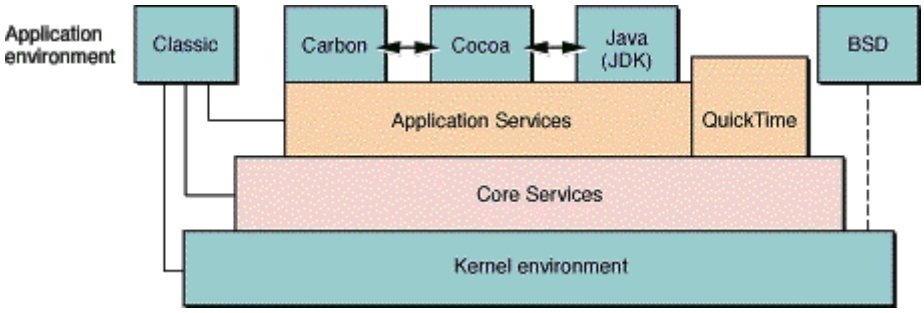
Kernel Architecture Overview

OS X provides many benefits to the Macintosh user and developer communities. These benefits include improved reliability and performance, enhanced networking features, an object-based system programming interface, and increased support for industry standards.

In creating OS X, Apple has completely re-engineered the Mac OS core operating system. Forming the foundation of OS X is the kernel. Figure 3-1 illustrates the OS X architecture.

Figure 3-1 OS X architecture



The kernel provides many enhancements for OS X. These include *preemption*, *memory protection*, enhanced performance, improved networking facilities, support for both Macintosh (Extended and Standard) and non-Macintosh (UFS, ISO 9660, and so on) file systems, object-oriented APIs, and more. Two of these features, preemption and memory protection, lead to a more robust environment.

In Mac OS 9, applications cooperate to share processor time. Similarly, all applications share the memory of the computer among them. Mac OS 9 is a *cooperative multitasking* environment. The responsiveness of all processes is compromised if even a single application doesn't cooperate. On the other hand, real-time applications such as multimedia need to be assured of predictable, time-critical, behavior.

In contrast, OS X is a *preemptive multitasking* environment. In OS X, the kernel provides enforcement of cooperation, scheduling processes to share time (preemption). This supports real-time behavior in applications that require it.

In OS X, processes do not normally share memory. Instead, the kernel assigns each *process* its own *address space*, controlling access to these address spaces. This control ensures that no application can inadvertently access or modify another application's memory (protection). Size is not an issue; with the virtual memory system included in OS X, each application has access to its own 4 GB address space.

Viewed together, all applications are said to run in user space, but this does not imply that they share memory. User space is simply a term for the combined address spaces of all user-level applications. The kernel itself has its own address space, called kernel space. In OS X, no application can directly modify the memory of the system software (the kernel).

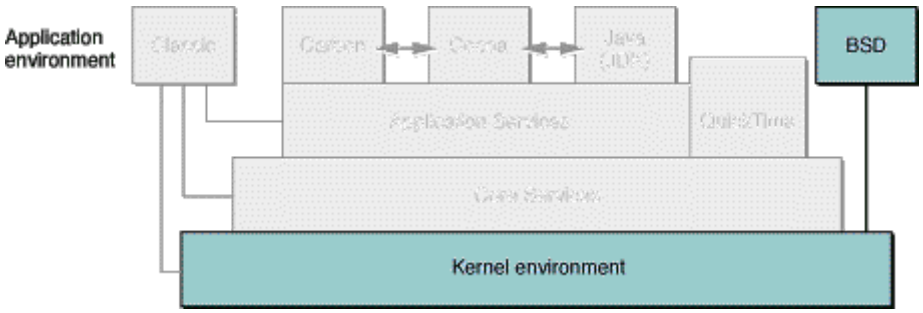
Although user processes do not share memory by default as in Mac OS 9, communication (and even memory sharing) between applications is still possible. For example, the kernel offers a rich set of primitives to permit some sharing of information among processes. These primitives include shared libraries, frameworks, and POSIX shared memory. Mach messaging provides another approach, handing memory from one process to another. Unlike Mac OS 9, however, memory sharing cannot occur without explicit action by the programmer.

