitar strings and a 2003 Linux adopter, said of total cost of ownership arguments: "I think that's propaganda...What about the cost of dealing with a virus? We don't have 'em...There's no doubt that what I'm doing is cheaper to operate. The analyst guys can say whatever they want."

In the SCO-Linux controversies, the SCO Group had alleged that UNIX source code donated by IBM was illegally incorporated into Linux. The threat that SCO might be able to legally assert ownership of Linux initially caused some potential Linux adopters to delay that move. The court cases bankrupted SCO in 2007 after they lost their four-year court battle over the ownership of the UNIX copyrights. SCO's case had hinged on showing that Linux included intellectual property that had been misappropriated from UNIX, but the case failed when the court discovered that Novell and not SCO, was the rightful owner of the copyrights. During the legal process, it was revealed that SCO's claims about Linux were fraudulent and that SCO's internal source code audits had showed no evidence of infringement.

A rival operating system vendor, Green Hills Software, has called the open source paradigm of Linux "fundamentally insecure".

The US Army does not agree that Linux is a security problem. Brigadier General Nick Justice, the Deputy Program Officer for the Army's Program Executive Office, Command, Control and Communications Tactical (PEO C3T) said in April 2007:

Our job is to provide accurate and timely information to the soldier in the field so they can perform their mission. Open source software is part of the integrated network fabric

which connects and enables our command and control system to work effectively, as people's lives depend on it.

When we rolled into Baghdad, we did it using open source. It may come as a surprise to many of you, but the U.S. Army is "the" single largest install base for Red Hat Linux. I'm their largest customer.

Servers

Servers designed for Linux

Linux became popular in the Internet server market particularly due to the LAMP software bundle. In September 2008 Steve Ballmer (Microsoft CEO) claimed 60% of servers run Linux and 40% run Windows Server.

Supercomputers

Linux is the most popular operating system among supercomputers due to its superior performance, flexibility, speed and lower costs. In November 2008 Linux held an 87.8 percent share of the world's top 500 supercomputers.

As of June 2010 the operating systems used on the world's top 500 supercomputers were:

OS	Share
Linux	91.0%
Unix	4.4%
Hybrid Unix/Linux	3.4%
Windows HPC	1.0%
BSD	0.2%

In January 2010 Weiwu Hu, chief architect of the Loongson family of CPUs at the Institute of Computing Technology, which is part of the Chinese Academy of Sciences, confirmed that the new Dawning 6000 supercomputer will use Chinese-made Loongson processors and will run Linux as its operating system. The most recent supercomputer the organization built, the Dawning 5000a, which was first run in 2008, utilized AMD chips and ran Windows HPC Server 2008.

Devices

Linux is often used in various single- or multi-purpose computer appliances and embedded systems. For instance the Palm webOS, Palm, Inc.'s flagship mobile operating system runs on the Linux kernel.

Advocacy

Many organizations advocate for Linux adoption. The foremost of these is the Linux Foundation which hosts and sponsors the key kernel developers, manages the Linux trademark, manages the Open Source Developer Travel Fund, provides legal-aid to open source developers and companies through the Linux Legal Defense Fund, sponsors kernel.org and also hosts the Patent Commons Project.

The International Free and Open Source Software Foundation (iFOSSF) is a nonprofit organization based in Michigan, USA dedicated to accelerating and promoting the adoption of FOSS worldwide through research and civil society partnership networks.

The Open Invention Network was formed to protect vendors and customers from patent royalty fees while using OSS.

Other advocates for Linux include:

- IBM through their Linux Marketing Strategy
- Linux User Groups
- Asian Open Source Centre (AsiaOSC)
- The Brazilian government, under president Luis Inácio Lula da Silva
- Livre Brasil, a Brazilian organization promoting Linux adoption in schools, public department's, commerce, industry and personal desktops.
- FOSSFP: Free and Open Source Software Foundation of Pakistan.

Chapter 5

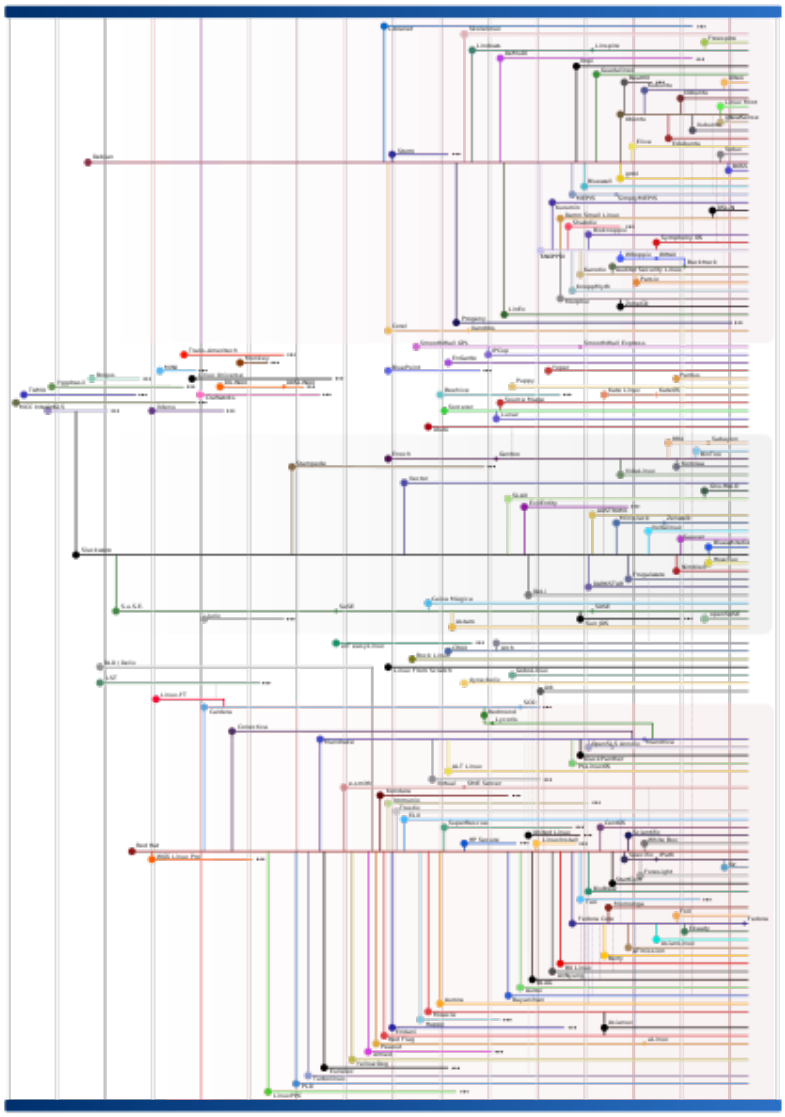
Linux Distribution

A **Linux distribution** is a member of the family of Unix-like operating systems built on top of the Linux kernel. Such distributions (often called *distros* for short) consist of a large collection of software applications such as word processors, spreadsheets, media players, and database applications. The operating system will consist of the Linux kernel and, usually, a set of libraries and utilities from the GNU project, with graphics support from the X Window System. Distributions optimized for size may not contain X and tend to use more compact alternatives to the GNU utilities, such as Busybox, uClibc, or dietlibc. There are currently over six hundred Linux distributions. Over three hundred of those are in active development, constantly being revised and improved.

Because most of the kernel and supporting packages are free and open source software, Linux distributions have taken a wide variety of forms — from fully featured desktop, server, laptop, netbook, Mobile Phone, and Tablet operating systems as well as minimal environments (typically for use in embedded systems or for booting from a floppy disk). Aside from certain custom software (such as installers and configuration tools), a distribution is most simply described as a particular assortment of applications installed on top of a set of libraries married with a version of the kernel, such that its "out-of-the-box" capabilities meet most of the needs of its particular end-user base.

One can distinguish between commercially-backed distributions, such as Fedora (Red Hat), openSUSE (Novell), Ubuntu (Canonical Ltd.), and Mandriva Linux (Mandriva), and entirely community-driven distributions, such as Debian and Gentoo, though there are other distributions that are driven neither by a corporation nor a community, perhaps most famously Slackware.

History



GNU/Linux Distro Timeline, timeline representing the development of various Linux distributions.

Before the first Linux distributions, a would-be Linux user was required to be something of a Unix expert, needing to know not only what libraries and executables were required to successfully get the system to boot and run, but also important details concerning configuration and placement of files in the system.

Linux distributions began to appear very soon after the Linux kernel was first used by individuals other than the original Linux programmers who were more interested in developing the operating system than developing application programs, the user interface, or convenient packaging.

Early distributions included:

- H J Lu's "Boot-root" a two disk pair with the kernel and the absolute minimal tools to get started
- MCC Interim Linux, which was made available to the public for download on the FTP server of University of Manchester in February 1992
- TAMU, created by individuals at Texas A&M University about the same time
- SLS (Softlanding Linux System)
- Yggdrasil Linux/GNU/X, the first CD-ROM based Linux distribution

SLS was not well maintained, so Patrick Volkerding released a distribution based on SLS, which he called Slackware, released in 1993. This is the oldest distribution still in active development.

Users were attracted to Linux distributions as alternatives to the DOS and Microsoft Windows operating systems on the PC, Mac OS on the Apple Macintosh, and proprietary versions of Unix. Most early adopters were familiar with Unix from work or school. They embraced Linux for its stability, low (if any) cost, and availability of the source code for most or all of the software included.

Originally, the distributions were simply a convenience, but today, they have become the usual choice even for Unix or Linux experts. To date, Linux has proven more popular in the server market, primarily for Web and database servers than in the desktop market.

Components

A typical desktop Linux distribution comprises a Linux kernel, GNU tools and libraries, additional software, documentation, a window system, window manager, and a desktop environment. Most of the included software is free software/open-source software which is distributed by its maintainers both as compiled binaries and in source code form, allowing users to modify and compile the original source code if they wish. Other software included with some distributions may be proprietary and may not be available in source code form.

Many distributions provide an installation system akin to that provided with other modern operating systems. Some distributions like Gentoo Linux, T2, and Linux From Scratch include only binaries of a basic kernel, compilation tools, and an installer; the installer compiles all the requested software for the specific microarchitecture of the user's machine, using these tools and the provided source code.

Package management

Distributions are normally segmented into **packages**. Each package contains a specific application or service. Examples of packages are a library for handling the PNG image format, a collection of fonts or a web browser.

The package is typically provided as compiled code, with installation and removal of packages handled by a package management system (PMS) rather than a simple file archiver. Each package intended for such a PMS contains meta-information such as a package description, version, and "dependencies". The package management system can evaluate this meta-information to allow package searches, to perform an automatic upgrade to a newer version, to check that all dependencies of a package are fulfilled, and/or to fulfill them automatically.

Although Linux distributions typically contain much more software than proprietary operating systems, it is normal for local administrators to also install software not included in the distribution. An example would be a newer version of a software application than that supplied with a distribution, or an alternative to that chosen by the distribution (*e.g.*, KDE rather than GNOME or vice versa for the user interface layer). If the additional software is distributed in source-only form, this approach requires local compilation. However, if additional software is locally added, the 'state' of the local system may fall out of synchronization with the state of the package manager's database. If so, the local administrator will be required to take additional measures to ensure the entire system is kept up to date. The package manager may no longer be able to do so automatically.

Most distributions install packages, including the kernel and other core operating system components, in a predetermined configuration. Few now require or even permit configuration adjustments at first install time. This makes installation less daunting, particularly for new users, but is not always acceptable. For specific requirements, much software must be carefully configured to be useful, to work correctly with other software, or to be secure, and local administrators are often obliged to spend time reviewing and reconfiguring assorted software.

Some distributions go to considerable lengths to specifically adjust and customize most or all of the software included in the distribution. Not all do so. Some distributions provide configuration tools to assist in this process.

By replacing *everything* provided in a distribution, an administrator may reach a "distribution-less" state: everything was retrieved, compiled, configured, and installed locally. It is possible to build such a system from scratch, avoiding a distribution altogether. One needs a way to generate the first binaries until the system is *self-hosting*. This can be done via compilation on another system capable of building binaries for the intended target (possibly by cross-compilation).

Types and trends

Broadly, Linux distributions may be:

- Commercial or non-commercial;
- Designed for enterprise users, power users, or for home users;

- Supported on multiple types of hardware, or platform-specific, even to the extent of certification by the platform vendor;
- Designed for servers, desktops, or embedded devices;
- General purpose or highly specialized toward specific machine functionalities (e.g. firewalls, network routers, and computer clusters);
- Targeted at specific user groups, for example through language internationalization and localization, or through inclusion of many music production or scientific computing packages;
- Built primarily for security, usability, portability, or comprehensiveness.

The diversity of Linux distributions is due to technical, organizational, and philosophical variation among vendors and users. The permissive licensing of free software means that any user with sufficient knowledge and interest can customize an existing distribution or design one to suit his or her own needs.

Installation-free distributions (Live CDs)

A Live Distro or Live CD is a Linux distribution that can be booted from a compact disc or other removable medium (such as a DVD or USB flash drive) instead of the conventional hard drive. Some minimal distributions such as tomsrtbt can be run directly from as little as one floppy disk without needing to change the system's hard drive contents.

When the operating system is booted from a read-only device such as a CD or DVD, if user data needs to be retained between sessions, it cannot be stored on the boot device but must be written to some other media such as a USB flash drive or an installed hard drive. Temporary operating system data is usually kept solely in RAM.

The portability of installation-free distributions makes them advantageous for applications such as demonstrations, borrowing someone else's computer, rescue operations, or as installation media for a standard distribution. Many popular distributions come in both "Live" and conventional forms (the conventional form being a network or removable media image which is intended to be used for installation only). This includes SUSE, Ubuntu, Linux Mint, MEPIS, Sidux, and Fedora. Some distributions, such as Knoppix, Devil-Linux, SuperGamer, and dyne:bolic are designed primarily for Live CD, Live DVD, or USB flash drive use.

Examples

Popular distributions

Well-known Linux distributions include:

- Arch Linux, a minimalist distribution maintained by a community volunteer and primarily based on binary packages in the tar.gz and tar.xz format.

- Debian, a non-commercial distribution maintained by a volunteer developer community with a strong commitment to free software principles
    - Knoppix, the first Live CD distribution to run completely from removable media without installation to a hard disk, derived from Debian
    - Ubuntu, a popular desktop and server distribution derived from Debian, maintained by Canonical Ltd..
        - Kubuntu, the KDE version of Ubuntu.
        - Linux Mint, a distribution based on and compatible with Ubuntu.
        - Xubuntu, is the Xfce version of the popular desktop distro Ubuntu. Commonly used by Linux users that wish to have the function of a bigger distro such as Ubuntu or openSuse with the speed of a smaller distro.
- Fedora, a community distribution sponsored by Red Hat
    - CentOS, a distribution derived from the same sources used by Red Hat, maintained by a dedicated volunteer community of developers with both 100% Red Hat-compatible versions and an upgraded version that is not always 100% upstream compatible
    - Mandriva, a Red Hat derivative popular in France and Brazil, today maintained by the French company of the same name.
        - PCLinuxOS, a derivative of Mandriva, grew from a group of packages into a community-spawned desktop distribution.
    - Red Hat Enterprise Linux, which is a derivative of Fedora, maintained and commercially supported by Red Hat.
        - Oracle Enterprise Linux, which is a derivative of Red Hat Enterprise Linux, maintained and commercially supported by Oracle.
- Gentoo, a distribution targeted at power users, known for its FreeBSD Ports-like automated system for compiling applications from source code
- openSUSE a community distribution mainly sponsored by Novell.
    - SUSE Linux Enterprise, derived from openSUSE, maintained and commercially supported by Novell.
- Pardus, Turkey's national operating system developed by TÜBİTAK/UEKAE.
- Slackware, one of the first Linux distributions, founded in 1993, and since then actively maintained by Patrick J. Volkerding.

DistroWatch attempts to include every known distribution of Linux, whether currently active or not; it also maintains a ranking of distributions based on page views, as a measure of relative popularity.

Niche distributions

Other distributions are targeted at other specific niches, such as the tiny embedded router distribution OpenWrt, distributions for bioinformatics, the Ubuntu project to create Edubuntu for educational users, and KnoppMyth, which wraps Knoppix around MythTV to ease building Linux-powered DVRs. Similarly, there is the XBMC Live distro which wraps Ubuntu around XBMC Media Center ease building Linux-powered HTPC (Home

Theater PC). Others target the Apple Macintosh platform, including mkLinux, Yellow Dog Linux, and Black Lab Linux. Karoshi is a server system based on PCLinuxOS and aimed at educational users. SuperGamer is one of the few distributions focused solely on gaming.

Interdistribution issues

The Free Standards Group is an organization formed by major software and hardware vendors that aims to improve interoperability between different distributions. Among their proposed standards are the Linux Standard Base, which defines a common ABI and packaging system for Linux, and the Filesystem Hierarchy Standard which recommends a standard filenaming chart, notably the basic directory names found on the root of the tree of any Linux filesystem. Those standards, however, see limited use, even among the distributions developed by members of the organization.

The diversity of Linux distributions means that not all software runs on all distributions, depending on what libraries and other system attributes are required. Packaged software and software repositories are usually specific to a particular distribution, though cross-installation is sometimes possible on closely related distributions.

Tools for choosing a distribution

There are tools available to help people select an appropriate distribution, such as several different versions of the Linux Distribution Chooser, and the universal package search tool *whohas*. There are easy ways to try out several Linux distributions before deciding on one: Multi Distro is a Live CD that contains nine space-saving distributions. Tools are available to make such CDs and DVDs, among them Nautopia.

Virtual machines such as VirtualBox and VMware Workstation permit booting of Live CD image files without actually burning a CD.

Details and interest rankings of Linux distributions are available on DistroWatch and a fairly comprehensive list of live CDs is available at livecdlist.com. Some websites such as OSDir.com and www.osvids.com offer screenshots and videos as a way to get a first impression of various distributions.

Workspot provides online Linux desktop demos using Virtual Network Computing (VNC).

Installation

There are many ways to install a Linux distribution. The most common method of installing Linux is by booting from a CD-ROM or DVD that contains the installation program and installable software. Such a CD can be burned from a downloaded ISO image, purchased alone for a low price, provided as a cover disk with a magazine, shipped for free by request, or obtained as part of a box set that may also include manuals

and additional commercial software. New users tend to begin by partitioning a hard drive in order to keep their previously-installed operating system. The Linux distribution can then be installed on its own separate partition without affecting previously saved data.

Early Linux distributions were installed using sets of floppies but this has been abandoned by all major distributions. Nowadays most distributions offer CD and DVD sets with the vital packages on the first disc and less important packages on later ones. They usually also allow installation over a network after booting from either a set of floppies or a CD with only a small amount of data on it.

