Lucy <whoislucy(at)gmail.com>

Inside the Mac OS X Kernel

Debunking Mac OS Myths

24th Chaos Communication Congress 24C3, Berlin 2007

Many buzzwords are associated with Mac OS X: Mach kernel, microkernel, FreeBSD kernel, C++, 64 bit, UNIX... and while all of these apply in some way, “XNU”, the Mac OS X kernel is neither Mach, nor FreeBSD-based, it's not a microkernel, it's not written in C++ and it's not 64 bit - but it is Open Source (with reservations) and it's UNIX... but just since recently.

This paper intends to clear up the confusion by presenting details of the Mac OS X kernel architecture, its components Mach, BSD and I/O-Kit, what's so different and special about this design, and what the special strengths of it are.

History

Unlike many other operating systems, the design of Mac OS X has never been strictly planned and implemented from scratch, instead, it is the result of code from very different sources put together over the last decades.

Mac OS

Mac OS started its life in 1984 on the original 128KB Macintosh as a mouse-operated graphical operating system that, due to memory constraints, did not support multitasking. It wasn't until 1988 that Mac OS supported a very simple form of cooperative multitasking (“MultiFinder”). In the mid-90s, Apple ended up having a ten year old code base designed for a single-tasking system on a Motorola 68000 that now ran on PowerPC CPUs. Parts of the kernel code ran in a 68K emulator, and it still did not support memory protection. There was no way to compete even with Windows 95, which is why Apple started the Copland project in 1994 in order to design and implement a new and modern operating system that would have the Mac OS API and user interface - much like Microsoft did with Windows NT. But although Copland had been heavily advertised with developers, programming books had been published and Betas had been given out, the pieces of Copland never fit together, and the unbearably unstable operating system was scrapped in 1996.

Mac OS Successor

As Apple was in bitter need of a successor for Mac OS, they decided to buy an operating system and build Mac OS compatibility into it. Despite negotiations with the company behind BeOS, Apple finally decided to buy NEXT, the company Steve Jobs had founded just after having left Apple in 1985, and to convert NEXTSTEP/OpenStep into the next Mac OS: Mac OS X.

Mach

The NEXTSTEP operating system was heavily based on Mach. Mach was an operating system project at the Carnegie Mellon University that was started in 1985 in response to the ever-increasing complexity of the UNIX and BSD kernels. As one of the first microkernels, it only included code for memory management (address spaces, tasks), scheduling (threads; a concept unknown to UNIX at that time) and inter-process communication (IPC) - all other functionality typically found in an operating system kernel, like filesystems, networking, security and device drivers, had to be implemented in so-called “servers” in user space. This could be a very big plus for reliability, since a crash in a driver didn't necessarily bring the system down, as well as maintainability, since it imposed strict rules on the interface between the core kernel functionality and the userland servers. Unlike in UNIX, operating system components couldn't just call each other arbitrarily (“The big mess” - Tanenbaum). Another advantage of a microkernel like Mach is the possibility to have several personalities, each of which is a set of userspace servers. This way, a Mach-based system could, for example, run UNIX and Windows applications at the same time. Having a minimal piece of code running in privileged mode that abstracts the hardware and allows different operating systems to run on top of it is basically the same approach implemented by virtualization today. But the typical configuration of a Mach operating system was to have a single BSD server in user mode, i.e. the majority of the

BSD kernel with memory management and scheduling stripped out, and process management built on top of Mach tasks.

The problem with the Mach design was that the kernel was slower than a traditional monolithic kernel because of the extra kernel/user context switches when a server communicated with the kernel or servers communicated with each other. On a monolithic kernel, these were just simple function calls. The simplest solution for this problem is "co-location": The personality servers run in kernel mode, and communication is fast again. While it somewhat defeats the original idea of a microkernel, it still has the advantage of well-partitioned kernel components and a more modern core kernel: The Mach memory management code was later integrated into BSD.

NEXTSTEP

NEXTSTEP, which was released in a 1.0 version in 1989, chose to go with this design. NEXT had removed the core kernel parts from the 4.3BSD kernel and layered it on top of Mach, in kernel mode. This way, NEXT was many years ahead of the competition with NEXTSTEP being the first desktop/GUI operating system that supported preemptive multitasking, memory protection and UNIX compatibility. At first NEXTSTEP only ran on their own Motorola 68K-based machines, but was later ported to SPARC, PA-RISC and i386, when NEXT started licensing it under the name "OpenStep" to other hardware manufacturers, so it was highly portable. When Apple acquired NEXT in 1997, they added PowerPC support and removed support for all architectures other than i386; the latter would serve as the fallback solution when Apple switched from PowerPC to i386 in 2005/2006.

