Unix

UNIX was developed at Bell Labs in the early 1970s.

The first version was written by Ken Thompson in assembler for the PDP-7 minicomputer.

This was soon followed by a version for the PDP-11, written in a new language called C that was devised and implemented by Dennis Ritchie.

In 1974, Ritchie and Ken Thompson published a landmark paper about UNIX (Ritchie and Thompson, 1974).

For the work described in this paper they were later given the prestigious ACM Turing Award (Ritchie, 1984, Thompson, 1984).

The publication of this paper stimulated many universities to ask Bell Labs for a copy of UNIX. Since Bell Labs' parent company, AT&T, was a regulated monopoly at the time and was not permitted to be in the computer business, it had no objection to licensing UNIX to universities for a modest fee.

249

Unix

The PDP-11 was the computer of choice at nearly all university computer science departments, and the original operating systems that came with the PDP-11 were widely regarded as being dreadful.

UNIX quickly filled the void, not least because it was supplied with the complete source code, so people could, and did, tinker with it endlessly.

One of the many universities that acquired UNIX early on was the University of California at Berkeley.

Because the complete source code was available, Berkeley was able to modify the system substantially.

Foremost among the changes was a port to the VAX minicomputer and the addition of paged virtual memory, the extension of file names from 14 characters to 255 characters, and the inclusion of the TCP/IP networking protocol, which is now used on the Internet (largely due to the fact that it was in Berkeley UNIX).

250

Unix

While Berkeley was making all these changes, AT&T itself continued to develop UNIX, leading to System III in 1982 and then System V in 1984.

By the late 1980s, two different, and quite incompatible, versions of UNIX were in widespread use: Berkeley UNIX and System V.

This split in the UNIX world.

In addition, no standards for binary program formats, inhibited the commercial success of UNIX because it was impossible for software vendors to write and package UNIX programs with the expectation that they would run on any UNIX system (as was routinely done then with MS-DOS).

Eventually, a standard called **POSIX** (**Portable Operating System-IX**) was created by the IEEE Standards Board. It later became an International Standard.

251

Unix

The standard is divided into many parts, each one covering a different area of UNIX.

The first part, P1003.1, defines the system calls,
the second part, P1003.2, defines the basic utility programs,
and so on.

The P1003.1 standard defines about 60 system calls that all conformant systems must support. These are the basic calls for reading and writing files, creating new processes, etc.

Nearly all UNIX systems now support the P1003.1 system calls. However many UNIX systems also support extra system calls, especially those defined by System V and/or those in Berkeley UNIX. Typically these add up to 200 system calls.

252

Unix

In 1987, Tanenbaum released the source code for a tiny version of UNIX, called MINIX, for use at universities.

Linus Torvalds a students studying MINIX in Helsinki wrote his own clone of MINIX, which he called called **Linux.**

Many operating systems running today on ARM platforms are based on Linux.

Both MINIX and Linux are POSIX conformant.

A rough breakdown of the Linux system calls by category is shown below.

The file-management and directory-management system calls are the largest and the most important categories.

253

Unix

Category	Some examples
File management	Open, read, write, close, and lock files
Directory management	Create and delete directories; move files around
Process management	Spawn, terminate, trace, and signal processes
Memory management	Share memory among processes; protect pages
Getting/setting parameters	Get user, group, process ID; set priority
Dates and times	Set file access times; use interval timer; profile execution
Networking	Establish/accept connection; send/receive message
Miscellaneous	Enable accounting; manipulate disk quotas; reboot the system

A rough breakdown of the Linux system calls

254

Unix

One area that is largely due to Berkeley UNIX is networking.

Berkeley invented the concept of a **socket**, which is the endpoint of a network connection.

It is possible for a UNIX process to create a socket, attach to it, and establish a connection to a socket on a distant machine.

Over this connection it can then exchange data in both directions, typically using the TCP/IP protocol.

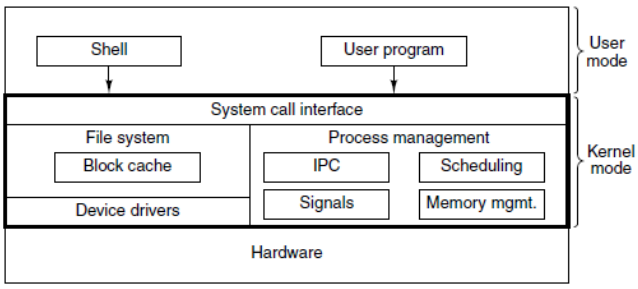
Networking technology has been in UNIX for decades and is very stable and mature, consequently, a substantial fraction of the servers on the Internet run UNIX.

255

Unix

There are many implementations of UNIX; it is difficult to say much about the structure of the operating system - each implementation is somewhat different.