Still another mode of installation is to install on a powerful computer to use as a servers and to use less powerful machines (perhaps without hard drives, with less memory and slower CPUs) as thin clients over the network. Clients can boot over the network from the server and display results and pass information to the server where all the applications run. The clients can be ordinary PCs with the addition of a network bootloader on a drive or network interface controller; hard disk space and processor power can be offloaded onto the client machine if desired. The cost savings achieved by using thin clients can be invested in greater computing power or storage on the server.

In a Live CD setup, the computer boots the entire operating system from CD without first installing it on the computer's hard disk. Some distributions have a Live CD *installer*, where the computer boots the operating system from the disk, and then proceeds to install it onto the computer's hard disk, providing a seamless transition from the OS running from the CD to the OS running from the hard disk.

Both servers and personal computers that come with Linux already installed are available from vendors including Hewlett-Packard and Dell.

On embedded devices, Linux is typically held in the device's firmware and may or may not be consumer-accessible.

Anaconda, one of the more popular installers, is used by Red Hat Enterprise Linux, Fedora and other distributions to simplify the installation process.

Installation via an existing operating system

Some distributions let the user install Linux on top of their current system, such as WinLinux or coLinux. Linux is installed to the Windows hard disk partition, and can be started from inside Windows itself.

Virtual machines (such as VirtualBox or VMware) also make it possible for Linux to be run inside another OS. The VM software simulates a separate computer onto which the Linux system is installed. After installation, the virtual machine can be booted as if it were an independent computer.

Various tools are also available to perform full dual-boot installations from existing platforms without a CD, most notably:

- The Wubi installer, which allows Windows users to download and install Ubuntu or its derivatives without the need for hard drive partitioning or an installation CD, allowing users to easily dual boot between either operating system on the same hard drive without losing data
- Win32-loader, which is in the process of being integrated in official Debian CDs/DVDs, and allows Windows users to install Debian without a CD, though it performs a network installation and thereby requires repartitioning
- UNetbootin, which allows Windows and Linux users to perform similar no-CD network installations for a wide variety of Linux distributions and additionally provides live USB creation support

Proprietary software

Some specific proprietary software products are not available in any form for Linux. This includes many popular computer games, although in recent years some game manufacturers have begun making their software available for Linux, such as Epic Games' *Unreal Tournament 2004*. Emulation and API-translation projects like Wine and Cedega make it possible to run non-Linux-based software on Linux systems, either by emulating a proprietary operating system or by translating proprietary API calls (e.g., calls to Microsoft's Win32 or DirectX APIs) into native Linux API calls. A virtual machine can also be used to run a proprietary OS (like Microsoft Windows) on top of Linux.

OEM contracts

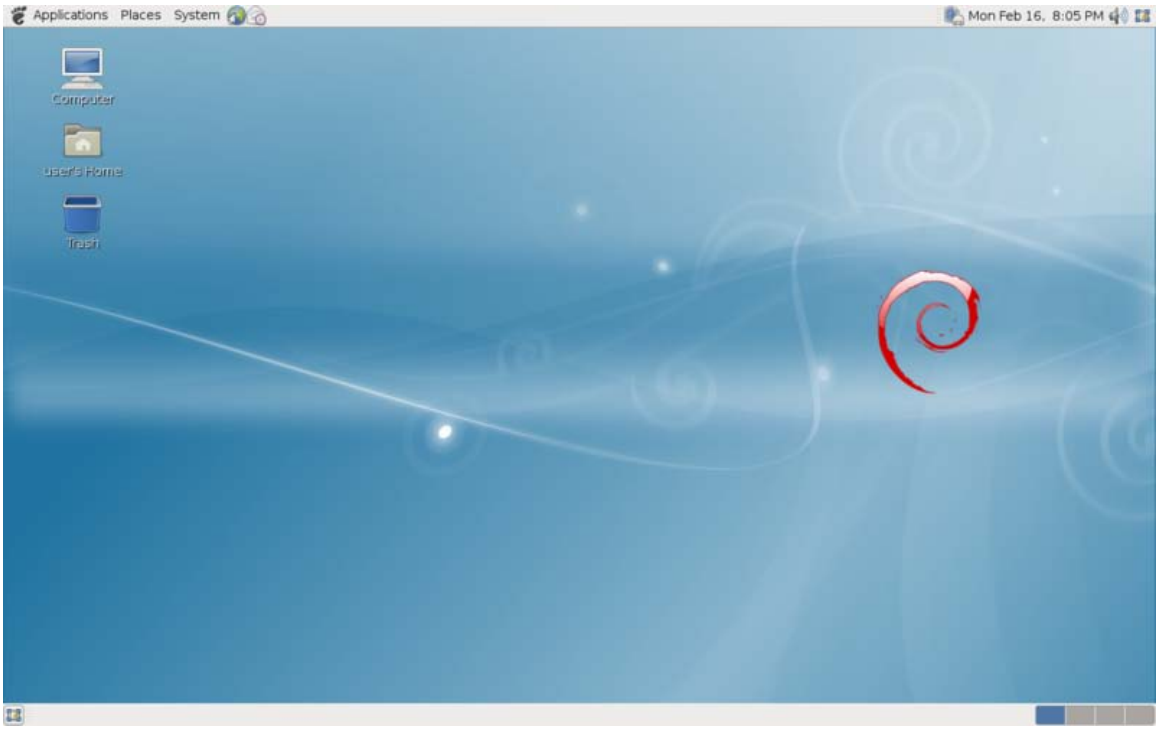
Computer hardware is usually sold with an operating system other than Linux already installed by the original equipment manufacturer (OEM). In the case of IBM PC compatibles the OS is usually Microsoft Windows; in the case of Apple Macintosh computers it has always been a version of Apple's OS, currently Mac OS X; Sun Microsystems sells SPARC hardware with Solaris installed; video game consoles such as the Xbox, PlayStation, and Wii each have their own proprietary OS. This limits Linux's market share: consumers are unaware that an alternative exists, they must make a conscious effort to use a different operating system, and they must either perform the actual installation themselves, or depend on support from a friend, relative, or computer professional.

However, it is possible to buy hardware with Linux already installed. Lenovo, Hewlett-Packard, Dell, Affordy, and System76 all sell general-purpose Linux laptops, and custom-order PC manufacturers will also build Linux systems (but possibly with the Windows key on the keyboard). Fixstars Solutions (formerly Terra Soft) sells Macintosh computers and PlayStation 3 consoles with Yellow Dog Linux installed.

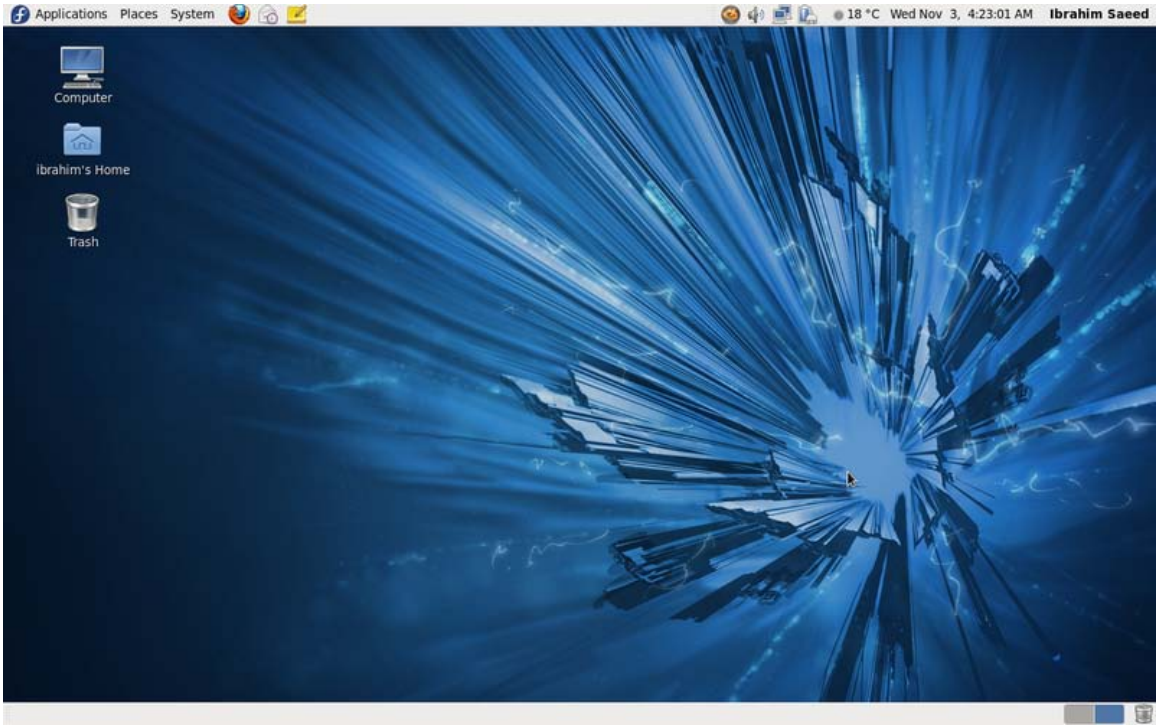
It is more common to find embedded devices sold with Linux as the default manufacturer-supported OS, including the Linksys NSLU2 NAS device, TiVo's line of personal video recorders, and Linux-based cellphones, PDAs, and portable music players.

Consumers also have the option of obtaining a refund for unused OEM operating system software. The end user license agreement (EULA) for Apple and Microsoft operating systems gives the consumer the opportunity to reject the license and obtain a refund. If requesting a refund directly from the manufacturer fails, it is also possible that a lawsuit in small claims court will work. On February 15, 1999, a group of Linux users in Orange County, California held a "Windows Refund Day" protest in an attempt to pressure Microsoft into issuing them refunds. In France, the Linuxfrench and AFUL organizations along with free software activist Roberto Di Cosmo started a "Windows Detax" movement, which led to a 2006 petition against "racketiciels" (translation: Racketware) and the DGCCRF branch of the French government filing several complaints against bundled software.

Screenshots of common distributions



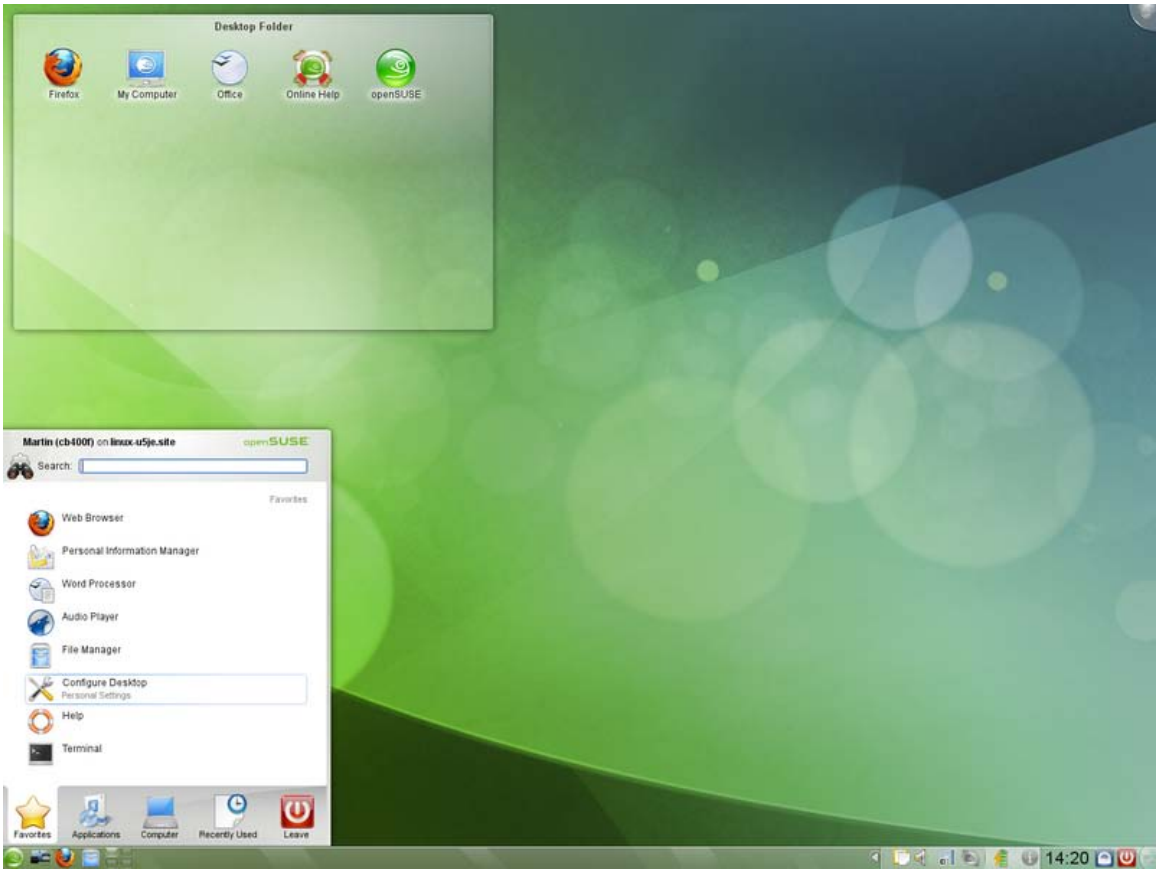
Debian 5.0 "Lenny"