Rhapsody and OS X

With Apple's acquisition of OpenStep, many more changes were made to the operating system which now had the interim name "Rhapsody": They replaced the "DriverKit" driver model with the new "I/O-Kit" system, updated Mach 2.5 with the Mach 3.0 codebase, updated the BSD part with 4.4BSD and FreeBSD code and added support for the HFS filesystem and Apple networking protocols to the kernel. In userland, Mac OS X is pretty much NEXTSTEP/OpenStep, with the native "NS" API renamed to Cocoa, the Mac OS 9 API "Toolbox" ported as a compatibility API (now named "Carbon"), "carbonized" versions of the OS 9 Finder and QuickTime technologies, plus a VMware-like Virtual Machine called Blue-Box ("Classic") that runs OS 9 and its applications unmodified.

Architecture

The Mac OS X kernel, named "XNU" ("X is not UNIX") consists of three main components: Mach, BSD and I/O-Kit.

Mach

Being the only operating system that still uses Mach code (not counting GNU/HURD), Mac OS X has evolved from the original code base quite a bit, but the architecture is basically unchanged. Mach ("osfmk" in the kernel source tree, which stands for "OSF microkernel") calls address spaces "tasks", and one task can contain zero or more threads. Being policy-free, there is little information associated with a task, so, for example, there is no UNIX-style current working directory or environment associated with it. While there are few surprises in the memory management code compared to other modern operating systems, the key distinctive feature of Mach is Mach Messaging. A task can have any number of "ports", which are interprocess communication (IPC) endpoints. One task can subsequently send a message from its originating port to its peer port, and Mach will take care of security, enqueueing, dequeueing, network opacity (ports can be on different machines) and, if necessary, byte swapping. For programming convenience, the Mach Interface Generator ("MIG") can generate stub code from interface definitions, so that two processes can talk to each other using simple function calls, but internally, this will be translated into Mach messages.

BSD

The BSD part of the kernel implements UNIX processes on top of Mach tasks, and UNIX signals on top of Mach exceptions and Mach IPC. UNIX filesystem semantics are implemented here just like TCP/IP networking. And while the VFS (virtual filesystem) component allows plugging in BSD-style filesystems, the /dev infrastructure plugs right into I/O-Kit. BSD exports all the semantics that an applica-

tion expects from a UNIX/BSD/POSIX compatible operating system, like "open()" and "fork()", through the syscall interface.

Since there are basically two kernels in XNU - Mach with its message passing API and BSD with the POSIX API - there are two kinds of syscalls. While both use a single int 0x80/sysenter/sc entry point, negative syscall numbers will be routed to Mach, while positive ones go to BSD. Note that, just like on Windows NT, applications may not use int 0x80/sysenter/sc directly, as this is a private interface. Instead, applications must call through libSystem, which is the equivalent of libc on OS X.

I/O-Kit

When NEXTSTEP was ported to different architectures and was renamed to OpenStep, it got a new driver model, called "DriverKit", which was based on the Objective C programming language and therefore was object oriented, and allowed an inheriting hierarchy of device drivers: For example, there could be a generic IDE/ATA device driver that handled reads and writes of blocks on an IDE bus, a hard disk driver and a CD-ROM driver that subclassed the generic IDE driver, and another CD-ROM driver that subclassed the generic CD-ROM driver to work around some quirks for one specific CD-ROM drive model. This architecture helps a lot to combat duplicate code: In contrast to other operating systems like Linux, a new device driver is not written by copying the closest match and modifying it, but by subclassing an existing driver binary and overwriting some methods with new code. "I/O-Kit" is a higher performance reimplementation of DriverKit in a subset of C++ (no exceptions, multiple inheritance, templates, runtime type information). I/O-Kit supports some classes of drivers in user mode.

KEXTs

I/O-Kit drivers are dynamically linked at runtime, as so-called "KEXTs" ("Kernel Extensions"). KEXT can not only link against the I/O-Kit component, but also against other parts of the kernel. This way, filesystem and networking KEXTs (NKEs) are possible. Every KEXT, which typically resides in /System/Library/Extensions, is a bundle, i.e. a subdirectory which contains the actual binary and an XML description of dependencies and the parts of the kernel it links against.

Other interesting details

The following sections describe some other interesting details of or around the Mac OS X kernel.

