GRIFFITH COLLEGE DUBLIN

Mac Operating System :
The Kernel of macOS

Operating Systems Design
BSC2-OSD OPERATING SYSTEMS DESIGN

By ***Tom-Eliott Herfray***
Student Number : 2999664
tomeliott.herfray@student.griffith.ie

Abstract

According to *StatCounter*, in January 2019, *macOS* accounted for almost 10.9% of the market share of desktop operating systems. This represents several hundred million devices around the world. This market share is gradually increasing thanks to the popularity of other Apple platforms, such as *iOS*, *tvOS* or *watchOS*.

As an engineering student, I have used various operating systems for many years, such as *ArchLinux*, *Fedora*, *Manjaro* or *macOS*. It is interesting to see how these different operating systems, which many sometimes mistakenly call *"Linux"* (or *UNIX*), work. In this report, we will see how *XNU*, the kernel of *Darwin*, *macOS* and *iOS* systems, works, how it has evolved and how it differs from other OS's kernels.

Introduction

Developed in the 1990s after the return of Steve Jobs at Apple, *XNU* was officially presented in December 1996 as the kernel of *Darwin*, an open-source operating system developed by Apple and derived from *NeXTSTEP*, *BSD* and *Mach*. Unlike *macOS*, *Darwin* does not have the *Aqua* GUI and the *Quartz* Graphics Engine, and many people often confuse *Darwin* and *XNU*. Apple presents *XNU* on its website as *"The Darwin Kernel",* which can be confusing.

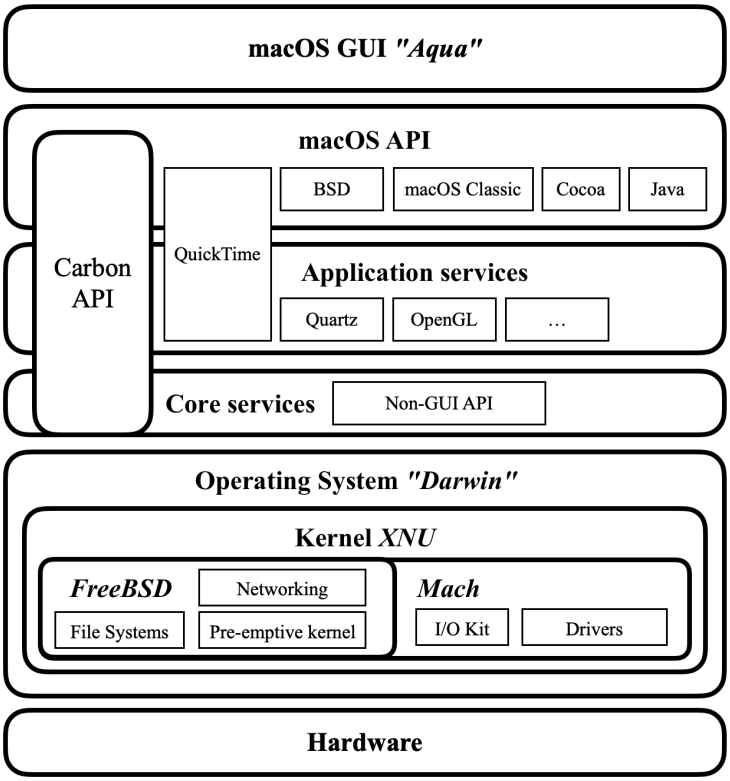
XNU is an acronym for *"X is Not Unix"*. Apple considers *XNU* as an hybrid kernel that combine the *Mach* kernel developed at the University of Carnegie Mellon, with *FreeBSD* components and a C++ API called *I/O Kit* (Input / Output Kit). *XNU* was designed to work on x86-32, x86-64, ARM and PowerPC architectures for both single processor and multi-processor configurations. Note also that Apple no longer uses PowerPC micro-processors since 2007, preferring Intel processors.

We will focus on the general operation of the kernel *XNU* : Firstly, we will focus on *Mach*, which mainly handles memory and processor resources of macOS. Secondly, we will see the advantages of *FreeBSD* (or *BSD 4.4*) which, unlike *Mach*, deals with file, network and process management. Finally, we will briefly explain the role of the *I/O Kit* framework in the kernel.