Fedora 14

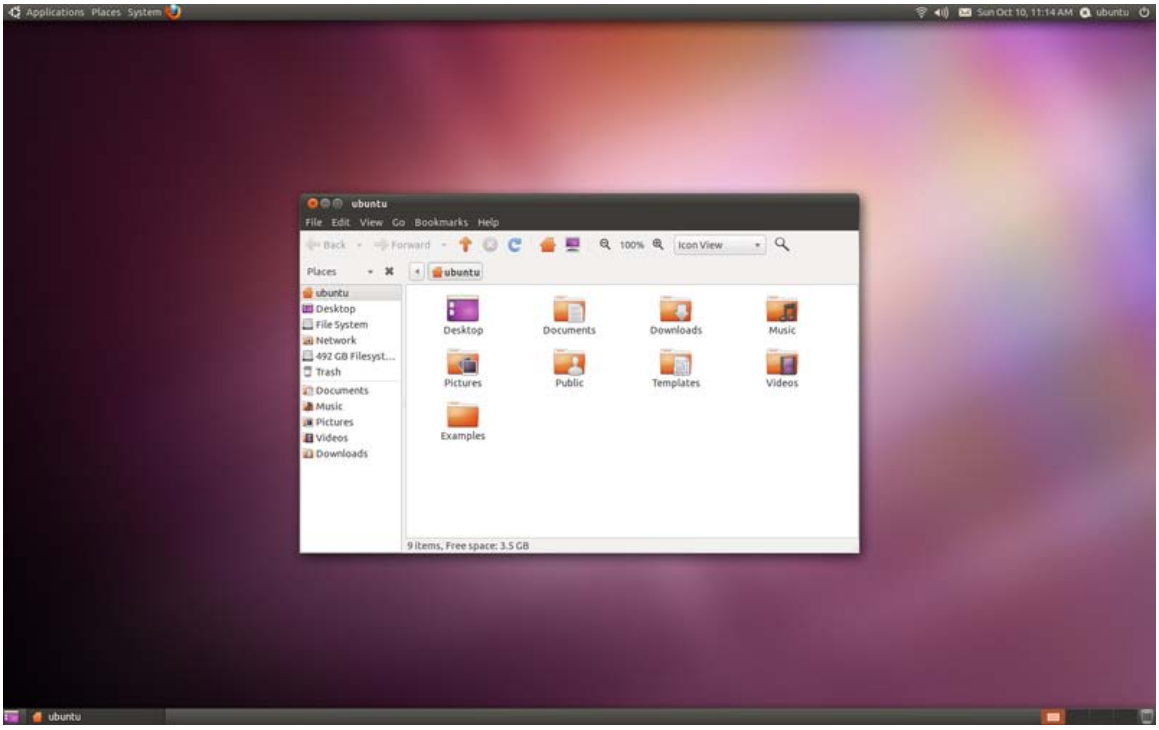
Gentoo Linux 10.1

Mandriva Linux 2010.0



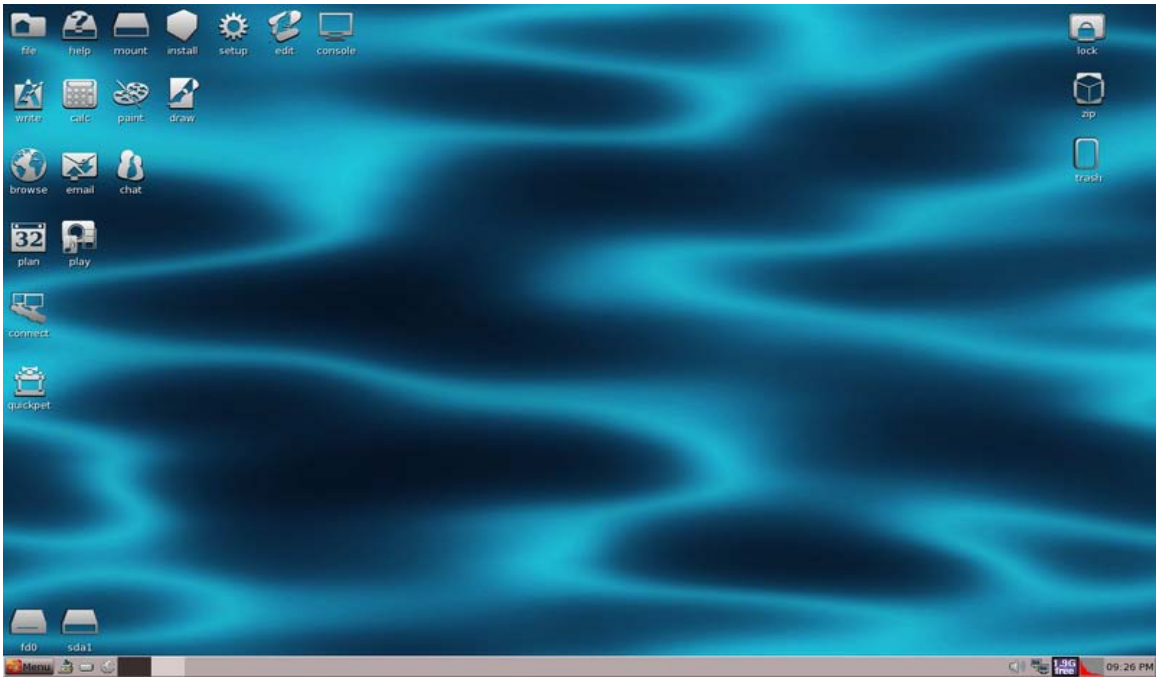
openSUSE 11.3

Slackware 13



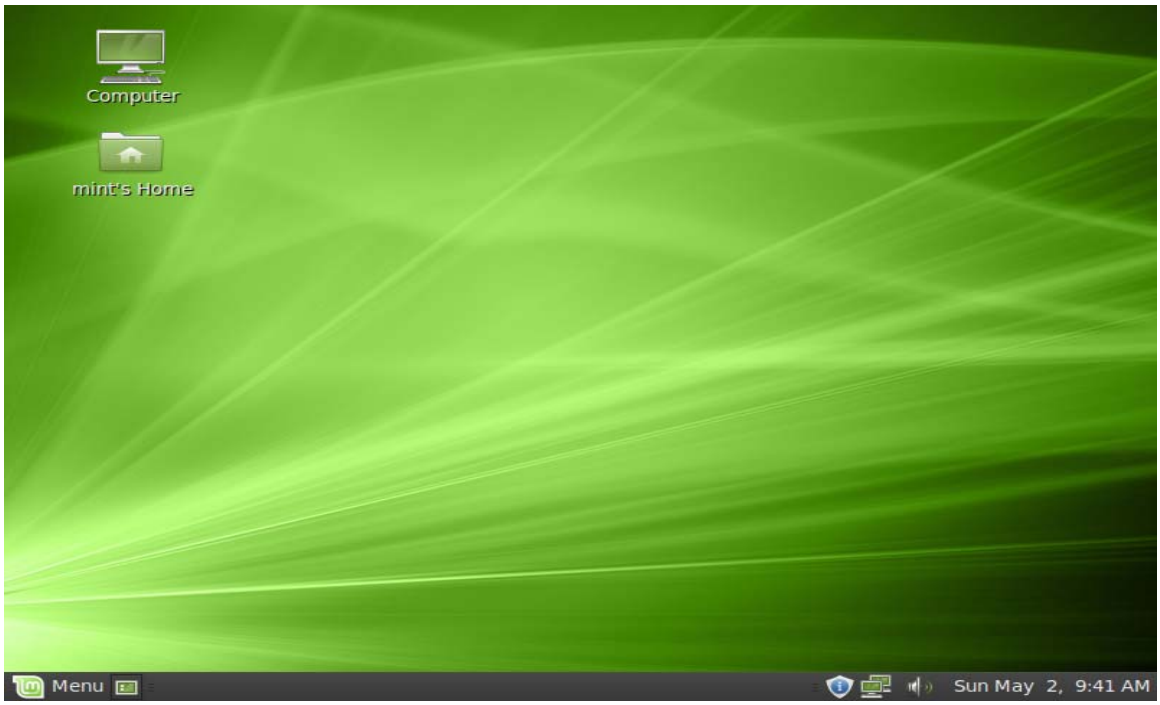
Ubuntu 10.10 "Maverick Meerkat"

Sabayon Linux 5.0

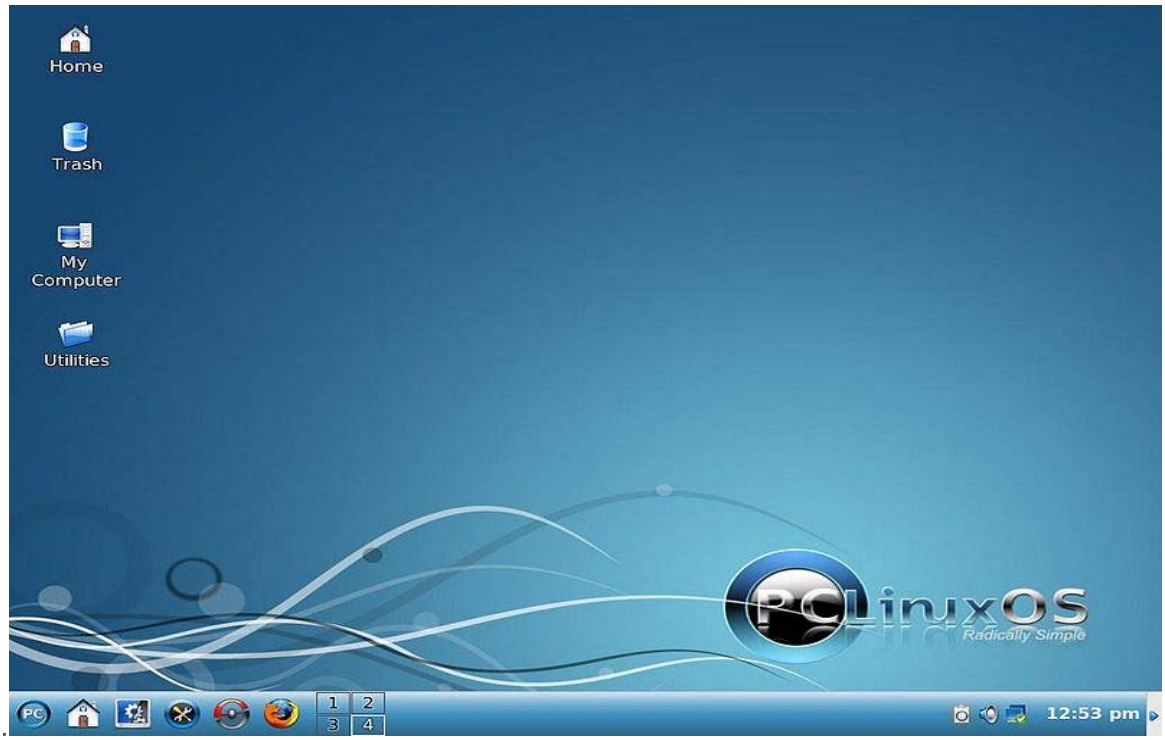


Puppy Linux 5.1

Slax 6.0.7



Linux Mint 9 "Isadora"



PCLinuxOS 2009.2

Chapter 6

Btrfs

Btrfs	
Developer	Oracle Corporation
Full name	Btrfs
Introduced	**Stable:** Yet to be released **Unstable:** v0.19, June 2009 (Linux)
Structures	
Directory contents	B-tree
File allocation	extents
Limits	
Max file size	16 EiB
Max number of files	2^{64}
Max filename length	255 bytes
Max volume size	16 EiB
Allowed characters in filenames	All bytes except NUL ('\0') and '/'
Features	
Dates recorded	modification (mtime), attribute modification (ctime), access (atime)
Date resolution	nanosecond
Attributes	POSIX, extended attributes

File system permissions	POSIX, ACL
Transparent compression	Yes
Transparent encryption	No (planned)
Supported operating systems	Linux

Btrfs (**B-tree file system**, variously pronounced "*Butter F S*", "*Better FS*", or "*B-tree F S*") is a GPL-licensed copy-on-write file system for Linux.

Btrfs is intended to address the lack of pooling, snapshots, checksums and integral multi-device spanning in Linux file systems, these features being crucial as the use of Linux scales upward into larger storage configurations common in the enterprise. Chris Mason, the principal author of the filesystem, has stated its goal was "to let Linux scale for the storage that will be available. Scaling is not just about addressing the storage but also means being able to administer and to manage it with a clean interface that lets people see what's being used and makes it more reliable."

Oracle has also begun work on CRFS (Coherent Remote File System), a network filesystem protocol intended to leverage the Btrfs architecture to gain higher performance than existing protocols (such as NFS and CIFS) and to expose Btrfs features such as snapshots to remote clients.

Btrfs 1.0 (with finalized on-disk format) was originally slated for a late 2008 release, but a stable release has not been made as of October 2010. It has, however, been accepted into the mainline kernel for testing as of 2.6.29-rc1. Several Linux distributions have also begun offering Btrfs as an experimental choice of root file system during installation, including openSUSE 11.3, SLES 11 SP1, Ubuntu 10.10, Red Hat Enterprise Linux 6, and MeeGo.

The principal developer of the ext3 and ext4 file systems, Theodore Ts'o, has stated that ext4 is a stop-gap and that Btrfs is the way forward, having "a number of the same design ideas that reiser3/4 had". Edward Shiskin, one of the Reiser4 developers now working for Redhat, got asked in Q2/2010 to look more detailled into Btrfs and judged that it has a broken design.

Features

Btrfs, thus far, has implemented:

- Online volume growth and shrinking
- Online block device addition and removal
- Online defragmentation
- Online balancing (movement of objects between block devices to balance load)
- Transparent compression (currently zlib)
- Subvolumes (separately-mountable filesystem roots)
- Snapshots (writeable, copy-on-write copies of subvolumes)
- File cloning (copy-on-write on individual files, or byte ranges thereof)
- Object-level (RAID1-like) mirroring, (RAID0-like) striping
- Checksums on data and metadata (currently CRC-32C)
- In-place conversion (with rollback) from ext3/4 to Btrfs
- File system seeding (Btrfs on read-only storage used as a copy-on-write backing for a writeable Btrfs)
- User-defined transactions
- Block discard support (reclaims space on some virtualized setups and improves wear leveling on SSDs by notifying the underlying device that storage is no longer in use)

Planned features include:

- Object-level (RAID5-like and RAID6-like) parity-based striping
- Online and offline filesystem check
- Incremental dumps
- Data deduplication

Btrfs, when complete, is expected to offer a feature set comparable to ZFS. Oracle's acquisition of ZFS as part of the Sun Microsystems merger has not changed their plans for developing Btrfs.

Cloning

Btrfs provides a *clone* operation which atomically creates a copy-on-write snapshot of a file, support for which was added to GNU coreutils 7.5. Cloning from byte ranges in one file to another is also supported, allowing large files to be more efficiently manipulated like standard rope data structures.

Subvolumes and snapshots

Subvolumes effectively allow a single instance of Btrfs to have multiple root directories, all using the file system as a pooled store. These roots are labeled and can be mounted separately by label. (There is always a "default" root which is mounted by default.) Subvolumes can also be nested inside other subvolumes, where they appear as subdirectories. Subvolumes are always created empty.

Snapshots are writeable copy-on-write clones of whole subvolumes; snapshotting itself is an atomic operation. Snapshots of individual subdirectories are not possible.

In-place ext3/4 conversion

Btrfs can warp to fit unusual spatial layouts because it has very little metadata anchored in fixed locations. The `btrfs-convert` tool exploits this property to perform an in-place conversion of any ext3 file system to Btrfs by nesting equivalent Btrfs metadata for all its files in the unallocated space of the ext3 file system. The result is a hybrid file system that, when mounted as a Btrfs, makes the ext3 files accessible in a writeable snapshot. The ext3 filesystem itself appears as a large sparse file which can be mounted as a read-only disk image. Deleting the image file commits the conversion; remounting as ext3 rolls back the conversion.

Transactions

Btrfs supports a very limited form of transaction without ACID semantics: rollback is not possible, only one transaction may run at a time and transactions are not atomic with respect to storage. They are analogous not to transactions in databases, but to the I/O "transactions" in ext3's JBD layer.

An ioctl interface, however, is provided so that user processes can start transactions to make temporary reservations of disk space. Once started, all file system I/O is then bound to the transaction until it closes, at which point the reservation is released and I/O is flushed to storage. To prevent trivial denial-of-service attacks, transactions are only available to root-privileged processes.

History

The core data structure of Btrfs—the copy-on-write B-tree—was originally proposed by IBM researcher Ohad Rodeh at a presentation at USENIX 2007. Rodeh suggested adding reference counts and certain relaxations to the balancing algorithms of standard B-trees that would make them suitable for a high-performance object store with copy-on-write snapshots, yet maintain good concurrency.

Chris Mason, an engineer working on ReiserFS for SUSE at the time, joined Oracle later that year and began work on a new file system using such B-trees almost exclusively—not just for metadata and file data, but also recursively to track space allocation of the trees themselves. This allowed all traversal and modifications to be funneled through a single code path, against which features such as copy-on-write, checksumming and mirroring needed to be implemented only once to benefit the entire file system.

Design

Btrfs is structured as several layers of trees, all using the same B-tree implementation to store their various data types as generic *items* sorted on a 136-bit key. The first 64 bits of the key are a unique *object id*. The middle 8 bits is an item type field; its use is hardwired into code as an item filter in tree lookups. *Objects* can have multiple items of multiple types. The remaining right-hand 64 bits are used in type-specific ways. Therefore items

for the same object end up adjacent to each other in the tree, ordered by type. By choosing certain right-hand key values, objects can further put items of the same type in a particular order.

Interior tree nodes are simply flat lists of key-pointer pairs, where the pointer is the logical block number of a child node. Leaf nodes contain item keys packed into the front of the node and item data packed into the end, with the two growing toward each other as the leaf fills up.

Root tree

Every tree appears as an object in the *root tree* (or *tree of tree roots*). Some trees, such as file system trees and log trees, have a variable number of instances, each of which is given its own object id. Trees which are singletons (the data relocation, extent and chunk trees) are assigned special, fixed object ids ≤256. The root tree appears in itself as a tree with object id 1.

Trees refer to each other by object id. They may also refer to individual nodes in other trees as a triplet of the tree's object id, the node's level within the tree and its leftmost key value. Such references are independent of where the tree is actually stored.

File system tree

User-visible files and directories all live in a *file system tree*. File and directory objects all have inode items. Extended attributes and ACL entries are stored alongside in separate items.

Each directory entry appears as a *directory item*, whose right-hand key value is a CRC32C hash of the filename. They collectively act as an index for path lookups. Directory iteration over these hashed names, however, would return entries in a random order. User applications reading sequentially through large directories would thus generate many more seeks between non-adjacent files—a notable performance drain in other file systems with hash-ordered directories such as ReiserFS, ext3 (with Htree-indexes enabled) and ext4, all of which have TEA-hashed filenames. To avoid this, directory objects keep a *directory index item* for each of their files. The right-hand value of the item is set to a counter that is incremented on every new file. Iteration over these index items returns entries in roughly the same order as they are stored on disk.

Files and directories have a *reference item* whose right-hand key value is the object id of their parent directory. This allows upward traversal through the directory hierarchy. Hard-linked files have multiple such back-references.

There is one file system tree per subvolume. Subvolumes can nest, in which case they appear as a directory item whose data is a reference to the nested subvolume's file system tree.

File data are kept outside the tree in *extents*, which are (possibly zlib-compressed) contiguous runs of disk blocks (which default to 4KB in size). Each such extent is tracked in-tree by an *extent data item*. To keep these items in the correct order, the starting byte offset of the extent is their right-hand key value.

Snapshots and cloned files share extents. When a small part of a large such extent is overwritten, the resulting copy-on-write may create three new extents: a small one containing the overwritten data, and two large ones with unmodified data on either side of the overwrite. To avoid having to re-write unmodified data, the copy-on-write may instead create *bookend extents*, or extents which are simply slices of existing extents. Extent data items allow for this by including an offset into the extent they are tracking: items for bookends are those with non-zero offsets.

If the file data is small enough to fit inside a tree node, it is instead pulled in-tree and stored inline in the extent data item. Tree nodes themselves live in uncompressed, single-block extents.

Extent allocation tree

An *extent allocation tree* is used to track space usage by extents, which are zoned into *block groups*. Block groups are variable-sized allocation regions which alternate successively between preferring metadata extents (tree nodes) and data extents (file contents). The default ratio of data to metadata block groups is 1:2. Inode items include a reference to their current block group. Together, these work like the Orlov allocator and block groups in ext3 in allocating related files together and resisting fragmentation by leaving allocation gaps between groups. (ext3 block groups, however, have fixed locations computed from the size of the file system, whereas those in Btrfs are dynamic and are created as needed.)

Items in the extent allocation tree do not have object ids, but instead use their byte offsets as the left-hand 64 bits of the key. A traditional free-space bitmap is not used, since the allocation tree essentially acts as a B-tree version of a BSP tree. (The current implementation of Btrfs, however, does keep an in-memory red-black tree of page-sized bitmaps to speed up allocations.)

Extent items contain a back-reference to the tree node or file occupying that extent. There may be multiple back-references if the extent is shared between snapshots. If there are too many back-references to fit in the item, they spill out into individual *extent data reference items*. Tree nodes, in turn, have back-references to their containing trees. This makes it possible to find which extents or tree nodes are in any region of space by doing a B-tree range lookup on a pair offsets bracketing that region, then following the back-references. When relocating data, this allows an efficient upwards traversal from the relocated blocks to quickly find and fix up all downwards references to those blocks, without having to walk the entire file system. This, in turn, allows the file system to efficiently shrink, migrate and defragment its storage online.

The extent tree, as with all other trees in the file system, is copy-on-write. Writes to the file system may thus cause a cascade whereby changed tree nodes and file data result in new extents being allocated, causing the extent tree to itself change. To avoid creating a feedback loop, extent tree nodes which are still in memory but not yet committed to disk may be updated in-place to reflect new copy-on-written extents.

Checksum tree

CRC-32C checksums are computed for both data and metadata and stored as *checksum items* in a *checksum tree*. There is one checksum item per contiguous run of allocated blocks, with per-block checksums packed end-to-end into the item data. If there are more checksums than can fit, they spill rightwards over into another checksum item in a new leaf.

Log tree

An fsync is a request to commit modified data immediately to stable storage. fsync-heavy workloads (such as databases) could potentially generate a great deal of redundant write I/O by forcing the file system to repeatedly copy-on-write and flush frequently-modified parts of trees to storage. To avoid this, a temporary per-subvolume *log tree* is created to journal fsync-triggered copy-on-writes. Log trees are self-contained, tracking their own extents and keeping their own checksum items. Their items are replayed and deleted at the next full tree commit or (if there was a system crash) at next remount.

Chunk and device trees

Block devices are divided into *chunks* of 256 MB or more. Chunks may be mirrored or striped across multiple devices. The mirroring/striping arrangement is transparent to the rest of the file system, which simply sees the single, logical address space that chunks are mapped into.

This is all tracked by the *chunk tree*, where each device is represented as a *device item* and each mapping from a logical chunk to its underlying physical chunks is stored in a *chunk map item*. The *device tree* is the inverse of the chunk tree, and contains *device extent items* which map byte ranges of block devices back to individual chunks. As in the extent allocation tree, this allows Btrfs to efficiently shrink or remove devices from volumes by locating the chunks they contain (and relocating their contents).

The file system, chunks and devices are all assigned a UUID. The header of every tree node contains both the UUID of its containing chunk and the UUID of the file system. The chunks containing the chunk tree, the root tree, device tree and extent tree are always mirrored—even on single-device volumes. These are all intended to improve the odds of successful data salvage in the event of media errors.

Data relocation tree

The *data relocation tree* serves as scratch space for extents and their temporary copies during rebalancing or defragmentation. It is exempt from copy-on-write.