Booting

While PowerPC-based Macs use OpenFirmware, Intel-based machines use EFI ("Extensible Firmware Interface"). Both kinds of firmware are a lot more powerful than the 16 bit BIOS still shipping on PCs. While EFI can boot off USB and supports GPT partitioning and FAT32 file systems, the rest of the feature sets of OpenFirmware and EFI are pretty similar: Both can boot off FireWire, and both support APM ("Apple Partition Map") partitioning and the HFS file system, as well as firmware-level drivers. BootX is the bootloader for OpenFirmware, and boot.efi the bootloader for EFI. Both can decode HFS and can therefore read the kernel from the root partition. If there is a "KEXT cache", i.e. a file with all prelinked KEXTs suited for this configuration, that is newer than the newest file in /System/Library/Extensions and newer than the running kernel, the boot loader will load this cache; otherwise, it will go through all KEXTs and load the appropriate ones by comparing them to the entries of the "device tree" which has been passed from the firmware to the bootloader. Later, a KEXT cache will be written to disk to speed up the next boot. This is somewhat similar but more flexible than the Linux "initrd" approach.

Mach-O

Mac OS X does not use the ELF file format for binaries (executables, libraries, KEXTs) like practically all other UNIX systems. Instead, it uses Mach-O, which has roughly the same feature set, but one interesting addition: A single, so-called "fat" or "universal" binary can contain code for more than one architecture. So on OS X 10.5 Leopard, for example /usr/lib/libSystem.dylib contains code for PowerPC, PowerPC 64, i386 (32 bit Intel) and x86_64 (64 bit Intel). This way, a single Mac OS X 10.5 Leopard installation DVD can boot on four different architectures, and there is no need for "lib/lib64" (64 bit Linux) or

"SYSTEM/SYSTEM32/SYSTEM64" (64 bit Windows) style duplicate directories for different architecture/bitness versions of the same code. The function grade_binary() in the kernel's Mach-O loader decides which part of the binary to run. If the system is an i386 and the Mach-O file contains only PowerPC code, execution will be handed to Rosetta.

Rosetta

Rosetta is a compatibility solution based on Transitive's QuickTransit technology that allows running (32 bit) PowerPC code on i386 CPUs. This is done by dynamically recompiling the PowerPC code into native i386 code and managing the interfaces between emulated and native code - in practice, this means byte-swapping all data passed between i386 and PPC code, because i386 is Little Endian and PPC is Big Endian. From a performance standpoint, the optimal design would have been to only emulate the application and to use the native versions of all libraries it links against, but this would have been very impractical, since the interface between native and emulated code would have been very broad. A much easier way to achieve high compatibility is to run the complete application including all of its libraries in emulation, and only byte swap when the application makes syscalls to the native kernel. A side effect of this approach is that you potentially need all PPC versions of the system libraries installed on an Intel system, as soon as you only use a single PowerPC application in emulation.

A user can easily make experiments with this amazing technology by invoking /usr/libexec/oah/translate manually to force emulation of PowerPC code, even if an executable is available in native code.

Intel specifics

While i386 support in XNU has existed since the mid-90s, and has been a shipping feature of OpenStep, the i386 part had not been used in Mac OS X until the advent of Intel machines in 2005/2006. And with the introduction of the 64 bit Mac Pro in 2006, x86_64 (AMD64, Intel64, EM64T, x64, ...) support has been added to XNU - but XNU is not a 64 bit kernel, though. XNU supports 64 bit user mode applications, but it is 32 bit itself. Since porting a 32 bit kernel to 64 bit is a big task, it could not be done in just half a year between the introduction of the first Intel machines in January of 2006 (until then, Apple developers had worked on finalizing the 32 bit i386 version) and the introduction of the Mac Pro in August.

There is just a single kernel image for 32 and 64 bit Intel: It is loaded as a 32 bit process in 32 bit protected mode on both kinds of machines, and if 64 bit support is detected, the kernel switches into long mode compatibility mode - a mode that supports running 32 bit code, but also allows easy switching to 64 bit code. So the whole kernel code is still unmodified 32 bit code, but tiny stubs that deal with copying between user address spaces (which can be 64 bit), and the syscall and trap handlers are 64 bit code. Next to being an easy port, this has the extra advantages that the 64 bit capable kernel can still easily support 32 bit KEXTs, and conserves memory by being able to use 32 bit pointers throughout a large part of kernel code. On the flip side, the kernel cannot use the extended x86_64 register set and is restricted to a 32 bit address space.