The general structure of a typical Unix system, however, is:



At the bottom, there is a layer of device drivers that isolate the file system from the bare hardware.

256

Unix

Originally, each device driver was written as an independent entity, separate from all the others.

This arrangement led to a lot of duplicated effort – all drivers typically deal with flow control, error handling, priorities and separating data from control.

This observation led Dennis Ritchie to develop a framework called **streams** for writing drivers in a modular way.

A stream establishes a two-way connection between a user process and a hardware device via a number of modules.

The user process pushes data into the stream, which then is processed or transformed by each module along the way until it gets to the hardware.

The process occurs in reverse for data coming from the hardware to a user process.

257

Unix

The File System sits on top of the device drivers. It manages file names, directories, disk block allocation, protection, and much more.

The file system uses a **block cache**, for holding the blocks most recently read in from disk, in case they are needed again soon.

A variety of file systems have been realized over the years, including the Berkeley fast file system and the log-structured file system.

The process-management portion of the UNIX kernel handles IPC (InterProcess Communication) - allowing processes to communicate with one another and to synchronize to avoid race conditions. A variety of mechanisms are provided.

The process-management code also handles process scheduling, which is based on priorities.

Signals, a form of (asynchronous) software interrupt, are also managed in the kernel as is **Memory management**..

258

Unix

From the beginning, UNIX has tried to be a small system, in order to enhance reliability and performance.

The first versions of UNIX were entirely text based, using terminals that could display 24 or 25 lines of 80 ASCII characters.

The user interface was handled by a user-level program called the **shell**, which offered a command-line interface. Since the shell was not part of the kernel, adding new shells to UNIX was easy, and over time a number of increasingly sophisticated ones were invented.

Later on, when graphics terminals came into existence, a windowing system for UNIX, called **X Windows**, was developed at M.I.T.

Still later, a full-fledged **GUI** (**Graphical User Interface**), called **Motif**, was put on top of **X Windows**. These GUIs eventually developed into full-blown desktop environments with window management, productivity tools, and utilities.

Examples of these desktop environments include **GNOME** and **KDE**. In keeping with the UNIX philosophy of having a small kernel, nearly all the code of X Windows and its accompanying GUIs run in user mode, outside the kernel.

259

Windows

The original IBM PC (1981) came with a16-bit, single-user, command-line-oriented operating system called MSDOS 1.0

Two years later, the more powerful MS-DOS 2.0 was introduced.

It contained a command-line processor (shell), with a number of features borrowed from UNIX.

When IBM released the 286-based PC/AT in 1984, it came equipped with MS-DOS 3.0.

MS-DOS continued to acquire new features, but it was still a command-line-oriented system.

260

Windows

To compete with Mac, Microsoft decided to give MSDOS a graphical user interface that it called **Windows**.

The first three versions of Windows, culminating in **Windows 3.x**, were not true operating systems but graphical user interfaces on top of MS-DOS, which was still in control of the machine.

All programs ran in the same address space and a bug in any one of them could crash the entire system.

Windows 95 (1995) still used MS-DOS (now version, 7.0).

Together, Windows 95 and MS-DOS 7.0 contained most of the features of a full-blown operating system, including virtual memory, process management, and multiprogramming.

However, Windows 95 was not a full 32-bit program. It contained large chunks of old 16-bit code (and some 32-bit code) and still used the MS-DOS file system, with nearly all its limitations.

261

Windows

The only major changes to the file system were the addition of long file names in place of the 8 + 3 character file names allowed in MS-DOS and the ability to have more than 65,536 blocks on a disk.

Windows 98 (1998) still contained MS-DOS (now called version 7.1) and running 16-bit code.

Although a bit more functionality migrated from the MS-DOS part to the Windows part, and a disk layout suitable for larger disks was now standard, under the hood, Windows 98 was not very different from Windows 95.

The main difference was the user interface, which integrated the desktop with its Internet Explorer. This close connection resulted the U.S. Dept. of Justice suing Microsoft, claiming that it was operating an illegal monopoly.

Windows 98 was followed by the short-lived **Windows Millennium Edition** (ME), which was a slightly improved Windows 98

262

Windows

Windows New Technology, or **Windows NT** followed and was a new 32-bit operating system, written from the ground up.

It was initially hyped as the replacement for all other operating systems for Intel-based PCs (and the MIPS PowerPC chips). It was slow to catch on and was later used in the upper end of the market, on large servers.

A second version of NT was **called Windows 2000** and became the mainstream version, also for the desktop market.

The successor to Windows 2000 was **Windows XP**, but the changes here were relatively minor (better backward compatibility and a few more features).

In 2007, the follow-up **Windows Vista** was released. Vista implemented many graphical enhancements over Windows XP, and it added many new user applications, such as a media center.

263

Windows

Vista's adoption was slow because of its poor performance and high resource demands.