Superblock

All the file system's trees—including the chunk tree itself—are stored in chunks, creating a potential chicken-and-egg problem when mounting the file system. To bootstrap into a mount, a list of physical addresses of chunks belonging to the chunk and root trees must be stored in the *superblock*.

Superblock mirrors are kept at fixed locations: 64 KiB into every block device, with additional copies at 64 MiB, 256 GiB and 1 PiB. When a superblock mirror is updated, its *generation number* is incremented. At mount time, the copy with the highest generation number is used. All superblock mirrors are updated in tandem, except when SSD mode is enabled, in which case updates will instead alternate between mirrors to provide some wear levelling.

Chapter 7

Ext2

ext2	
Developer	Rémy Card
Full name	Second extended file system
Introduced	January 1993 (Linux)
Partition identifier	`Apple_UNIX_SVR2` (Apple Partition Map) `0x83` (Master Boot Record) `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7` (GPT)
Structures	
File allocation	bitmap (free space), table (metadata)
Bad blocks	Table
Limits	
Max file size	16 GB - 2 TB
Max number of files	10^{18}
Max filename length	255 bytes
Max volume size	2-32 TB
Allowed characters in filenames	Any byte except NUL and '/'
Features	

Dates recorded	modification (mtime), attribute modification (ctime), access (atime)
Date range	December 14, 1901 - January 18, 2038
Date resolution	1s
File system permissions	POSIX
Transparent compression	No (Available through patches)
Transparent encryption	No
Supported operating systems	Linux, BSD, Windows (through an IFS), Mac OS X (through an IFS)

The **ext2** or **second extended filesystem** is a file system for the Linux kernel. It was initially designed by Rémy Card as a replacement for the extended file system (ext).

The canonical implementation of ext2 is the ext2fs filesystem driver in the Linux kernel. Other implementations (of varying quality and completeness) exist in GNU Hurd, Mac OS X (third-party), Darwin (same third-party as Mac OS X but untested), some BSD kernels, in Atari MiNT, and as third-party Microsoft Windows drivers.

ext2 was the default filesystem in several Linux distributions, including Debian and Red Hat Linux, until supplanted more recently by ext3, which is almost completely compatible with ext2 and is a journaling file system. ext2 is still the filesystem of choice for flash-based storage media (such as SD cards, and USB flash drives) since its lack of a journal minimizes the number of writes and flash devices have only a limited number of write cycles. Recent kernels, however, support a journal-less mode of ext4, which would offer the same benefit along with a number of ext4-specific benefits.

History

The early development of the Linux kernel was made as a cross-development under the Minix operating system. Naturally, it was obvious that the Minix file system would be used as Linux's first file system. The Minix file system was mostly free of bugs, but used 16-bit offsets internally and thus only had a maximum size limit of 64 megabytes. There was also a filename length limit of 14 characters. Because of these limitations, work began on a replacement native file system for Linux.

To ease the addition of new file systems and provide a generic file API, VFS, a virtual file system layer was added to the Linux kernel. The extended file system (ext), was released in April 1992 as the first file system using the VFS API and was included in Linux version 0.96c. The ext file system solved the two major problems in the Minix file system (maximum partition size and filename length limitation to 14 characters), and allowed 2 gigabytes of data and filenames of up to 255 characters. But it still had problems: there was no support for separate access, inode modification and data modification timestamps.

As a solution for these problems, two new filesystems were developed in January 1993: xiafs and the **second extended file system** (**ext2**), which was an overhaul of the extended file system incorporating many ideas from the Berkeley Fast File System. ext2 was also designed with extensibility in mind, with space left in many of its on-disk data structures for use by future versions.

Since then, ext2 has been a testbed for many of the new extensions to the VFS API. Features such as POSIX ACLs and extended attributes were generally implemented first on ext2 because it was relatively simple to extend and its internals were well-understood.

On Linux kernels prior to 2.6.17, restrictions in the block driver mean that ext2 filesystems have a maximum file size of 2TB.

ext2 is still recommended over journaling file systems on bootable USB flash drives and other solid-state drives. ext2 performs fewer writes than ext3 since it does not need to write to the journal. As the major aging factor of a flash chip is the number of erase cycles, and as those happen frequently on writes, this increases the life span of the solid-state device. Another good practice for filesystems on flash devices is the use of the *noatime* mount option, for the same reason.

ext2 data structures

The space in ext2 is split up into blocks. These blocks are divided into block groups, analogous to cylinder groups in the Unix File System. There are typically thousands of blocks on a large file system. Data for any given file is typically contained within a single block group where possible. This is done to reduce external fragmentation and minimize the number of disk seeks when reading a large amount of consecutive data.

Each block group contains a copy of the superblock and block group descriptor table, and all block groups contain a block bitmap, an inode bitmap, an inode table and finally the actual data blocks.

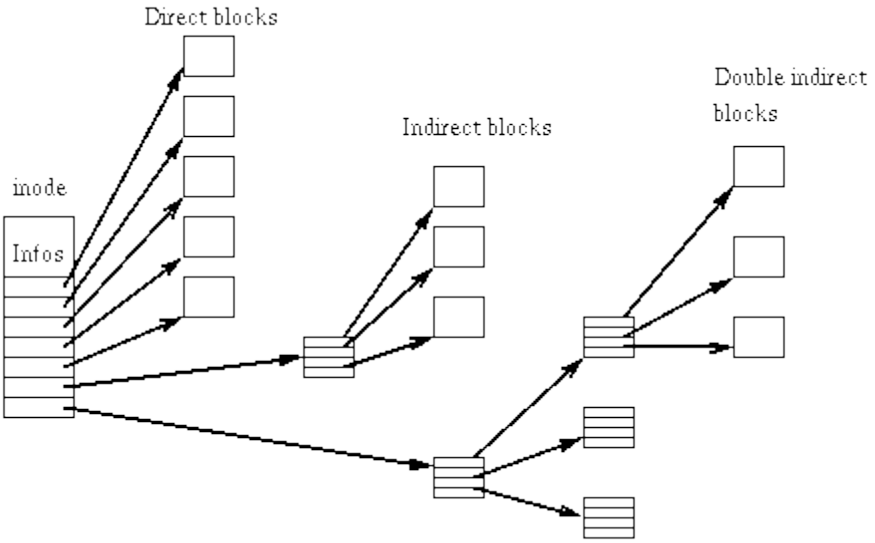
The superblock contains important information that is crucial to the booting of the operating system, thus backup copies are made in multiple block groups in the file system. However, typically only the first copy of it, which is found at the first block of the file system, is used in the booting.

The group descriptor stores the location of the block bitmap, inode bitmap and the start of the inode table for every block group and these, in turn are stored in a group descriptor table.

Inodes

Every file or directory is represented by an inode. The inode includes data about the size, permission, ownership, and location on disk of the file or directory.

Example of ext2 inode structure:



Quote from the linux kernel documentation for ext2:

"There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly-indirect block (which contains pointers to indirect blocks) and a pointer to a trebly-indirect block (which contains pointers to doubly-indirect blocks)."

So, there is a structure in ext2 that has 15 pointers, the first 12 are for direct blocks. Pointer number 13 points to an indirect block, number 14 to a doubly-indirect block and number 15 to a trebly-indirect block.

Directories

Each directory is a list of directory entries. Each directory entry associates one file name with one inode number, and consists of the inode number, the length of the file name, and the actual text of the file name. To find a file, the directory is searched front-to-back for the associated filename. For reasonable directory sizes, this is fine. But for huge large

directories this is inefficient, and ext3 offers a second way of storing directories that is more efficient than just a list of filenames.

The root directory is always stored in inode number two, so that the file system code can find it at mount time. Subdirectories are implemented by storing the name of the subdirectory in the name field, and the inode number of the subdirectory in the inode field. Hard links are implemented by storing the same inode number with more than one file name. Accessing the file by either name results in the same inode number, and therefore the same data.

The special directories "." and ".." are implemented by storing the names "." and ".." in the directory, and the inode number of the current and parent directories in the inode field. The only special treatment these two entries receive is that they are automatically created when any new directory is made, and they cannot be deleted.

Allocating Data

When a new file or directory is created, the EXT2 file system must decide where to store the data. If the disk is mostly empty, then data can be stored almost anywhere. However, performance is maximized if the data is clustered with other related data to minimize seek times.

The EXT2 file system attempts to allocate each new directory in the group containing its parent directory, on the theory that accesses to parent and children directories are likely to be closely related. The EXT2 file system also attempts to place files in the same group as their directory entries, because directory accesses often lead to file accesses. However, if the group is full, then the new file or new directory is placed in some other non-full group.

The data blocks needed to store directories and files can found by looking in the data allocation bitmap. Any needed space in the inode table can be found by looking in the inode allocation bitmap.

File system limits

Theoretical ext2 filesystem limits under Linux

Block size:	1 KB	2 KB	4 KB	8 KB
max. file size:	16 GB	256 GB	2 TB	2 TB
max. filesystem size:	4* TB	8 TB	16 TB	32 TB

The reason for some limits of the ext2-file system are the file format of the data and the operating system's kernel. Mostly these factors will be determined once when the file

system is built. They depend on the block size and the ratio of the number of blocks and inodes. In Linux the block size is limited by the architecture page size.

There are also some userspace programs that can't handle files larger than 2 GB.

The maximum file size is limited to min $((b/4)^3+(b/4)^2+b/4+12, 2^{32}*b)$ due to the i_block (an array of EXT2_N_BLOCKS) and i_blocks(32-bits integer value) representing the amount of b-bytes "blocks" in the file.

The limit of sublevel-directories is 31998 due to the link count limit. Enabling directory indexing will increase performance for directories with a large number of files (10,000+). The theoretical limit on the number of files in a directory is 1.3×10^{20}, although this is not relevant for practical situations.

Note: In Linux kernel 2.4 and earlier block devices were limited to 2 TB, limiting the maximum size of a partition regardless of block size.

Compression extension

e2compr is a modification to the ext2 file system driver in the Linux kernel to support online compression and decompression of files on file system level without any support by user applications.

e2compr is a small patch against the ext2 file system that allows on-the-fly compression and decompression. It compresses only regular files; the administrative data (superblock, inodes, directory files etc.) are not compressed (mainly for safety reasons). Access to compressed blocks is provided for read and write operations. The compression algorithm and cluster size is specified on a per-file basis. Directories can also be marked for compression, in which case every newly created file in the directory will be automatically compressed with the same cluster size and the same algorithm that was specified for the directory.

e2compr is not a new file system. It is only a patch to the ext2 file system made to support the EXT2_COMPR_FL flag. It does not require you to make a new partition, and will continue to read or write existing ext2 file systems. One can consider it as simply a way for the read and write routines to access files that could have been created by a simple utility similar to gzip or compress. Compressed and uncompressed files coexist nicely on ext2 partitions.

The latest e2compr-branch is available for current releases of 2.6 and 2.4 Linux kernels, but development is stalled. There are also older branches for older 2.0 and 2.2 kernels, which are more stable.

Chapter 8

Ext3

ext3	
Developer	Stephen Tweedie
Full name	Third extended file system
Introduced	November 2001 (Linux 2.4.15)
Partition identifier	0x83 (MBR) EBD0A0A2-B9E5-4433-87C0-68B6B72699C7 (GPT)
Structures	
Directory contents	Table, hashed B-tree with dir_index enabled
File allocation	bitmap (free space), table (metadata)
Bad blocks	Table
Limits	
Max file size	16 GB – 2 TB
Max number of files	Variable, allocated at creation time
Max filename length	254 bytes
Max volume size	2 TB – 16 TB
Allowed characters in filenames	All bytes except NULL and '/'

Features	
Dates recorded	modification (mtime), attribute modification (ctime), access (atime)
Date range	December 14, 1901 - January 18, 2038
Date resolution	1s
Attributes	No-atime, append-only, synchronous-write, no-dump, h-tree (directory), immutable, journal, secure-delete, top (directory), allow-undelete
File system permissions	Unix permissions, ACLs and arbitrary security attributes (Linux 2.6 and later)
Transparent compression	No
Transparent encryption	No (provided at the block device level)
Data deduplication	No
Supported operating systems	Linux, BSD, Windows (through an IFS)

The **ext3** or **third extended filesystem** is a journaled file system that is commonly used by the Linux kernel. It is the default file system for many popular Linux distributions. Stephen Tweedie first revealed that he was working on extending ext2 in *Journaling the Linux ext2fs Filesystem* in a 1998 paper and later in a February 1999 kernel mailing list posting, and the filesystem was merged with the mainline Linux kernel in November 2001 from 2.4.15 onward. Its main advantage over ext2 is journaling which improves reliability and eliminates the need to check the file system after an unclean shutdown. Its successor is ext4.

Advantages

Although its performance (speed) is less attractive than competing Linux filesystems such as ext4, JFS, ReiserFS and XFS, it has a significant advantage in that it allows in-place upgrades from the ext2 file system without having to back up and restore data. Benchmarks suggest that ext3 also uses less CPU power than ReiserFS and XFS. It is

also considered safer than the other Linux file systems due to its relative simplicity and wider testing base.

The ext3 file system adds, over its predecessor:

- A Journaling file system
- Online file system growth
- Htree indexing for larger directories. An HTree is a specialized version of a B-tree

Without these, any ext3 file system is also a valid ext2 file system. This has allowed well-tested and mature file system maintenance utilities for maintaining and repairing ext2 file systems to also be used with ext3 without major changes. The ext2 and ext3 file systems share the same standard set of utilities, e2fsprogs, which includes an fsck tool. The close relationship also makes conversion between the two file systems (both forward to ext3 and backward to ext2) straightforward.

While in some contexts the lack of "modern" filesystem features such as dynamic inode allocation and extents could be considered a disadvantage, in terms of recoverability this gives ext3 a significant advantage over file systems with those features. The file system metadata is all in fixed, well-known locations, and there is some redundancy inherent in the data structures that may allow ext2 and ext3 to be recoverable in the face of significant data corruption, where tree-based file systems may not be recoverable.

Size limits

ext3 has a maximum size for both individual files and the entire filesystem. For the filesystem as a whole that limit is 2^{32} blocks. Both limits are dependent on the block size of the filesystem; the following chart summarizes the limits:

Block size	Maximum file size	Maximum file system size
1 KiB	16 GiB	2 TiB
2 KiB	256 GiB	8 TiB
4 KiB	2 TiB	16 TiB
8 KiB[1]	2 TiB	32 TiB

1. In Linux, 8 KiB block size is only available on architectures which allow 8 KiB pages, such as Alpha.

Journaling levels

There are three levels of journaling available in the Linux implementation of ext3:

Journal (lowest risk)

Both metadata and file contents are written to the journal before being committed to the main file system. Because the journal is relatively continuous on disk, this can improve performance in some circumstances. In other cases, performance gets worse because the data must be written twice - once to the journal, and once to the main part of the filesystem.

Ordered (medium risk)
: Only metadata is journaled; file contents are not, but it's guaranteed that file contents are written to disk before associated metadata is marked as committed in the journal. This is the default on many Linux distributions. If there is a power outage or kernel panic while a file is being written or appended to, the journal will indicate the new file or appended data has not been "committed", so it will be purged by the cleanup process. (Thus appends and new files have the same level of integrity protection as the "journaled" level.) However, files being *overwritten* can be corrupted because the original version of the file is not stored. Thus it's possible to end up with a file in an intermediate state between new and old, without enough information to restore either one or the other (the new data never made it to disk completely, and the old data is not stored anywhere). Even worse, the intermediate state might intersperse old and new data, because the order of the write is left up to the disk's hardware. XFS uses this form of journaling.

Writeback (highest risk)
: Only metadata is journaled; file contents are not. The contents might be written before or after the journal is updated. As a result, files modified right before a crash can become corrupted. For example, a file being appended to may be marked in the journal as being larger than it actually is, causing garbage at the end. Older versions of files could also appear unexpectedly after a journal recovery. The lack of synchronization between data and journal is faster in many cases. JFS uses this level of journaling, but ensures that any "garbage" due to unwritten data is zeroed out on reboot.

Disadvantages

Functionality

Since ext3 aims to be backwards compatible with the earlier ext2, many of the on-disk structures are similar to those of ext2. Because of that, ext3 lacks a number of features of more recent designs, such as extents, dynamic allocation of inodes, and block suballocation. There is a limit of 31998 sub-directories per one directory, stemming from its limit of 32000 links per inode.

ext3, like most current Linux filesystems, cannot be fsck-ed while the filesystem is mounted for writing. Attempting to check a file system that is already mounted may detect bogus errors where changed data has not reached the disk yet, and corrupt the file system in an attempt to "fix" these errors.

Defragmentation

There is no online ext3 defragmentation tool that works on the filesystem level. An offline ext2 defragmenter, `e2defrag`, exists but requires that the ext3 filesystem be converted back to ext2 first. But depending on the feature bits turned on in the filesystem, `e2defrag` may destroy data; it does not know how to treat many of the newer ext3 features.

There are userspace defragmentation tools like Shake and defrag. Shake works by allocating space for the whole file as one operation, which will generally cause the allocator to find contiguous disk space. It also tries to write files used at the same time next to each other. Defrag works by copying each file over itself. However they only work if the filesystem is reasonably empty. A true defragmentation tool does not exist for ext3.

That being said, as the Linux System Administrator Guide states, "Modern Linux filesystem(s) keep fragmentation at a minimum by keeping all blocks in a file close together, even if they can't be stored in consecutive sectors. Some filesystems, like ext3, effectively allocate the free block that is nearest to other blocks in a file. Therefore it is not necessary to worry about fragmentation in a Linux system."

While ext3 is more resistant to file fragmentation than the FAT filesystem, nonetheless ext3 filesystems can get fragmented over time or on specific usage patterns, like slowly-writing large files. Consequently the successor to the ext3 filesystem, ext4, is planned to eventually include an online filesystem defragmentation utility, and currently supports extents (contiguous file regions).

Recovery

There is no support of deleted file recovery in file system design. Ext3 driver actively deletes files by wiping file inodes for crash safety reasons. This is why an accidental 'rm -rf ...' command may cause permanent data loss.

There are still several techniques and some commercial software like UFS Explorer Standard Recovery version 4 for recovery of deleted or lost files using file system journal analysis; however, they do not guarantee any specific file recovery.

Compression

Support for transparent compression is available as an unofficial patch for ext3. This patch is a direct port of e2compr and still needs further development, it compiles and boots well with upstream kernels but journaling is not implemented yet. The current patch is named e3compr.

Lack of snapshots support

Unlike a number of modern file systems, Ext3 does not have native support for snapshots - the ability to quickly capture the state of the filesystem at arbitrary times, instead relying on less space-efficient volume level snapshots provided by the Linux LVM. The Next3 file system is a modified version of Ext3 which offers snapshots support, yet retains compatibility to the EXT3 on-disk format.