But while all other common 32 bit operating systems like Linux, Windows and the BSDs split the address space into 2 GB for user and 2 GB for kernel (2/2) or 3 GB for user and 1 GB for kernel (3/1), the i386/x86_64 version of XNU uses a 4/4 split: While the kernel is running, the user's data is not mapped into its address space, and while user code is running, the kernel is not mapped. So user and kernel can each have 4 GB of address space with the disadvantage of being less efficient in copying of data between user and kernel. But this way, kernel mode can map more devices into its address space (like video cards with a lot of memory), and manage more RAM, thus pushing out the limit when a true 64 bit kernel is required.

iPhone

Mac OS X runs on 32 and 64 bit PowerPC and i386/x86_64 ("Intel") Macintosh machines, on the Apple TV set-top-box, which is also i386 based, and on the iPhone and the iPod touch - these devices have ARM CPUs. Specifically for these devices, XNU and parts of the Mac OS X userland have been ported to ARM. The ARM kernel does not support loading arbitrary KEXTs and is digitally signed, but

otherwise mostly equivalent to the PowerPC and i386/x86_64 versions.

What makes XNU great

While XNU might not be as scalable or as tidy as other operating systems (but catching up), it is a very modern UNIX with novel ideas and unique features:

- The kernel extension ABI is stable over several major releases of the OS.
- Fat/universal binaries allow for a single install CD or hard disk installation that runs on different CPU architectures, without the clutter of duplicating files or directories. Furthermore, 3rd party application vendors can ship a single binary that runs on multiple architectures.
- I/O-Kit allows code reuse for drivers without code duplication.
- The KEXT cache is a clean way to speed up boot times.
- The clear separation between Mach, BSD and I/O-Kit helps keeping the cost of code maintenance low.
- The powerful Mach Message API is useful for user mode applications.
- Since Mac OS X 10.5 Leopard, the i386 port of OS X is the only operating system with full POSIX-conformance that doesn't contain AT&T UNIX code.

Open Source & Hacking

With every minor operating system release (i.e. 10.5.0, 10.5.1...), Apple usually releases the whole set of source code for all components of the system that are under an open source license. which is basically everything but the GUI. About half of these packages are patched versions of common open source projects (like “bash” and “perl”), the rest is Apple code, and is released under the “Apple Public Source License” APSL, which is a BSD-style license. This makes it compatible with the standard BSD license, as well as with the OpenSolaris CDDL. But there is no live source code repository for developers visible outside Apple, so there is no real open source community that does any development on the APSL components. But there are other uses for Open Source: It helps KEXT developers debugging, it allows governmental or educational institutions to build their own versions, with added security for example, and it allows commercial companies or universities to add functionality to the kernel, either to sell it, or for research (SEDarwin, L4/Darwin).

But the source code is not necessarily complete. The XNU source code lacks most of the ARM bits, and Apple also states that other parts have been left out because of trade secrets with Intel. But a kernel compiled from the open source can still be used as a drop-in replacement for the shipping binary.

Revisiting the Buzzwords

- The OS X kernel is not Mach. The OS X kernel is called “XNU”, which consists of Mach, BSD and I/O-Kit.
- The OS X kernel is not a microkernel. Although Mach has been used as a microkernel in other projects, XNU is a very traditional monolithic kernel with BSD and (most) drivers in kernel mode.
- The OS X kernel is not based on FreeBSD. The BSD part is based on 4.4BSD with some code from FreeBSD, NetBSD and others. The OS X userland UNIX tools are mostly based on FreeBSD code, though.
- The OS X kernel is not written in C++. The I/O-Kit part is written in a subset of C++, but Mach and BSD are written in C.
- The OS X kernel is not 64 bit. It supports 64 bit user mode applications on a 64 bit PowerPC or Intel CPU, but the kernel itself runs in 32 bit mode and is bound to the 4 GB address space limit.
- The OS X kernel is Open Source, but there is no live source code repository visible outside of Apple, and the released source does not necessarily contain all code, but can be compiled into a working system.
- The OS X kernel is UNIX, but only since OS X 10.5 Leopard, and only for 32 bit i386, since this is the configuration that passed the POSIX conformance test and may therefore use the OpenGroup's “UNIX” trademark.

References

- Singh, Amit: Mac OS X Internals. A Systems Approach; Addison-Wesley, 2006.
- http://kernel.macosforge.org/
- http://www.opensource.apple.com/darwinsource/