Two years later, **Windows 7** was released, which by all accounts is a tuned-up version of Windows Vista.

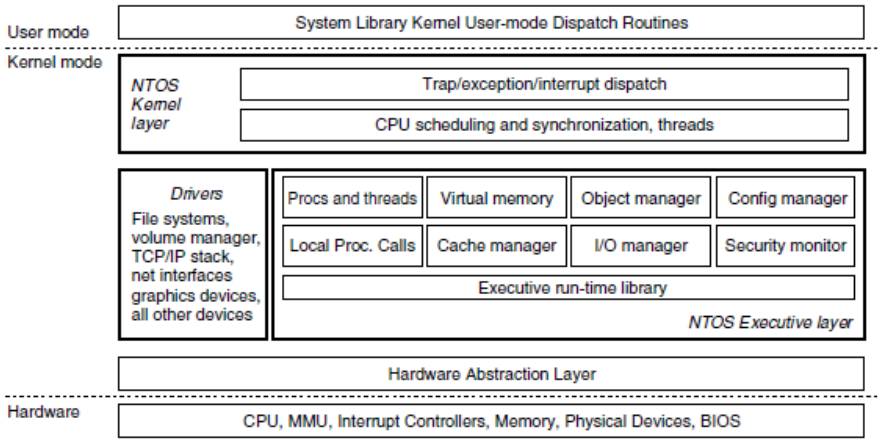
Windows 7 ran much better on older hardware, and it required significantly less hardware resources.

Windows 7 was sold in six different versions. These versions are nearly identical and differ primarily in focus, advanced features, and optimizations made.

264

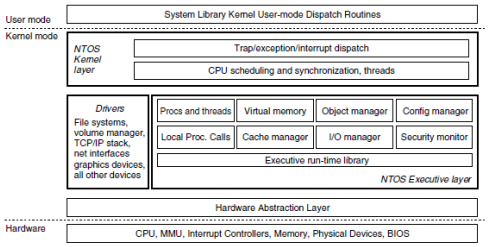
Windows

The internal structure of Windows 7 is shown.



265

Windows



It consists of a number of modules that are structured in layers.

Each module has some particular function and a well-defined interface to the other modules.

Nearly all the modules are written in C, although part of the graphics device interface is written in C++ and some parts of the lowest layers are written in assembly language.

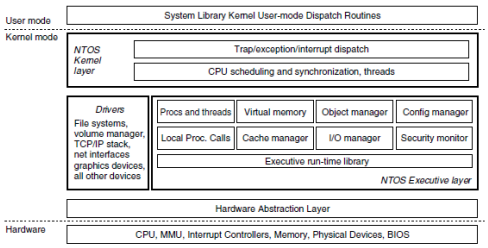
At the bottom is a thin layer called the **Hardware Abstraction Layer**. It presents the rest of the operating system with an abstraction of the hardware devices.

Among the devices are off-chip caches, timers, I/O buses, interrupt controllers, and DMA controllers.

Exposing these to the rest of the operating system in an abstract form, makes it easier to port Windows 7 to other hardware platforms, since most of the modifications required are concentrated in one place.

266

Windows



Above the HAL, the code is divided into two major parts, the **NTOS Executive layer** and the **Windows drivers**, which includes the file systems, networking, and graphics code.

The **Executive** manages the fundamental abstractions used in Windows 7, including threads, processes, virtual memory, kernel objects, and configurations.

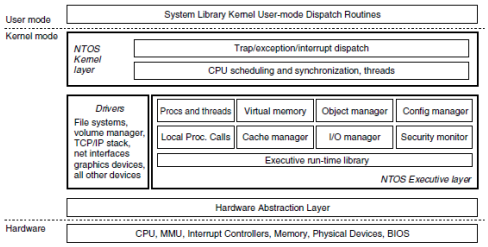
Also here are the managers for local procedure calls, the file cache, I/O, and security.

On top of that is the **kernel layer**. All of this code runs in protected kernel mode.

The kernel layer handles trap and exception handling, as well as scheduling and synchronization.

267

Windows



Outside the kernel are the user programs and the system library used to interface to the operating system.

In contrast to UNIX systems, Microsoft does not encourage user programs to make direct system calls.

Instead users are expected to call procedures in the library. To provide standardization across different versions of Windows.

Microsoft defined a set of calls called the **Win32 API** (**Application Programming Interface**).

These are library procedures that either make system calls to do the work, or, in some case, do the work in the user-space library procedure.

268

Windows

The Win32 API philosophy is completely different from the UNIX philosophy.

In UNIX, system calls are all publicly known and form a minimal interface: removing even one of them would reduce the functionality of the operating system.

The Win32 philosophy is to provide a very comprehensive interface, often with three or four ways of doing the same thing.