No checksumming in journal

Ext3 does not do checksumming when writing to the journal. If *barrier=1* is not enabled as a mount option (in /etc/fstab), and if the hardware is doing out-of-order write caching, one runs the risk of severe filesystem corruption during a crash.

Consider the following scenario: If hard disk writes are done out-of-order (due to modern hard disks caching writes in order to amortize write speeds), it is likely that one will write a commit block of a transaction before the other relevant blocks are written. If a power failure or unrecoverable crash should occur before the other blocks get written, the system will have to be rebooted. Upon reboot, the file system will replay the log as normal, and replay the "winners" (transactions with a commit block, including the invalid transaction above which happened to be tagged with a valid commit block). The unfinished disk write above will thus proceed, but using corrupt journal data. *The file system will thus mistakenly overwrite normal data with corrupt data while replaying the journal.* There is a test program available to trigger the problematic behavior. If checksums had been used, where the blocks of the "fake winner" transaction were tagged with a mutual checksum, the file system could have known better and not replayed the corrupt data onto the disk. Journal checksumming has been added to ext4.

Filesystems going through the device mapper interface (including software RAID and LVM implementations) may not support barriers, and will issue a warning if that mount option is used. There are also some disks that do not properly implement the write cache flushing extension necessary for barriers to work, which causes a similar warning. In these situations, where barriers are not supported or practical, reliable write ordering is possible by turning off the disk's write cache and using the data=journal mount option. Turning off the disk's write cache may be required even when barriers are available. Applications like databases expect a call to fsync() will flush pending writes to disk, and the barrier implementation doesn't always clear the drive's write cache in response to that call. There is also a potential issue with the barrier implementation related to error handling during events such as a drive failure

ext4

An enhanced version of the filesystem was announced by Theodore Ts'o on June 28, 2006 under the name of ext4. On October 11, 2008, the patches that mark ext4 as stable code were merged in the Linux 2.6.28 source code repositories, marking the end of the development phase and recommending its adoption.

Chapter 9

Ext4

ext4	
Developer	Mingming Cao, Andreas Dilger, Alex Zhuravlev (Tomas), Dave Kleikamp, Theodore Ts'o, Eric Sandeen, others
Full name	Fourth extended file system
Introduced	**Stable:** 21 October 2008 **Unstable:** 10 October 2006 (Linux 2.6.28, 2.6.19)
Partition identifier	0x83 (MBR) EBD0A0A2-B9E5-4433-87C0-68B6B72699C7 (GPT)
Structures	
Directory contents	Linked list, hashed B-tree
File allocation	Extents/Bitmap
Bad blocks	Table
Limits	
Max file size	16 TiB (for 4k block filesystem)
Max number of files	4 billion (specified at filesystem creation time)
Max filename length	256 bytes

Max volume size	1 EiB (limited to 16TiB because of e2fsprogs limitation until v1.42)
Allowed characters in filenames	All bytes except NULL ('\0') and '/'

Features

Dates recorded	modification (mtime), attribute modification (ctime), access (atime), delete (dtime), create (crtime)
Date range	14 December 1901 - 25 April 2514
Date resolution	Nanosecond
Forks	No
Attributes	extents, noextents, mballoc, nomballoc, delalloc, nodelalloc, data=journal, data=ordered, data=writeback, commit=nrsec, orlov, oldalloc, user_xattr, nouser_xattr, acl, noacl, bsddf, minixdf, bh, nobh, journal_dev
File system permissions	POSIX
Transparent compression	No
Transparent encryption	No
Data deduplication	No
Supported operating systems	Linux

The **ext4** or **fourth extended filesystem** is a journaling file system for Linux, developed as the successor to ext3.

It was born as a series of backward compatible extensions to ext3, many of them originally developed by Cluster File Systems for the Lustre file system between 2003 and 2006, meant to extend storage limits and add other performance improvements. However, other Linux kernel developers opposed accepting extensions to ext3 for stability reasons, and proposed to fork the source code of ext3, rename it as ext4, and do all the development there, without affecting the current ext3 users. This proposal was accepted, and on 28 June 2006, Theodore Ts'o, the ext3 maintainer, announced the new plan of development for ext4.

A preliminary development version of ext4 was included in version 2.6.19 of the Linux kernel. On 11 October 2008, the patches that mark ext4 as stable code were merged in the Linux 2.6.28 source code repositories, denoting the end of the development phase and recommending ext4 adoption. Kernel 2.6.28, containing the ext4 filesystem, was finally released on 25 December 2008. On 15 January 2010, Google announced that it would upgrade its storage infrastructure from ext2 to ext4. On December 14, 2010, they also announced they would use ext4, instead of YAFFS, on Android 2.3.

Features

Large file system
: The ext4 filesystem can support volumes with sizes up to 1 exabyte and files with sizes up to 16 terabytes. The current e2fsprogs can only handle a filesystem of 16 TB, but support for larger drives is under development.

Extents
: Extents replace the traditional block mapping scheme used by ext2/3 filesystems. An extent is a range of contiguous physical blocks, improving large file performance and reducing fragmentation. A single extent in ext4 can map up to 128 MB of contiguous space with a 4 KB block size. There can be 4 extents stored in the inode. When there are more than 4 extents to a file, the rest of the extents are indexed in an Htree.

Backward compatibility
: The ext4 filesystem is backward compatible with ext3 and ext2, making it possible to mount ext3 and ext2 filesystems as ext4. This will slightly improve performance, because certain new features of ext4 can also be used with ext3 and ext2, such as the new block allocation algorithm.
: The ext3 file system is partially forward compatible with ext4, that is, an ext4 filesystem can be mounted as an ext3 partition (using "ext3" as the filesystem type when mounting). However, if the ext4 partition uses extents (a major new feature of ext4), then the ability to mount the file system as ext3 is lost.

Persistent pre-allocation
: The ext4 filesystem allows for pre-allocation of on-disk space for a file. The current method for this on most file systems is to write the file full of 0s to reserve the space when the file is created. This method is no longer required for ext4; instead, a new fallocate() system call was added to the Linux kernel for use by filesystems, including ext4 and XFS, that have this capability. The space allocated for files such as these would be guaranteed and would likely be contiguous. This has applications for media streaming and databases.

Delayed allocation

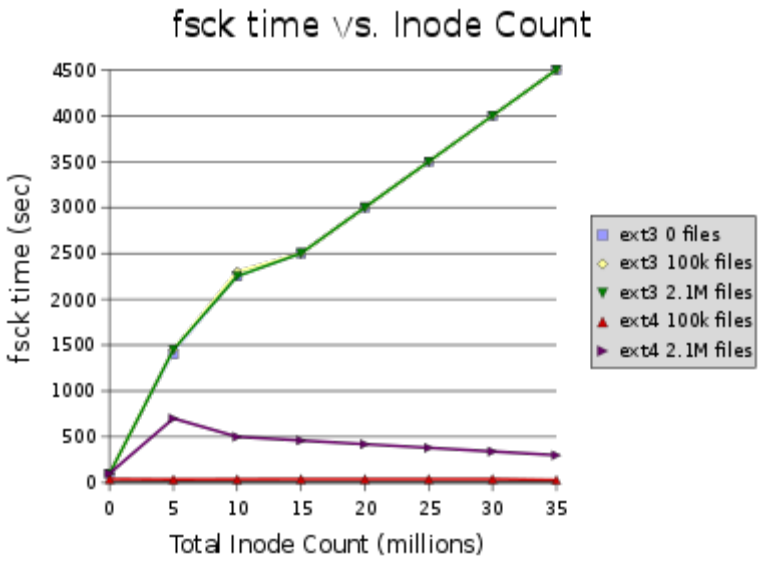
Ext4 uses a filesystem performance technique called allocate-on-flush, also known as *delayed allocation*. It consists of delaying block allocation until the data is going to be written to the disk, unlike some other file systems, which may allocate the necessary blocks before that step. This improves performance and reduces fragmentation by improving block allocation decisions based on the actual file size.

Break 32,000 subdirectory limit

In ext3 the number of subdirectories that a directory can contain is limited to 32,000. This limit has been raised to 64,000 in ext4, and with the "dir_nlink" feature it can go beyond this (although it will stop increasing the link count on the parent). To allow for continued performance given the possibility of much larger directories, Htree indexes (a specialized version of a B-tree) are turned on by default in ext4. This feature is implemented in Linux kernel 2.6.23. Htree is also available in ext3 when the dir_index feature is enabled.

Journal checksumming

Ext4 uses checksums in the journal to improve reliability, since the journal is one of the most used files of the disk. This feature has a side benefit; it can safely avoid a disk I/O wait during the journaling process, improving performance slightly. The technique of journal checksumming was inspired by a research paper from the University of Wisconsin titled *IRON File Systems* (specifically, section 6, called "transaction checksums").



fsck time/Inode Count(ext3 vs. ext4)

Faster file system checking

In ext4, unallocated block groups and sections of the inode table are marked as such. This enables e2fsck to skip them entirely on a check and greatly reduces the time it takes to check a file system of the size ext4 is built to support. This feature is implemented in version 2.6.24 of the Linux kernel.

Multiblock allocator
: When a file is being appended to, ext3 calls the block allocator once for each block individually; with multiple concurrent writers, files can easily become fragmented on disk. With delayed allocation, however, ext4 buffers up a larger amount of data, and then allocates a group of blocks in a batch. This means that the allocator has more information about what's being written and can make better choices for allocating files contiguously on disk. The multiblock allocator is used when delayed allocation is enabled for a file system, or when files are opened in O_DIRECT mode. This feature does not affect the disk format.

Improved timestamps
: As computers become faster in general and as Linux becomes used more for mission critical applications, the granularity of second-based timestamps becomes insufficient. To solve this, ext4 provides timestamps measured in nanoseconds. In addition, 2 bits of the expanded timestamp field are added to the most significant bits of the seconds field of the timestamps to defer the year 2038 problem for an additional 204 years.

  Ext4 also adds support for date-created timestamps. But, as Theodore Ts'o points out, while it is easy to add an extra creation-date field in the inode (thus technically enabling support for date-created timestamps in ext4), it is more difficult to modify or add the necessary system calls, like stat() (which would probably require a new version), and the various libraries that depend on them (like glibc). These changes would require coordination of many projects. So, even if ext4 developers implement initial support for creation-date timestamps, this feature will not be available to user programs for now.

Disadvantages

Delayed allocation and potential data loss

Because delayed allocation changes the behavior that programmers have been relying on with ext3, the feature poses some additional risk of data loss in cases where the system crashes or loses power before all of the data has been written to disk. Other Linux file systems like XFS have never offered ext3-like behavior. Due to this, ext4 in kernel versions 2.6.30 and later automatically detects these cases and reverts to the old behavior.

The typical scenario in which this might occur is a program replacing the contents of a file without forcing a write to the disk with fsync. There are two common ways of replacing the contents of a file on Unix systems:

- fd=open("file", O_TRUNC); write(fd, data); close(fd);

  In this case, an existing file is truncated at the time of open (due to O_TRUNC flag), then new data is written out. Since the write can take some time, there is an opportunity of losing contents even with ext3, but usually very small. However, because ext4 can delay allocating file data for a long time, this opportunity is much greater.

There are several problems with this approach:

1. If the write does not succeed (which may be due to error conditions in the writing program, or due to external conditions such as a full disk), then both the original version *and* the new version of the file will be lost, and the file may be corrupted because only a part of it has been written.
2. If other processes access the file while it is being written, they see a corrupted version.
3. If other processes have the file open and do not expect its contents to change, those processes may crash. One notable example is a shared library file which is mapped into running programs.

Because of these issues, often the following idiom is preferred over the above one:

- fd=open("file.new"); write(fd, data); close(fd); rename("file.new", "file");

  A new temporary file ("file.new") is created, which initially contains the new contents. Then the new file is renamed over the old one. Replacing files by the "rename" call is guaranteed to be atomic by POSIX standards – i.e. either the old file remains, or it's overwritten with the new one. Because the ext3 default "ordered" journalling mode guarantees file data is written out on disk before metadata, this technique guarantees that either the old or the new file contents will persist on disk. ext4's delayed allocation breaks this expectation, because the file write can be delayed for a long time, and the rename is usually carried out before new file *contents* reach the disk.

Using fsync more often to reduce the risk for ext4 could lead to performance penalties on ext3 filesystems mounted with the `data=ordered` flag (the default on most Linux distributions). Given that both file systems will be in use for some time, this complicates matters for end-user application developers. In response, ext4 in Linux kernels 2.6.30 and newer detect the occurrence of these common cases and force the files to be allocated immediately. For a small cost in performance, this provides semantics similar to ext3 ordered mode and increases the chance that either version of the file will survive the crash. This new behavior is enabled by default, but can be disabled with the "noauto_da_alloc" mount option.

The new patches have become part of the mainline kernel 2.6.30, but various distributions chose to backport them to 2.6.28 or 2.6.29. For instance Ubuntu made them part of the 2.6.28 kernel in version 9.04 ("Jaunty Jackalope").

These patches don't completely prevent potential data loss or help at all with new files. No other filesystem is perfect in terms of data loss either, although the probability of data loss is lower on ext3. The only way to be safe is to write and use software that does fsync when it needs to. Performance problems can be minimized by limiting crucial disk writes that need fsync to occur less frequently.

Chapter 10

FAT Filesystem & Linux

Linux has several filesystem drivers for the File Allocation Table (FAT) filesystem format. These are commonly known by the names used in the `mount` command to invoke particular drivers in the kernel: *msdos*, *vfat*, and *umsdos*.

Differences, advantages, and disadvantages

All of the Linux filesystem drivers support all of the three File Allocation Table sizes, 12-bit, 16-bit, and 32-bit (the last commonly known as FAT32). Where they differ is in the provision of support for long filenames, beyond the 8.3 filename structure of the original FAT filesystem format, and in the provision of Unix file semantics that do not exist as standard in the FAT filesystem format such as file permissions. The filesystem drivers are mutually exclusive. Only one can be used to mount any given disc volume at any given time. Thus the choice among them is determined by what long filenames and Unix semantics they support and what use one wants to make of the disc volume.

The *msdos* filesystem driver provides no extra Unix file semantics and no long filename support. If a FAT disc filesystem is mounted using this driver, only 8.3 filenames will be visible, no long filenames will be accessible, nor will any long filename data structures of any kind on the disc volume be maintained. The *vfat* filesystem driver provides long filename support using the same disc data structures that Microsoft Windows uses for long filename support on FAT format volumes, but it does not support any extra Unix file semantics. The *umsdos* filesystem driver provides long filename support, and extra Unix file semantics. However, it does so using on-disc data structures that are not recognized by any filesystem drivers for any operating systems other than Linux.

The key advantage to *umsdos* out of the three is that it provides full Unix file semantics. Therefore it can be used in situations where it is desirable to install Linux on and run it from a FAT disc volume, which require such semantics to be available. However, Linux installed on and running from such a disc volume is slower than Linux installed on and running from a disc volume formatted with, for example, the ext2 filesystem format. Further, unless a utility program is regularly run every time that one switches from running Windows to running Linux, certain changes made to files and directories on the disc by Windows will cause error messages about inaccessible files in Linux.

vfat, whilst lacking full Unix file semantics and lacking the ability to have Linux installed on and running from a FAT disc volume, does not have the aforementioned disadvantages of *umsdos* when it comes to simply sharing data on a FAT disc volume between Linux and other operating systems such as Windows. Its data structures are the same as those used by Windows for long filenames, and it does not require running a synchronization utility in order to prevent Windows and Linux data structures from becoming disjoint. For this reason, it is the most appropriate of Linux's FAT filesystem drivers to use in the majority of situations.

Commonalities

As mentioned previously, all of the Linux filesystem drivers support all of the three File Allocation Table sizes, 12-bit, 16-bit, and 32-bit. Other common features that they all support are various Linux mounting options (specified with the `-o` option to the `mount` command):

`uid` and `gid`
: These two options tell the filesystem driver to set the (default, in the case of *umsdos*) owner ID and group ID to be a single, specified, value for all files in the volume. Both IDs are specified as numeric values (as to be found in the `/etc/passwd` file). So, for example, to specify to the *vfat* filesystem driver that all files and directories are to have owner ID 745 and group ID 15, the `mount` command would be invoked as mount -t vfat -o uid=745,gid=15 .

`umask`
: This option sets the umask to apply globally to all files in the volume. For example, to specify to the *vfat* filesystem driver that no "group" or "other" access is to be allowed, the `mount` command would be invoked as mount -t vfat -o umask=077 .

`conv`
: This option specifies *file content conversion* semantics. It is possible for the filesystem drivers to convert the newline conventions in files, between LF termination and CRLF termination, on the fly as files are read and written. By default this conversion is entirely disabled. The filesystem drivers can perform conversion for some files, attempting to auto-detect what files to convert based upon the extension portion of the filename, or globally for all files. These three conversion levels are specified as `conv=b` (for "binary"), `conv=a` (for "auto-detect"), and `conv=t` (for "text"), respectively. The latter two options carry an inherent risk of corrupting non-text file data. No conversion at all is the default.

Data structures of umsdos

The *umsdos* FAT filesystem driver stores all of the extra information relating to Unix file semantics in what, to another FAT filesystem driver, appears to be just a normal file in each directory and subdirectory, named `--LINUX-.---`.

In the absence of this file in any given directory, and thus by default, the *umsdos* filesystem driver provides the same semantics as the *msdos* filesystem driver does for the directory: only 8.3 filenames and no extra Unix file semantics. To enable the *umsdos* driver's extra abilities, it is necessary to create that file in the directory and synchronize its internal data with the normal FAT data for any existing entries already in the directory. This is done with a tool called `umssync`.

This is the utility program that is run, across every directory on the disc volume, every time that one switches from running Windows to running Linux, in order for the *umsdos* filesystem driver to incorporate any changes made to files and directories by Windows into its private data structures in its `--LINUX-.---` file. By default, the `umssync` tool creates `--LINUX-.---` files in directories if they do not already exist, resulting in such a file in every directory in the disc volume. When switching between Windows and Linux this behaviour is not often considered desirable. Therefore the normal mode of operation when invoking `umssync` after switching from Windows to Linux (which is usually done by running the tool at Linux boot time from a startup script) is to employ the `-c` option to the command, which prevents the creation of any new `--LINUX-.---` files in directories that do not already possess them.

Installing Linux on and booting it from FAT volumes using umsdos

As mentioned, *umsdos* permits installing Linux on, and then bootstrapping and running it from, a FAT format disc volume. The advantage of this is that it permits the use of Linux on a computer where DOS is already installed, without requiring that the hard disc be repartitioned. Linux is not bootstrapped directly from a Volume Boot Record in such a scenario. Instead DOS is first bootstrapped, and loadlin is used to then bootstrap Linux from DOS.