Darwin

The OS X kernel is an *Open Source* project. The kernel, along with other core parts of OS X are collectively referred to as *Darwin*. Darwin is a complete operating system based on many of the same technologies that underlie OS X. However, Darwin does not include Apple's proprietary graphics or applications layers, such as Quartz, QuickTime, Cocoa, Carbon, or OpenGL.

Figure 3-2 shows the relationship between Darwin and OS X. Both build upon the same kernel, but OS X adds Core Services, Application Services and QuickTime, as well as the *Classic*, *Carbon*, *Cocoa*, and Java (JDK) application environments. Both Darwin and OS X include the BSD command-line application environment; however, in OS X, use of environment is not required, and thus it is hidden from the user unless they choose to access it.

Figure 3-2 Darwin and OS X



Darwin technology is based on *BSD*, Mach 3.0, and Apple technologies. Best of all, Darwin technology is Open Source technology, which means that developers have full access to the source code. In effect, OS X third-party developers can be part of the Darwin core system software development team. Developers can also see how Apple is doing things in the core operating system and adopt (or adapt) code to use within their own products. Refer to the *Apple Public Source License* (APSL) for details.

Because the same software forms the core of both OS X and Darwin, developers can create low-level software that runs on both OS X and Darwin with few, if any, changes. The only difference is likely to be in the way the software interacts with the application environment.

Darwin is based on proven technology from many sources. A large portion of this technology is derived from FreeBSD, a version of 4.4BSD that offers advanced networking, performance, security, and compatibility features. Other parts of the system software, such as Mach, are based on technology previously used in Apple's MkLinux project, in OS X Server, and in technology acquired from NeXT. Much of the code is platform-independent. All of the core operating-system code is available in source form.

The core technologies have been chosen for several reasons. Mach provides a clean set of abstractions for dealing with memory management, interprocess (and interprocessor) communication (IPC), and other low-level operating-system functions. In today's rapidly changing hardware environment, this provides a useful layer of insulation between the operating system and the underlying hardware.

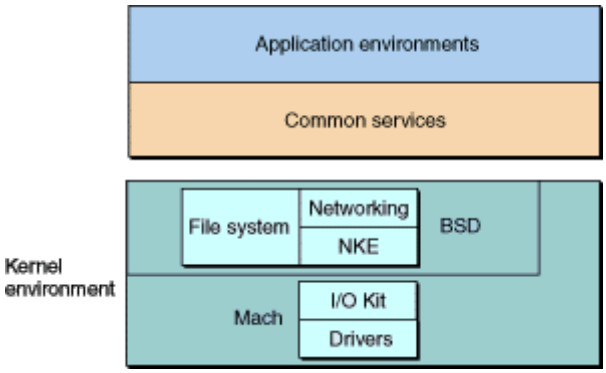
BSD is a carefully engineered, mature operating system with many capabilities. In fact, most of today's commercial UNIX and UNIX-like operating systems contain a great deal of BSD code. BSD also provides a set of industry-standard APIs.

New technologies, such as the I/O Kit and Network Kernel Extensions (NKEs), have been designed and engineered by Apple to take advantage of advanced capabilities, such as those provided by an object-oriented programming model. OS X combines these new technologies with time-tested industry standards to create an operating system that is stable, reliable, flexible, and extensible.

Architecture

The foundation layer of Darwin and OS X is composed of several architectural components, as shown in Figure 3-3. Taken together, these components form the *kernel environment*.

Figure 3-3 OS X kernel architecture



Important: Note that OS X uses the term *kernel* somewhat differently than you might expect.

"A kernel, in traditional operating-system terminology, is a small nucleus of software that provides only the minimal facilities necessary for implementing additional operating-system services." — from *The Design and Implementation of the 4.4 BSD Operating System*, McKusick, Bostic, Karels, and Quarterman, 1996.

Similarly, in traditional Mach-based operating systems, the kernel refers to the Mach microkernel and ignores additional low-level code without which Mach does very little.