The convention for such an installation is for the Linux root directory to be a subdirectory of the actual root directory of the DOS boot volume, e.g. `C:\LINUX` . The various Linux top-level directories are thus, to DOS, directories such as `C:\LINUX\ETC` (for `/etc`), `C:\LINUX\BIN` (for `/bin`), `C:\LINUX\LIB` (for `/lib`), and so forth. The *umsdos* filesystem driver automatically prepends the `C:\LINUX\` to all pathnames. The location of the Linux root directory is supplied to the *umsdos* filesystem driver in the first place via an option to the `loadlin` command. So, for example, for the aforegiven root directory `loadlin` would be invoked with a command line such as loadlin c:\linux\boot\vmlinuz rw root=c:\linux .

The installation of Linux into such a directory in the first place simply involves unpacking files from an archive into that directory and its subdirectories. Such an installation also generally requires the use of a swap file rather than a swap partition for Linux, however this is related to the desire not to repartition the hard disc and unrelated to the *umsdos* filesystem driver per se.

Development history and kernel/distribution support

Most of the major Linux distributions, including RedHat, SuSE, and Debian, do not employ *umsdos* to permit installation of Linux on a FAT disc volume. A few distributions do, however. These include distributions such as Phat Linux, a distribution created by two schoolchildren which installs in `C:\PHAT` on DOS by unpacking a ZIP file and is booted by running a COMMAND.COM script named `LINUX.BAT` and ZipSlack.

The UMSDOS project was started in 1992 by Jacques Gelinas and made available to the net in January 1994 as a patch. It was included in the standard distribution starting with kernel 1.1.36. UMSDOS was removed from the Linux 2.6.11 kernel for lack of maintenance. UVFAT, an extension of UMSDOS to use the Windows data structures for long filenames instead of its own, was discontinued before release. They should work in 2.4.x kernels.

Earlier Linux distributions which used UMSDOS are MuLinux, Monkey Linux and Winlinux 2000.

Accessing FAT formatted volumes without kernel support

Although the filesystem drivers in the kernel make it possible to access files and directories on FAT formatted volumes in the normal manner, it is also possible to do so without kernel driver support, using the utility programs that form the mtools utility suite. Like the *vfat* FAT filesystem driver, mtools provides long filename support using the same disc data structures that Microsoft Windows uses.

Chapter 11

Global File System

GFS	
Developer	Red Hat (formerly, Sistina Software)
Full name	Global File System
Introduced	1996 (IRIX (1996), Linux (1997))
Structures	
Directory contents	Hashed (small directories stuffed into inode)
File allocation	bitmap (resource groups)
Bad blocks	No
Limits	
Max number of files	Variable
Max filename length	255 bytes
Allowed characters in filenames	All except NULL
Features	
Dates recorded	attribute modification (ctime), modification (mtime), access (atime)
Date resolution	1s
Attributes	No-atime, journaled data (regular files

	only), inherit journaled data (directories only), synchronous-write, append-only, immutable, exhash (dirs only, read only)
File system permissions	Unix permissions, ACLs
Transparent compression	No
Transparent encryption	No
Data deduplication	across nodes only
Supported operating systems	IRIX (now obsolete), FreeBSD (now obsolete), Linux

GFS2

Developer	Red Hat
Full name	Global File System 2
Introduced	2005 (Linux 2.6.19)
	Structures
Directory contents	Hashed (small directories stuffed into inode)
File allocation	bitmap (resource groups)
Bad blocks	No
	Limits
Max number of files	Variable
Max filename length	255 bytes
Allowed characters in	All except NULL

filenames	
	Features
Dates recorded	attribute modification (ctime), modification (mtime), access (atime)
Date resolution	Nanosecond
Attributes	No-atime, journaled data (regular files only), inherit journaled data (directories only), synchronous-write, append-only, immutable, exhash (dirs only, read only)
File system permissions	Unix permissions, ACLs and arbitrary security attributes
Transparent compression	No
Transparent encryption	No
Data deduplication	across nodes only
Supported operating systems	Linux

In computing, the **Global File System** (**GFS**) is a shared disk file system for Linux computer clusters.

GFS and GFS2 differ from distributed file systems (such as AFS, Coda, or InterMezzo) because they allow all nodes to have direct concurrent access to the same shared block storage. In addition, GFS or GFS2 can also be used as a local filesystem.

GFS has no disconnected operating-mode, and no client or server roles. All nodes in a GFS cluster function as peers. Using GFS in a cluster requires hardware to allow access to the shared storage, and a lock manager to control access to the storage. The lock manager operates as a separate module: thus GFS and GFS2 can use the Distributed Lock Manager (DLM) for cluster configurations and the "nolock" lock manager for local filesystems. Older versions of GFS also support GULM, a server based lock manager which implements redundancy via failover.

GFS and GFS2 are free software, distributed under the terms of the GNU General Public License.

History

Development of GFS began in 1995 and was originally developed by University of Minnesota professor Matthew O'Keefe and a group of students. It was originally written for SGI's IRIX operating system, but in 1998 it was ported to Linux since the open source code provided a more convenient development platform. In late 1999/early 2000 it made its way to Sistina Software, where it lived for a time as an open-source project. Sometime in 2001 Sistina made the choice to make GFS a commercial product — not under an open-source license.

Developers forked OpenGFS from the last public release of GFS and then further enhanced it to include updates allowing it to work with OpenDLM. But OpenGFS and OpenDLM became defunct, since Red Hat purchased Sistina in December 2003 and released GFS and many cluster-infrastructure pieces under the GPL in late June 2004.

Red Hat subsequently financed further development geared towards bug-fixing and stabilization. A further development, **GFS2** derives from GFS and was included along with its distributed lock manager (shared with GFS) in Linux 2.6.19. Red Hat Enterprise Linux 5.2 included GFS2 as a kernel module for evaluation purposes. With the 5.3 update, GFS2 became part of the kernel package.

As of 2009, GFS forms part of the Fedora, Red Hat Enterprise Linux 5.3 and upwards and associated CentOS Linux distributions. Users can purchase commercial support to run GFS fully supported on top of Red Hat Enterprise Linux. Since Red Hat Enterprise Linux version 5.3, Red Hat Enterprise Linux Advanced Platform has included support for GFS at no additional cost.

The following list summarizes some version numbers and major features introduced:

- v1.0 (1996) SGI IRIX only
- v3.0 Linux port
- v4 journaling
- v5 Redundant Lock Manager
- v6.1 (2005) Distributed Lock Manager
- Linux 2.6.19 - GFS2 and DLM merged into Linux kernel
- Red Hat Enterprise Linux 5.3 releases the first fully supported GFS2

Hardware

The design of GFS and of GFS2 targets SAN-like environments. Although it is possible to use them as a single node filesystem, the full feature-set requires a SAN. This can take the form of iSCSI, FibreChannel, AoE, or any other device which can be presented under Linux as a block device shared by a number of nodes.

The DLM requires an IP based network over which to communicate. This is normally just Ethernet, but again, there are many other possible solutions. Depending upon the choice of SAN, it may be possible to combine this, but normal practice involves separate networks for the DLM and storage.

The GFS requires fencing hardware of some kind. This is a requirement of the cluster infrastructure, rather than GFS/GFS2 itself, but it is required for all multi-node clusters. The usual options include power switches and remote access controllers (e.g. DRAC, IPMI, or ILO). Fencing is used to ensure that a node which the cluster believes to be failed cannot suddenly start working again while another node is recovering the journal for the failed node. It can also optionally restart the failed node automatically once the recovery is complete.

Differences from a local filesystem

Although the designers of GFS/GFS2 aimed to emulate a local filesystem closely, there are a number of differences to be aware of. Some of these are due to the existing filesystem interfaces not allowing the passing of information relating to the cluster. Some stem from the difficulty of implementing those features efficiently in a clustered manner. For example:

- The flock() system call on GFS/GFS2 is not interruptible by signals.
- The fcntl() F_GETLK system call returns a PID of any blocking lock. Since this is a cluster filesystem, that PID might refer to a process on any of the nodes which have the filesystem mounted. Since the purpose of this interface is to allow a signal to be sent to the blocking process, this is no longer possible.
- Leases are not supported with the lock_dlm (cluster) lock module, but they are supported when used as a local filesystem
- dnotify will work on a "same node" basis, but its use with GFS/GFS2 is not recommended
- inotify will also work on a "same node" basis, and is also not recommended (but it may become supported in the future)
- splice is supported on GFS2 only

The other main difference, and one that is shared by all similar cluster filesystems, is that the cache control mechanism, known as glocks (pronounced Gee-locks) for GFS/GFS2, has an effect across the whole cluster. Each inode on the filesystem has two glocks associated with it. One (called the iopen glock) keeps track of which processes have the inode open. The other (the inode glock) controls the cache relating to that inode. A glock has four states, UN (unlocked), SH (shared - a read lock), DF (deferred - a read lock incompatible with SH) and EX (exclusive). Each of the four modes maps directly to a DLM lock mode.

When in EX mode, an inode is allowed to cache data and metadata (which might be "dirty", i.e. waiting for write back to the filesystem). In SH mode, the inode can cache data and metadata, but it must not be dirty. In DF mode, the inode is allowed to cache

metadata only, and again it must not be dirty. The DF mode is used only for direct I/O. In UN mode, the inode must not cache any metadata.

In order that operations which change an inode's data or metadata do not interfere with each other, an EX lock is used. This means that certain operations, such as create/unlink of files from the *same* directory and writes to the *same* file should be, in general, restricted to one node in the cluster. Of course, doing these operations from multiple nodes will work as expected, but due to the requirement to flush caches frequently, it will not be very efficient.

The single most frequently asked question about GFS/GFS2 performance is why the performance can be poor with email servers. It should be reasonably obvious from the above that the solution is to break up the mail spool into separate directories and to try and keep (so far as is possible) each node reading and writing to a private set of directories.

Journaling

GFS and GFS2 are both journaled filesystems; and GFS2 supports a similar set of journaling modes as ext3. In `data=writeback` mode, only metadata is journaled. This is the only mode supported by GFS, however it is possible to turn on journaling on individual data-files, but only when they are of zero size. Journaled files in GFS have a number of restrictions placed upon them, such as no support for the mmap or sendfile system calls, they also use a different on-disk format from regular files. There is also an "inherit-journal" attribute which when set on a directory causes all files (and sub-directories) created within that directory to have the journal (or inherit-journal, respectively) flag set. This can be used instead of the `data=journal` mount option which ext3 supports (and GFS/GFS2 doesn't).

GFS2 also supports `data=ordered` mode which is similar to `data=writeback` except that dirty data is synced before each journal flush is completed. This ensures that blocks which have been added to an inode will have their content synced back to disk before the metadata is updated to record the new size and thus prevents uninitialised blocks appearing in a file under node failure conditions. The default journaling mode is `data=ordered`, to match ext3's default.

As of 2010 GFS2 does not yet support `data=journal` mode, but it does (unlike GFS) use the same on-disk format for both regular and journaled files, and it also supports the same journaled and inherit-journal attributes. GFS2 also relaxes the restrictions on when a file may have its journaled attribute changed to any time that the file is not open (also the same as ext3).

For performance reasons, each node in GFS and GFS2 has its own journal. In GFS the journals are disk extents, in GFS2 the journals are just regular files. The number of nodes which may mount the filesystem at any one time is limited by the number of available journals.

Features of GFS2 compared with GFS

GFS2 adds a number of new features which are not in GFS. Here is a summary of those features not already mentioned in the boxes to the right of this page:

- The metadata filesystem (really a different root)
- GFS2 specific trace points have been available since kernel 2.6.32
- The XFS-style quota interface has been available in GFS2 since kernel 2.6.33
- Caching ACLs have been available in GFS2 since 2.6.33
- GFS2 supports the generation of "discard" requests for thin provisioning/SCSI TRIM requests
- GFS2 supports I/O barriers (on by default, assuming underlying device supports it. Configurable from kernel 2.6.33 and up)
- FIEMAP ioctl (to query mappings of inodes on disk)
- splice system call support
- mmap/splice support for journaled files (enabled by using the same on disk format as for regular files)
- Far fewer tweekables (making set-up less complicated)
- Ordered write mode (as per ext3, GFS only has writeback mode)

Compatibility and the GFS2 meta filesystem

GFS2 was designed so that upgrading from GFS would be a simple procedure. To this end, most of the on-disk structure has remained the same as GFS, including the big-endian byte ordering. There are a few differences though:

- GFS2 has a "meta filesystem" through which processes access system files
- GFS2 uses the same on-disk format for journaled files as for regular files
- GFS2 uses regular (system) files for journals, whereas GFS uses special extents
- GFS2 has some other "`per_node`" system files
- The layout of the inode is (very slightly) different
- The layout of indirect blocks differs slightly

The journaling systems of GFS and GFS2 are not compatible with each other. Upgrading is possible by means of a tool (`gfs2_convert`) which is run with the filesystem off-line to update the metadata. Some spare blocks in the GFS journals are used to create the (very small) `per_node` files required by GFS2 during the update process. Most of the data remains in place.

The GFS2 "meta filesystem" is not a filesystem in its own right, but an alternate root of the main filesystem. Although it behaves like a "normal" filesystem, its contents are the various system files used by GFS2, and normally users do not need to ever look at it. The GFS2 utilities mount and unmount the meta filesystem as required, behind the scenes.

Chapter 12

JFFS and JFFS2

JFFS

JFFS	
Developer	Axis Communications
Full name	Journalling Flash File System
Features	
Supported operating systems	Linux

The **Journalling Flash File System** (or **JFFS**) is a log-structured file system for use on NOR flash memory devices on the Linux operating system. It has been superseded by JFFS2.

Design

Flash memory (specifically NAND flash) must be erased prior to writing. The erase process has several limitations:

- Erasing is very slow (typically 1-100 ms per erase block, which is 10^3-10^5 times slower than reading data from the same region)
- It is only possible to erase flash in large segments (usually 64 KiB or more), whereas it can be read or written in smaller blocks (often 512 bytes)
- Flash memory can only be erased a limited number of times (typically 10^3-10^6) before it becomes worn out

These constraints combine to produce a profound asymmetry between patterns of read and write access to flash memory. In contrast, magnetic hard disk drives offer nearly symmetrical read and write access: read speed and write speed are nearly identical (as both are constrained by the rate at which the disk spins), it is possible to both read and

write small blocks or sectors (typically 512 bytes), and there is no practical limit to the number of times magnetic media can be written and rewritten.

Traditional file systems, such as ext2 or FAT which were designed for use on magnetic media typically update their data structures in-place, with data structures like inodes and directories updated on-disk after every modification. This concentrated lack of wear-levelling makes conventional file systems unsuitable for read-write use on flash devices.

JFFS enforces wear levelling by treating the flash device as a circular log. All changes to files and directories is written to the tail of the log in *nodes*. In each node, a header containing metadata is written first, followed by file data, if any. Nodes are chained together with offset pointers in the header. Nodes start out as *valid* and then become *obsolete* when a newer version of them is created.

The free space remaining in the file system is the gap between the log's tail and its head. When this runs low, a garbage collector copies valid nodes from the head to the tail and skips obsolete ones, thus reclaiming space.

Disadvantages

- At mount time, the file system driver must read the entire inode chain and then keep it in memory. This can be very slow. Memory consumption of JFFS is also proportional to the number of files in the file system.
- The circular log design means *all* data in the filesystem is re-written, regardless of whether it is static or not. This generates many unnecessary erase cycles and reduces the life of the flash medium.

JFFS2

	JFFS2
Developer	David Woodhouse
Full name	Journalling Flash File System version 2
Introduced	(Linux 2.4.10)
	Features
Transparent compression	zlib, rubin and rtime

Supported operating systems	Linux

Journalling Flash File System version 2 or **JFFS2** is a log-structured file system for use in flash memory devices. It is the successor to JFFS. JFFS2 has been included in the Linux kernel since the 2.4.10 release. JFFS2 is also available for Open Firmware, the eCos RTOS and the RedBoot bootloader.

The new LogFS filesystem is aimed to replace JFFS2 for most uses, but focuses more on large devices (> 64 MB - 512 MB).

Features

JFFS2 introduced:

- Support for NAND flash devices. This involved a considerable amount of work as NAND devices have a sequential I/O interface and cannot be memory-mapped for reading.
- Hard links. This was not possible in JFFS because of limitations in the on-disk format.
- Compression. Three algorithms are available: zlib, rubin and rtime.
- Better performance. JFFS treated the disk as a purely circular log. This generated a great deal of unnecessary I/O. The garbage collection algorithm in JFFS2 makes this mostly unnecessary.

Design

As with JFFS, changes to files and directories are "logged" to flash in *nodes*, of which there are two types:

- *inodes*: a header with file metadata, followed by a payload of file data (if any). Compressed payloads are limited to one page.
- *dirent* nodes: directory entries each holding a name and an inode number. Hard links are represented as different names with the same inode number. The special inode number 0 represents an unlink.

As with JFFS, nodes start out as *valid* when they are created, and become *obsolete* when a newer version has been created elsewhere.

Unlike JFFS, however, there is no circular log. Instead, JFFS2 deals in *blocks*, a unit the same size as the erase segment of the flash medium. Blocks are filled, one at a time, with nodes from bottom up. A *clean* block is one that contains only *valid* nodes. A *dirty* block contains at least one *obsolete* node. A *free* block contains no nodes.

The garbage collector runs in the background, turning *dirty* blocks into *free* blocks. It does this by copying *valid* nodes to a new block and skipping *obsolete* ones. That done, it erases the *dirty* block and tags it with a special marker designating it as a *free* block (to prevent confusion if power is lost during an erase operation).

To make wear-levelling more even and prevent erasures from being too concentrated on mostly-static file systems, the garbage collector will occasionally also consume *clean* blocks.

Disadvantages

- All nodes must still be scanned at mount time. This is slow and is becoming an increasingly serious problem as flash devices scale upward into the Gigabyte range.
- Writing many small blocks of data can even lead to negative compression rates, so it is essential for applications to use large write buffers.
- There is no practical way to tell how much usable free space is left on a device since this depends both on how well additional data can be compressed, and the writing sequence.

Chapter 13

LogFS and Lustre (File System)

LogFS

LogFS	
Developer	Jörn Engel
Features	
Supported operating systems	Linux

LogFS is a Linux log-structured and scalable flash filesystem, intended for use on large devices of flash memory. It is written by Jörn Engel and in part sponsored by the CE Linux Forum.

LogFS is included in the mainline Linux kernel in version 2.6.34 released on May 16, 2010.

In contrast to JFFS2, YAFFS and UBIFS, LogFS also provides a (very) basic, slow support for use with block devices like Solid-state drives (SSDs), USB flash drives and memory cards.

History

As of November 2008, LogFS finally is capable of passing all test cases of its test suite - indicating the start of maturing. Still being in heavy development, there were no known installations of LogFS in the field.

Operation

LogFS was motivated by the difficulties of JFFS2 with larger flash-memory drives. LogFS stores the inode tree on the drive; JFFS2 does not, which requires it to scan the entire drive at mount and cache the entire tree in RAM. For larger drives, the scan can take tens of seconds and the tree can take a significant amount of main memory. LogFS

avoids these penalties, but it does do more work while the system is running and uses some of the drive's space for holding the inode tree.

LogFS stores a file's inode tree on the drive, which means on a write to the file, each ancestor node in the tree must be rewritten. This is done by a "wandering tree" update. The lowest node in the tree (i.e., the data) is written first, each node is written ascending the tree, until the root inode is updated. Writing the root last maintains atomicity of the update.

A flash-memory block is the unit for erasures and is usually larger than the file-system block. LogFS handles this disparity by packing multiple file-system blocks into a single flash-memory block. A "sum" entry at the end of the flash-memory block records what data is stored in it. When the flash-memory block has all its file-system blocks moved or deleted, it can be erased and used for new data.

For peak usage of the flash-memory drive, it is necessary to compact data so that flash-memory blocks are full of useful data. This is accomplished by garbage collection. LogFS's garbage collection strategy relies on file data being placed in a certain way into flash-memory blocks: a flash-memory block will hold only file data from the same level in a inode tree. LogFS can garbage collect the top level of the trees using just 1 empty flash-memory block. It can garbage collect the top 2 levels of the trees using 2 empty flash-memory blocks. And can garbage collect all N levels of the trees using N empty flash memory blocks. The algorithm is exponential time in the worst case, but the worst case is rare and the algorithm requires reserving only a handful of flash-memory blocks.

Lustre

Lustre

Developer(s)	Oracle Corporation
Stable release	1.8.4 / August 30, 2010; 3 months ago
Preview release	2.0.0 / August 30, 2010; 3 months ago
Operating system	Linux
Type	Distributed file system
License	GPL

Lustre is a massively parallel distributed file system, generally used for large scale cluster computing. The name Lustre is a portmanteau word derived from **L**inux and

cl**uster**. Available under the GNU GPL, the project aims to provide a file system for clusters of tens of thousands of nodes with petabytes of storage capacity, without compromising speed, security or availability.

Lustre is designed, developed and maintained by Oracle Corporation, by way of its 2010 acquisition of Sun Microsystems, with input from many other individuals and companies. Sun had acquired Cluster File Systems, Inc., including the Lustre file system, in 2007, with the intention of bringing the benefits of Lustre technologies to Sun's ZFS file system and the Solaris operating system.

Lustre file systems are used in computer clusters ranging from small workgroup clusters to large-scale, multi-site clusters. Fifteen of the top 30 supercomputers in the world use Lustre file systems, including the world's fastest supercomputer (as of October 2010), the Tianhe-1A at the National Supercomputing Center in Tianjin, China. Other supercomputers that use the Lustre file system include the second fastest Jaguar supercomputer at Oak Ridge National Laboratory (ORNL) and systems at the National Energy Research Scientific Computing Center located at Lawrence Berkeley National Laboratory (LBNL), Lawrence Livermore National Laboratory (LLNL), Pacific Northwest National Laboratory, Texas Advanced Computing Center and NASA in North America, the largest system in Asia at Tokyo Institute of Technology, and one of the largest systems in Europe at CEA.

Lustre file systems can support up to tens of thousands of client systems, petabytes (PBs) of storage and hundreds of gigabytes per second (GB/s) of I/O throughput. Due to Lustre's high scalability, businesses such as Internet service providers, financial institutions, and the oil and gas industry deploy Lustre file systems in their data centers.

History

The Lustre file system architecture was developed as a research project in 1999 by Peter Braam, who was a Senior Systems Scientist at Carnegie Mellon University at the time. Braam went on to found his own company Cluster File Systems, which released Lustre 1.0 in 2003. Cluster File Systems was acquired by Sun Microsystems in October 2007 (which in turn was acquired by Oracle Corporation in 2010). In November 2008, Braam left Sun Microsystems and founded *ClusterStor* to work on another filesystem, leaving Eric Barton and Andreas Dilger in charge of Lustre architecture and development.

The Lustre file system was first installed for production use in March 2003 on the MCR Linux Cluster at LLNL, one of the largest supercomputers at the time.

Lustre 1.2.0, released in March 2004, provided Linux kernel 2.6 support, a "size glimpse" feature to avoid lock revocation on files undergoing write, and client side data write-back cache accounting (grant).

Lustre 1.4.0, released in November 2004, provided protocol compatibility between versions, InfiniBand network support, and support for extents/mballoc in the ldiskfs on-disk filesystem.

Lustre 1.6.0, released in April 2007, supported mount configuration ("mountconf") allowing servers to be configured with "mkfs" and "mount", supported dynamic addition of *object storage targets* (OSTs), enabled Lustre distributed lock manager (LDLM) scalability on symmetric multiprocessing (SMP) servers, and supported free space management for object allocations.

Lustre 1.8.0, released in May 2009, provided OSS Read Cache, support for different types of OST Pools, adaptive timeouts, and version-based recovery. It also serves as a transition release, being interoperable with both Lustre 1.6 and Lustre 2.0.

Lustre 2.0.0, released in August 2010, provided a rewritten metadata server stack to provide a basis for Clustered Metadata (CMD) to allow distribution of the Lustre metadata across multiple metadata servers, a new Client IO stack (CLIO) for portability to other client operating systems such as MacOS, Windows, and Solaris, and an abstracted Object Storage Device (OSD) back-end for portability to other filesystems such as ZFS.

The Lustre file system and associated open source software has been adopted by many partners. Both Red Hat and SUSE (Novell) offer Linux kernels that work without patches on the client for easy deployment.

In April 2010 Oracle announced it would limit paid support for new Lustre 2.0 deployment to Oracle hardware, or hardware provided by approved third party vendors. Lustre will still be developed and remain freely available under the GPL license to all users, and existing Lustre 1.8 customers will continue to receive support from Oracle, and/or other support providers.

Architecture

A Lustre file system has three major functional units:

- A single *metadata target* (MDT) per filesystem that stores metadata, such as filenames, directories, permissions, and file layout, on the *metadata server* (MDS)

- One or more *object storage servers* (OSSes) that store file data on one or more *object storage targets (OSTs)*. Depending on the server's hardware, an OSS typically serves between two and eight targets, each target a local disk filesystem up to 16 terabytes (TiB) in size. The capacity of a Lustre file system is the sum of the capacities provided by the OSTs.

- *Client(s)* that access and use the data. Lustre presents all clients with a unified namespace for all of the files and data in the filesystem, using standard POSIX

semantics, and allow concurrent and coherent read and write access to the files in the filesystem.

The MDT, OST, and client can be on the same node or on different nodes, but in typical installations, these functions are on separate nodes communicating over a network. Lustre supports several network types, including native Infiniband verbs, TCP/IP on Ethernet and other networks, Myrinet, Quadrics, and other proprietary technologies. Lustre will take advantage of remote direct memory access (RDMA) transfers, when available, to improve throughput and reduce CPU usage.

The storage attached to the servers is partitioned, optionally organized with logical volume management (LVM) and/or RAID, and normally formatted as ext4 file systems. The Lustre OSS and MDS servers read, write, and modify data in the format imposed by these file systems.

An OST is a dedicated filesystem that exports an interface to byte ranges of objects for read/write operations. An MDT is a dedicated filesystem that controls file access and tells clients which object(s) make up a file. MDTs and OSTs currently use an enhanced version of ext4 called *ldiskfs* to store data. In the future, Sun's ZFS/DMU will also be used to store data.

When a client accesses a file, it completes a filename lookup on the MDS. As a result, a file is created on behalf of the client or the layout of an existing file is returned to the client. For read or write operations, the client then passes the layout to a *logical object volume* (LOV), which maps the offset and size to one or more objects, each residing on a separate OST. The client then locks the file range being operated on and executes one or more parallel read or write operations directly to the OSTs. With this approach, bottlenecks for client-to-OST communications are eliminated, so the total bandwidth available for the clients to read and write data scales almost linearly with the number of OSTs in the filesystem.

Clients do not directly modify the objects on the OST filesystems, but, instead, delegate this task to OSSes. This approach ensures scalability for large-scale clusters and supercomputers, as well as improved security and reliability. In contrast, shared block-based filesystems such as Global File System and OCFS must allow direct access to the underlying storage by all of the clients in the filesystem and risk filesystem corruption from misbehaving/defective clients.

Implementation

In a typical Lustre installation on a Linux client, a Lustre filesystem driver module is loaded into the kernel and the filesystem is mounted like any other local or network filesystem. Client applications see a single, unified filesystem even though it may be composed of tens to thousands of individual servers and MDT/OST filesystems.

On some massively parallel processor (MPP) installations, computational processors can access a Lustre file system by redirecting their I/O requests to a dedicated I/O node configured as a Lustre client. This approach was used in the LLNL Blue Gene installation.

Another approach uses the *liblustre* library to provide userspace applications with direct filesystem access. Liblustre is a user-level library that allows computational processors to mount and use the Lustre file system as a client. Using liblustre, the computational processors can access a Lustre file system even if the service node on which the job was launched is not a Lustre client. Liblustre allows data movement directly between application space and the Lustre OSSs without requiring an intervening data copy through the kernel, thus providing low latency, high bandwidth access from computational processors to the Lustre file system directly. Good performance characteristics and scalability make this approach the most suitable for using Lustre with MPP systems. Liblustre is the most significant design difference between Lustre implementations on MPPs such as Cray XT3 and Lustre implementations on conventional clustered computational systems.

Lustre 1.8, released in May 2009, focuses on improving recovery in the face of multiple failures, and storage management via OST Pools.

Lustre technology has been integrated in the HP StorageWorks Scalable File Share product.

Data objects and file striping

In a traditional UNIX disk file system, an inode data structure contains basic information about each file, such as where the data contained in the file is stored. The Lustre file system also uses inodes, but inodes on MDSs point to one or more objects associated with the file rather than to data blocks. These objects are implemented as files on the OSSs. When a client opens a file, the file open operation transfers a set of object pointers and their layout from the MDS to the client, so that the client can directly interact with the OSS node where the object is stored, allowing the client to perform I/O on the file without further communication with the MDS.

If only one object is associated with an MDS inode, that object contains all the data in the Lustre file. When more than one object is associated with a file, data in the file is "striped" across the objects similar to RAID 0. Striping provides significant performance benefits. When striping is used, the maximum file size is not limited by the size of a single target. Also, capacity and aggregate I/O bandwidth scale with the number of OSTs a file is striped over. Also, since the locking of each object is managed independently for each OST, adding more stripes (OSTs) scales the file IO locking capability of the filesystem proportionately.

Networking

In a cluster with a Lustre file system, the system network connecting the servers and the clients is implemented using Lustre Networking (LNET), which provides the communication infrastructure required by the Lustre file system. Disk storage is connected to the Lustre file system MDSs and OSSs using traditional storage area network (SAN) technologies.

LNET supports many commonly-used network types, such as InfiniBand and IP networks, and allows simultaneous availability across multiple network types with routing between them. Remote Direct Memory Access (RDMA) is permitted when supported by underlying networks such as Quadrics Elan, Myrinet, and InfiniBand. High availability and recovery features enable transparent recovery in conjunction with failover servers.

LNET provides end-to-end throughput over Gigabit Ethernet (GigE) networks in excess of 100 MB/s, throughput up to 1.5 GB/s over InfiniBand double data rate (DDR) links, and throughput over 1 GB/s across 10GigE interfaces.

High availability

Lustre file system high availability features include a robust failover and recovery mechanism, making server failures and reboots transparent. Version interoperability between successive minor versions of the Lustre software enables a server to be upgraded by taking it offline (or failing it over to a standby server), performing the upgrade, and restarting it, while all active jobs continue to run, merely experiencing a delay.

Lustre MDSs are configured as an active/passive pair, while OSSs are typically deployed in an active/active configuration that provides redundancy without extra overhead. Often the standby MDS is the active MDS for another Lustre file system, so no nodes are idle in the cluster.

Chapter 14

NILFS and ReiserFS

NILFS

NILFS	
Developer	Nippon Telegraph and Telephone Cyber Space Laboratories
Full name	New Implementation of a Log-structured File System
Structures	
File allocation	B-tree
Limits	
Max file size	8 EiB
Features	
Supported operating systems	Linux, NetBSD

NILFS (**New Implementation of a Log-structured File System**) is a log-structured file system implementation for Linux. It is being developed by Nippon Telegraph and Telephone Corporation (NTT) CyberSpace Laboratories and released under the terms of the GNU General Public License (GPL).

Version 2 of the filesystem, known as NILFS2, is included in Linux kernel 2.6.30. A separate, BSD licensed implementation, currently with read-only support, is included in NetBSD.

Features

Using a copy-on-write technique known as "nothing in life is free", NILFS records all data in a continuous log-like format that is only appended to, never overwritten, an approach that is designed to reduce seek times, as well as minimize the kind of data loss that occurs after a crash with conventional Linux filesystems. For example, data loss occurs on ext3 filesystems when the system crashes during a write operation. When the system reboots, the journal notes that the write did not complete, and any partial data writes are lost.

Some filesystems, like the legacy UFS filesystem used by the Solaris operating system, provide a snapshot feature that prevents such data loss, but filesystem operation must be suspended to use the feature, reducing performance. NILFS, in contrast, can "continuously and automatically [save] instantaneous states of the file system without interrupting service", according to NTT Labs.

The "instantaneous states" that NILFS continuously saves can actually be mounted, read-only, at the same time that the actual filesystem is mounted read-write — a capability useful for data recovery after hardware failures and other system crashes. The "lscp" (list checkpoint) command of an interactive NILFS "inspect" utility is first used to find the checkpoint's address, in this case "2048":

```
# inspect /dev/sda2
...
nilfs> listcp
   1     6 Tue Jul 12 14:55:57 2005 MajorCP|LogiBegin|LogiEnd
2048  2352 Tue Jul 12 14:55:58 2005 MajorCP|LogiEnd
...
nilfs> quit
```

The checkpoint address is then used to mount the checkpoint:

```
# mount -t nilfs -r -o cp=2048 /dev/sda2 /nilfs-cp
# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda2             70332412   8044540  62283776  12% /nilfs
/dev/sda2             70332412   8044540  62283776  12% /nilfs-cp
```

Additional features

- Fast write and recovery times
- Minimal damage to file data and system consistency on hardware failure
    - 32-bit checksums (CRC32) on data and metadata for integrity assurance (per block group, in segment summary)
    - Correctly ordered data and meta-data writes
    - Redundant superblock
- File and inode blocks are managed by a B-tree structure
- Internal data is processed in 64-bit wide word size

- Can create and store huge files (8 EiB)
- Block sizes smaller than page size (e.g. 1 KB or 2 KB)

ReiserFS

ReiserFS	
Developer	Namesys
Full name	ReiserFS
Introduced	2001 (Linux 2.4.1)
Partition identifier	`Apple_UNIX_SVR2` (Apple Partition Map) `0x83` (MBR) `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7` (GPT)
Structures	
Directory contents	B+ tree
File allocation	Bitmap
Limits	
Max file size	1 EiB (8 TiB on 32 bit systems)
Max number of files	2^{32}-3 (~4 billion)
Max filename length	4032 bytes, limited to 255 by Linux VFS
Max volume size	16 TiB
Allowed characters in filenames	All bytes except NULL and `'/'`
Features	
Dates recorded	modification (mtime), metadata change (ctime), access (atime)
Date range	December 14, 1901 - January 18,

	2038
Date resolution	1s
Forks	Extended attributes
File system permissions	Unix permissions, ACLs and arbitrary security attributes
Transparent compression	No
Transparent encryption	No
Supported operating systems	Linux

ReiserFS is a general-purpose, journaled computer file system designed and implemented by a team at Namesys led by Hans Reiser. ReiserFS is currently supported on Linux. Introduced in version 2.4.1 of the Linux kernel, it was the first journaling file system to be included in the standard kernel. ReiserFS is the default file system on the Elive, Xandros, Linspire, GoboLinux, and Yoper Linux distributions. ReiserFS was the default file system in Novell's SUSE Linux Enterprise until Novell decided to move to ext3 on October 12, 2006 for future releases.

Namesys considers ReiserFS (now occasionally referred to as Reiser3) stable and feature-complete and, with the exception of security updates and critical bug fixes, has thus ceased development on it to concentrate on its successor, Reiser4.

Features

At the time of its introduction, ReiserFS offered features that had not been available in existing Linux file systems:

- Metadata-only journaling (also block journaling, since Linux 2.6.8), its most-publicized advantage over what was the stock Linux file system at the time, ext2.
- Online resizing (growth only), with or without an underlying volume manager such as LVM. Since then, Namesys has also provided tools to resize (both grow and shrink) ReiserFS file systems offline.
- Tail packing, a scheme to reduce internal fragmentation. Tail packing, however, can have a significant performance impact. Reiser4 may have improved this by packing tails where it does not hurt performance.

Performance

Compared with ext2 and ext3 in version 2.4 of the Linux kernel, when dealing with files under 4 KiB and with tail packing enabled, ReiserFS may be faster. This was said to be of great benefit in Usenet news spools, HTTP caches, mail delivery systems and other applications where performance with small files is critical. However, in practice news spools use a feature called cycbuf, which holds articles in one large file; fast HTTP caches and several revision control systems use similar approach, nullifying these performance advantages. For email servers, reiserfs was problematic due to semantic problems explained below. Also, ReiserFS had a problem with very fast filesystem aging when compared to other filesystems - in several usage scenarios filesystem performance lowered dramatically with time.

Because ReiserFS still uses the big kernel lock (BKL) — a global kernel-wide lock — in some places, it does not scale very well for systems with multiple cores, as the critical code parts are only ever executed by one core at a time.

Criticism

Some directory operations (including `unlink`(2)) are not synchronous on ReiserFS, which can result in data corruption with applications relying heavily on file-based locks (such as mail transfer agents qmail and Postfix) if the machine halts before it has synchronized the disk.

There are no programs to specifically defragment a ReiserFS file system, although tools have been written to automatically copy the contents of fragmented files hoping that more contiguous blocks of free space can be found. However, a "repacker" tool was planned for the next Reiser4 file system to deal with file fragmentation.

fsck

The tree rebuild process of ReiserFS's fsck has attracted much criticism: If the file system becomes so badly corrupted that its internal tree is unusable, performing a tree rebuild operation may further corrupt existing files or introduce new entries with unexpected contents, but this action is not part of normal operation or a normal file system check and has to be explicitly initiated and confirmed by the administrator.