In OS X, however, the kernel environment contains much more than the Mach kernel itself. The OS X kernel environment includes the Mach kernel, BSD, the I/O Kit, file systems, and networking components. These are often referred to collectively as the kernel. Each of these components is described briefly in the following sections. For further details, refer to the specific component chapters or to the reference material listed in the bibliography.

Because OS X contains three basic components (Mach, BSD, and the I/O Kit), there are also frequently as many as three APIs for certain key operations. In general, the API chosen should match the part of the kernel where it is being used, which in turn is dictated by what your code is attempting to do. The remainder of this chapter describes Mach, BSD, and the I/O Kit and outlines the functionality that is provided by those components.

Mach

Mach manages processor resources such as CPU usage and memory, handles scheduling, provides memory protection, and provides a messaging-centered infrastructure to the rest of the operating-system layers. The Mach component provides

- untyped interprocess communication (*IPC*)
- remote procedure calls (*RPC*)
- scheduler support for symmetric multiprocessing (*SMP*)
- support for *real-time* services
- virtual memory support
- support for *pagers*
- modular architecture

General information about Mach may be found in the chapter Mach Overview. Information about scheduling can be found in the chapter Mach Scheduling and Thread Interfaces. Information about the VM system can be found in Memory and Virtual Memory.

BSD

Above the Mach layer, the BSD layer provides “OS personality” APIs and services. The BSD layer is based on the BSD kernel, primarily *FreeBSD*. The BSD component provides

- file systems
- networking (except for the hardware device level)
- UNIX security model
- `syscall` support
- the BSD process model, including process IDs and signals
- FreeBSD kernel APIs
- many of the *POSIX* APIs
- kernel support for *pthreads* (POSIX threads)

The BSD component is described in more detail in the chapter BSD Overview.

Networking

OS X networking takes advantage of BSD’s advanced networking capabilities to provide support for modern features, such as Network Address Translation (*NAT*) and *firewalls*. The networking component provides

- 4.4BSD TCP/IP stack and socket APIs
- support for both IP and DDP (AppleTalk transport)
- *multihoming*
- routing
- *multicast* support
- server tuning
- packet filtering
- Mac OS Classic support (through filters)

More information about networking may be found in the chapter Network Architecture.

File Systems

OS X provides support for numerous types of file systems, including *HFS*, *HFS+*, *UFS*, *NFS*, *ISO 9660*, and others. The default file-system type is HFS+; OS X boots (and “roots”) from HFS+, UFS, ISO, NFS, and UDF. Advanced features of OS X file systems include an enhanced Virtual File System (*VFS*) design. VFS provides for a layered architecture (file systems are *stackable*). The file system component provides

- *UTF-8* (Unicode) support
- increased performance over previous versions of Mac OS.

More information may be found in the chapter File Systems Overview.

I/O Kit

The I/O Kit provides a framework for simplified driver development, supporting many categories of devices.The I/O Kit features an object-oriented I/O architecture implemented in a restricted subset of C++. The I/O Kit framework is both modular and extensible. The I/O Kit component provides

- true plug and play
- dynamic device management
- dynamic (“on-demand”) loading of drivers
- power management for desktop systems as well as portables
- multiprocessor capabilities

The I/O Kit is described in greater detail in the chapter I/O Kit Overview.

Kernel Extensions

OS X provides a kernel extension mechanism as a means of allowing dynamic loading of pieces of code into kernel space, without the need to recompile. These pieces of code are known generically as *plug-ins* or, in the OS X kernel environment, as *kernel extensions* or *KEXTs*.

Because KEXTs provide both modularity and dynamic loadability, they are a natural choice for any relatively self-contained service that requires access to interfaces that are not exported to user space. Many of the components of the kernel environment support this extension mechanism, though they do so in different ways.

For example, some of the new networking features involve the use of network kernel extensions (*NKEs*). These are discussed in the chapter Network Architecture.

The ability to dynamically add a new file-system implementation is based on VFS KEXTs. Device drivers and device families in the I/O Kit are implemented using KEXTs. KEXTs make development much easier for developers writing drivers or those writing code to support a new volume format or networking protocol. KEXTs are discussed in more detail in the chapter Kernel Extension Overview.