ReiserFS v3 images should not be stored on a ReiserFS v3 partition (e.g. backups or disk images for emulators) without transforming them (e.g., by compressing or encrypting) in order to avoid confusing the rebuild. Reformatting an existing ReiserFS v3 partition can also leave behind data that could confuse the rebuild operation and make files from the old system reappear. This also allows malicious users to intentionally store files that will confuse the rebuilder. As the metadata is always in a consistent state after a file system check, *corruption* here means that contents of files are merged in unexpected ways with the contained file system's metadata. The ReiserFS successor, Reiser4, fixes this problem.

Earlier issues

ReiserFS in versions of the Linux kernel before 2.4.16 were considered unstable by Namesys and not recommended for production use, especially in conjunction with NFS.

Early implementations of ReiserFS (prior to that in Linux 2.6.2) were also susceptible to out-of-order write hazards. But the current journaling implementation in ReiserFS is now on par with that of ext3's "ordered" journaling level.

Novell / SuSE move away from ReiserFS to ext3

Jeff Mahoney of SuSE wrote a post on Sep 14 2006 proposing to move from ReiserFS to ext3 for the default installation file system. Some reasons he mentioned were scalability, "performance problems with extended attributes and ACLs", "a small and shrinking development community", and that "Reiser4 is not an incremental update and requires a reformat, which is unreasonable for most people." On October 4 he wrote a response comment on a blog in order to clear up some issues. He wrote that his proposal for the switch was unrelated to Reiser's "legal troubles" (i.e. Hans Reiser murdering his wife, and his subsequent conviction) Mahoney wrote he "was concerned that people would make a connection where none existed" and that "the timing is entirely coincidental and the motivation is unrelated."

On Oct 12, 2006, Novell similarly announced that SuSE Linux Enterprise would switch from ReiserFS to ext3.

Design

ReiserFS stores file metadata ("stat items"), directory entries ("directory items"), inode block lists ("indirect items"), and tails of files ("direct items") in a single, combined B+ tree keyed by a universal object ID. Disk blocks allocated to nodes of the tree are "formatted internal blocks". Blocks for leaf nodes (in which items are packed end-to-end) are "formatted leaf blocks". All other blocks are "unformatted blocks" containing file contents. Directory items with too many entries or indirect items which are too long to fit into a node spill over into the right leaf neighbour. Block allocation is tracked by free space bitmaps in fixed locations.

By contrast, ext2 and other Berkeley FFS-like file systems of that time simply used a fixed formula for computing inode locations, hence limiting the number of files they may contain. Most such file systems also store directories as simple lists of entries, which makes directory lookups and updates linear time operations and degrades performance on very large directories. The single B+ tree design in ReiserFS avoids both of these problems due to better scalability properties.

Reiser4

Reiser4	
Developer	Namesys
Full name	Reiser4
Introduced	2004 (Linux)
Partition identifier	`Apple_UNIX_SVR2` (Apple Partition Map) `0x83` (MBR) `EBD0A0A2-B9E5-4433-87C0-68B6B72699C7` (GPT)
Structures	
Directory contents	Dancing B*-tree
Limits	
Max file size	8 TiB on x86
Max filename length	3976 bytes
Allowed characters in filenames	All bytes except NUL and '/'
Features	
Dates recorded	modification (mtime), metadata change (ctime), access (atime)
Date range	64-bit timestamps
Forks	Extended attributes
File system permissions	Unix permissions, ACLs and arbitrary security attributes
Transparent compression	Yes
Transparent encryption	No

Data deduplication	No
Supported operating systems	Linux

Reiser4 is a computer file system, successor to the ReiserFS file system, developed from scratch by Namesys and sponsored by DARPA as well as Linspire. It is named after its former lead developer Hans Reiser, although the continued development of Reiser4 is uncertain, following the murder by Hans Reiser of his wife.

Features

Some of the goals of the Reiser4 file system are:

- More efficient journaling through wandering logs
- More efficient support of small files, in terms of disk space and speed through block suballocation
- Faster handling of directories with large numbers of files
- Flexible plugin infrastructure (through which special metadata types, encryption and compression will be supported)
- Dynamically optimized disk-layout through allocate-on-flush (also called delayed allocation in XFS)
- Transaction support

Some of the more advanced Reiser4 features (such as user-defined transactions) are also not available because of a lack of a VFS API for them.

At present Reiser4 lacks a few standard file system features, such as an online repacker (similar to the defragmentation utilities provided with other file systems). The creators of Reiser4 say they will implement these later, or sooner if someone pays them to do so.

Performance

Reiser4 uses B*-trees in conjunction with the dancing tree balancing approach, in which underpopulated nodes will not be merged until a flush to disk except under memory pressure or when a transaction completes. Such a system also allows Reiser4 to create files and directories without having to waste time and space through fixed blocks.

As of 2004, synthetic benchmarks performed by Namesys show that Reiser4 is 10 to 15 times faster than its most serious competitor ext3 working on files smaller than 1 KiB. Namesys's benchmarks suggest it is typically twice the performance of ext3 for general-purpose filesystem usage patterns. Other benchmarks show results of Reiser4 being slower on many operations.

Integration with Linux

As of 2010, Reiser4 has not yet been merged into the mainline Linux kernel and consequently is still not supported on many Linux distributions; however, its predecessor ReiserFS v3 has been widely adopted. Reiser4 is also available from Andrew Morton's -mm kernel sources. Linux kernel developers claim that Reiser4 does not follow Linux coding standards, but Hans Reiser suggested political reasons.

History of Reiser4

Hans Reiser was convicted of murder on April 28, 2008, leaving the future of Reiser4 uncertain. After his arrest, employees of Namesys assured they would continue to work and that the events would not slow down the software development in the immediate future. In order to afford increasing legal fees, Hans Reiser announced on December 21, 2006 that he was going to sell Namesys; as of March 26, 2008, it has not been sold, although the website is unavailable. In January 2008, Edward Shishkin, an employee of and programmer for Namesys, was quoted in a CNET interview saying that "Commercial activity of Namesys has stopped." Shishkin and others continued the development of Reiser4, making source code available from Shishkin's web site, later relocated to kernel.org. Since 2008, Namesys employees have received 100% of their sponsored funding from DARPA which is why Edward said 'commercial funding has stopped'.

Future of Reiser4

Although kernel developer Theodore Ts'o has suggested btrfs as an alternative for those interested in the design ideas of Reiser4, Reiser4 development still continues, delivering patches via kernel.org.

In a mailing list post from July 2009, Edward Shishkin wrote that in the coming autumn, they would start exploring the opportunity of getting Reiser4 into the main Linux kernel. In a November 2009 interview to Phoronix he said he is going to publish a plug-in design document for independent expert review. He is currently aiming for January 2011.

Chapter 15

XFS

XFS	
Developer	Silicon Graphics Inc.
Full name	XFS
Introduced	1994 (IRIX v5.3)
Structures	
Directory contents	B+ trees
File allocation	B+ trees
Limits	
Max file size	8 exabytes (minus 1 byte)
Max filename length	255 bytes
Max volume size	16 exabytes
Allowed characters in filenames	All bytes except NUL ('\0') and '/'
Features	
Dates recorded	Yes
Date resolution	1ns
Attributes	Yes
File system permissions	Yes
Transparent compression	No

Transparent encryption	No (provided at the block device level)
Data deduplication	No
Supported operating systems	IRIX, Linux, FreeBSD (experimental)

XFS is a high-performance journaling file system created by Silicon Graphics, originally for their IRIX operating system and later ported to the Linux kernel. XFS is particularly proficient at handling large files and at offering smooth data transfers.

History

Development of XFS was started by Silicon Graphics in 1993, with first deployment being seen on IRIX 5.3 in 1994. The filesystem was released under the GNU General Public License in May 2000, and ported to Linux, with the first distribution support appearing in 2001. It is available in almost all Linux distributions today.

XFS was first merged into mainline Linux in version 2.4, making it almost universally available on Linux systems. Installation programs for the Arch, Debian, Fedora, openSUSE, Gentoo, Kate OS, Mandriva, Slackware, Ubuntu, VectorLinux and Zenwalk Linux distributions all offer XFS as a choice of filesystem, but few of these let the user create XFS for the /boot filesystems due to deficiencies and unpredictable behavior in GRUB, often the default bootloader. FreeBSD gained read-only support for XFS in December 2005 and in June 2006 experimental write support was introduced; however this is supposed to be used only as an aid in migration from Linux, not to be used as a "main" filesystem. Red Hat Enterprise Linux 5.4 64 bit version has all needed kernel support but does not include command line tools for creating and using the system (CentOS tools are known to work in some extent). The motivation is that the system port may not yet be stable enough.

Specifications

Capacity

XFS is a 64-bit file system. It supports a maximum file system size of 8 exbibytes minus one byte, though this is subject to block limits imposed by the host operating system. On 32-bit Linux systems, this limits the file and file system sizes to 16 tebibytes.

Journaling

Journaling is an approach to guaranteeing file system consistency even in presence of power failures or system crashes. XFS provides journaling for file system metadata, where file system updates are first written to a serial journal before the actual disk blocks

are updated. The journal is a circular buffer of disk blocks that is never read in normal filesystem operation. The XFS journal is limited to a maximum size of both 64k blocks and 128MB with the minimum size dependent upon a calculation of the filesystem block size and directory block size. Placing the journal on an external device larger than the maximum journal size will cause the extra space to be unused. It can be stored within the data section of the filesystem (an internal log), or on a separate device to minimize disk contention. On XFS the journal contains “logical” entries that describe at a high level what operations are being performed (as opposed to a “physical” journal that stores a copy of the blocks modified during each transaction). Journal updates are performed asynchronously to avoid incurring a performance penalty. In the event of a system crash, operations immediately prior to the crash can be redone using data in the journal, which allows XFS to retain file system consistency. Recovery is performed automatically at file system mount time, and the recovery speed is independent of the size of the file system. Where recently modified data has not been flushed to disk before a system crash, XFS ensures that any unwritten data blocks are zeroed on reboot, obviating any possible security issues arising from residual data (as far as access through the filesystem interface is concerned, as distinct from accessing the raw device or raw hardware).

Allocation groups

XFS filesystems are internally partitioned into *allocation groups*, which are equally sized linear regions within the file system. Files and directories can span allocation groups. Each allocation group manages its own inodes and free space separately, providing scalability and parallelism — multiple threads and processes can perform I/O operations on the same filesystem simultaneously. This architecture helps to optimise parallel I/O performance on multiprocessor or multicore systems, as metadata updates are also parallelizable. The internal partitioning provided by allocation groups can be especially beneficial when the file system spans multiple physical devices, allowing for optimal usage of throughput of the underlying storage components.

Striped allocation

If an XFS filesystem is to be created on a striped RAID array, a *stripe unit* can be specified when the file system is created. This maximises throughput by ensuring that data allocations, inode allocations and the internal log (journal) are aligned with the stripe unit.

Extent based allocation

Blocks used in files stored on XFS filesystems are managed with variable length extents where one extent describes one or more contiguous blocks. This can shorten the list considerably compared to file systems that list all blocks used by a file individually. Also many file systems manage space allocation with one or more block oriented bitmaps — in XFS these structures are replaced with an extent oriented structure consisting of a pair of B+ trees for each filesystem allocation group (AG). One of the B+ trees is indexed by the length of the free extents, while the other is indexed by the starting block of the free

extents. This dual indexing scheme allows for highly efficient location of free extents for file system operations.

Variable block sizes

The file system block size represents the minimum allocation unit. XFS allows file systems to be created with block sizes ranging between 512 bytes and 64 kilobytes, allowing the file system to be tuned for the expected use. When many small files are expected a small block size would typically maximize capacity, but for a system dealing mainly with large files, a larger block size can provide a performance advantage.

Delayed allocation

XFS makes use of lazy evaluation techniques for file allocation. When a file is written to the buffer cache, rather than allocating extents for the data, XFS simply reserves the appropriate number of file system blocks for the data held in memory. The actual block allocation occurs only when the data is finally flushed to disk. This improves the chance that the file will be written in a contiguous group of blocks, reducing fragmentation problems and increasing performance.

Sparse files

XFS provides a 64-bit sparse address space for each file, which allows both for very large file sizes, and for *holes* within files for which no disk space is allocated. As the file system uses an extent map for each file, the file allocation map size is kept small. Where the size of the allocation map is too large for it to be stored within the inode, the map is moved into a B+ tree which allows for rapid access to data anywhere in the 64-bit address space provided for the file.

Extended attributes

XFS provides multiple data streams for files through its implementation of extended attributes. These allow the storage of a number of name/value pairs attached to a file. Names are null-terminated printable character strings of up to 256 bytes in length, while their associated values can contain up to 64 KB of binary data. They are further subdivided into two namespaces, `root` and `user`. Extended attributes stored in the root namespace can be modified only by the superuser, while attributes in the user namespace can be modified by any user with permission to write to the file. Extended attributes can be attached to any kind of XFS inode, including symbolic links, device nodes, directories, etc. The `attr` program can be used to manipulate extended attributes from the command line, and the `xfsdump` and `xfsrestore` utilities are aware of them and will back up and restore their contents. Most other backup systems are not aware of extended attributes.

Direct I/O

For applications requiring high throughput to disk, XFS provides a direct I/O implementation that allows non-cached I/O directly to userspace. Data is transferred between the application's buffer and the disk using DMA, which allows access to the full I/O bandwidth of the underlying disk devices.

Guaranteed-rate I/O

The XFS guaranteed-rate I/O system provides an API that allows applications to reserve bandwidth to the filesystem. XFS will dynamically calculate the performance available from the underlying storage devices, and will reserve bandwidth sufficient to meet the requested performance for a specified time. This feature is unique to the XFS file system. Guarantees can be *hard* or *soft*, representing a trade off between reliability and performance, though XFS will only allow *hard* guarantees if the underlying storage subsystem supports it. This facility is most used by real-time applications, such as video-streaming.

DMAPI

XFS implements the DMAPI interface to support Hierarchical Storage Management. As of October 2010, the Linux implementation of XFS supports the required on-disk metadata for DMAPI implementation, but the kernel support is reportedly not in a usable state. For some time, SGI hosted a kernel tree which included the DMAPI hooks, but this support has not been adequately maintained, though kernel developers have stated an intention to bring it up to date.

Snapshots

XFS does not provide direct support for snapshots, as it expects the snapshot process to be implemented by the volume manager. Taking a snapshot of an XFS filesystem involves freezing I/O to the filesystem using the `xfs_freeze` utility, having the volume manager perform the actual snapshot, and then unfreezing I/O to resume normal operations. The snapshot can then be mounted read-only for backup purposes. XFS releases on IRIX incorporated an integrated volume manager called XLV. This volume manager has not been ported to Linux and XFS works with standard LVM instead. In recent Linux kernels, the `xfs_freeze` functionality is implemented in the VFS layer, and happens automatically when the Volume Manager's snapshot functionality is invoked. This was once a valuable advantage as Ext3 system could not be suspended and volume manager was unable to create a consistent 'hot' snapshot to backup a heavily busy database. Fortunately this is no longer the case. Since Linux 2.6.29 ext3, ext4, gfs2 and jfs have the freeze feature as well.

Online defragmentation

Although the extent-based nature of XFS and the delayed allocation strategy it uses significantly improves the file system's resistance to fragmentation problems, XFS provides a filesystem defragmentation utility (`xfs_fsr`, short for XFS filesystem reorganizer) that can defragment the files on a mounted and active XFS filesystem.

Online resizing

XFS provides the `xfs_growfs` utility to perform online resizing of XFS file systems. XFS filesystems can be grown provided there is remaining unallocated space on the device holding the filesystem. This feature is typically used in conjunction with volume management, as otherwise the partition holding the filesystem will need enlarging separately. XFS partitions cannot (as of August 2010) be shrunk in place, although several possible workarounds have been discussed.

Native backup/restore utilities

XFS provides the `xfsdump` and `xfsrestore` utilities to aid in backup of data held on XFS file systems. The `xfsdump` utility backs up an XFS filesystem in inode order, and in contrast to traditional UNIX file systems which must be unmounted before dumping to guarantee a consistent dump image, XFS file systems can be dumped while the file system is in use. This is not the same as a snapshot since files are not frozen during the dump. XFS dumps and restores are also resumable, and can be interrupted without difficulty. The multi-threaded operation of `xfsdump` provides high performance of backup operations by splitting the dump into multiple streams, which can be sent to different dump destinations. The multi stream capabilities have not been fully ported to Linux yet, however.

Atomic disk quotas

Quotas for XFS filesystems are turned on when initially mounted; this fixes a race window that is present with most other filesystems that first require to be mounted and where no quotas are enforced until quotaon(8) is called.

Performance considerations

Write barriers

XFS filesystems mount by default with "write barriers" enabled. This feature will cause the write back cache of the underlying storage device to be flushed at appropriate times, particularly on write operations to the XFS log. This feature is intended to assure filesystem consistency, and its implementation is device specific - not all underlying hardware will support cache flush requests. When an XFS filesystem is used on a logical device provided by a hardware RAID controller with battery backed cache, this feature can cause significant performance degradation, as the filesystem code is not aware that

the cache is nonvolatile, and if the controller honors the flush requests, data will be written to physical disk more often than necessary. To avoid this problem, where the data in the device cache is protected from power failure or other host problems, the filesystem should be mounted with the "nobarrier" option.

Journal placement

By default, XFS filesystems are created with an "internal" log, which places the filesystem journal on the same block device as the filesystem data. As filesystem writes are preceded by metadata updates to the journal, this can be a cause of disk contention. Under most workloads, the level of contention caused is too low to impact performance, but random-write heavy workloads, such as those seen on busy database servers, can suffer from sub-optimal performance as a result of this I/O contention. An additional factor which can increase the severity of this problem is that writes to the journal are committed synchronously – they must complete successfully before the associated write operation can begin.

Where optimum filesystem performance is required, XFS provides the option of placing the log on a separate physical device, with its own I/O path. This requires little physical space, and if a low-latency path can be provided for synchronous writes, it can provide significant performance enhancements to the operation of the filesystem. The required performance characteristics make this a suitable candidate for the use of an SSD device, or a RAID system with write-back cache, though the latter can reduce data safety in the event of power problems. The use of an external log simply requires the filesystem to be mounted with the `logdev` option, indicating a suitable journal device.

Disadvantages

- An XFS file system cannot be shrunk.
- Creation and deletion of directory entries can be a much slower metadata operation than other file systems, depending on configuration.