The macOS Architecture

Nota Bene *: To avoid confusion, macOS and Mac OS* ***<u>are not</u>*** *exactly the same systems. Mac OS, which is also called "Mac OS Classic", is the ancestor of macOS (from version 1 to version 9.2.2). macOS, also known as "Mac OS X" or "OS X", is very different from Mac OS and is used since version 10 ("Mac OS X 10"). We will talk here about the macOS's kernel, which has the advantage over Mac OS to be much less limited.*

macOS is a very complex system that has undergone many changes. *macOS* is designed in several layers : the visual interface (*GUI*), APIs, application services, core services and the kernel. Here is a visual architecture of *macOS* that I made.



Architecture of *macOS* since version 10 *("Mac OS X 10")*

As we see with this diagram, *macOS*'s kernel is divided into two "layers" : *Mach 3.0* (a microkernel, which serves as a base for supporting any of several operating systems) and *FreeBSD* (a popular open-source version of UNIX which represents the XNU's second layer). Each of these two kernels have their use and their specificity within the system. To them, they make up the basics of *macOS*.

The *XNU* Kernel

The ancestor of *macOS*, *Mac OS Classic*, was a cooperative multi-tasking environment, which means that the responsiveness of all processes is compromised if even one application does not cooperate. *Mac OS Classic*, with the cooperative multi-tasking and the lack of memory protection, was very limited compared to *macOS*. With *macOS*, Apple introduced a preemptive kernel, called *XNU*, where a process running in *kernel-mode* can be replaced by another process in the middle of a kernel process. The kernel ensures the execution of planning processes to share-time and supports real-time behavior in tasks that require it.

The advantage of the *XNU* kernel (*BSD* is derived from *UNIX*) is to have a stable and robust system, and also to be able to easily install *Linux* program. Its role is to assign to each process its unique address space. With this security, no program can access to the memory of another program. *XNU* has also its own address space, called "kernel-space". In *macOS*, no application can change the kernel-space. For *macOS*, the address space and kernel-space are managed by the *Mach* kernel. We will now see the precise role of *Mach* and *FreeBSD* in the use of *XNU*.

Darwin's First Layer : *FreeBSD*

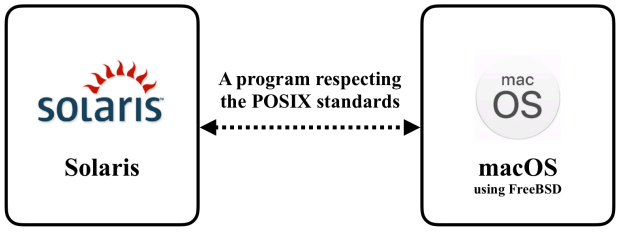
As we saw previously, the *XNU* kernel is partly derived from *BSD* (more especially *FreeBSD*). *FreeBSD* provides many advanced features to *XNU*, like the boot (*BootROM*, which is part of the Mac's hardware, initializes system hardware and selects an operating system to run), shutdown, or accounting procedures. *FreeBSD* deals with files, networking and processes, where *Mach* is more designed to deal with processor and memory resources. With the expansion of the Internet, *FreeBSD* also brought to Apple the opportunity to properly manage networking with support for industry standards, and to provide to the user vital functions like email or Internet services (Firewall security services, HTTP, routing, FTP, etc.). Concerning the local network, the user can share files (using *NFS*, cited in the file system section below) with computers running *MacOS Classic* or a *Linux* operating system.

FreeBSD is also widely used on *macOS* and *Darwin* to manage file system, which controls how the data is stored and retrieved on the system. By supporting a file system, *macOS* can read and write files using one of six major *macOS* formats : *HFS+*, *HFS*, *UFS*, *UDF*, *ISO-9660* and since 2017, *APFS* (for Apple File System). *FreeBSD* also provides compatibility between *macOS* and *Windows*, especially between *MS-DOS* and *APFS*. With file system, the user can reformat the disks on *macOS* into *MS-DOS* or a *Windows* format.

One of the most important aspects of *FreeBSD* is also the support of *POSIX* standards (for "*Portable Operating System Interface"*). *POSIX* is a family of standards using by *FreeBSD* and other operating systems to make uniform APIs

between *UNIX* operating systems. *POSIX* was designed so that programs designed for a *UNIX* system can run on a other *UNIX* system. By compiling a program using *POSIX* standards, a *UNIX* program can run easily on *macOS*.

I made a small example below to illustrate a *POSIX*-compliant program. A program built on *Solaris*, a *POSIX*-compliant system, can be easily compiled on *macOS* (which uses *POSIX* standards through *FreeBSD*), sometimes with some minor modifications to make.



Nota Bene *: To compile a Windows program on macOS, FreeBSD or a UNIX operating system, the procedure is different and "Wine" can be an interesting solution.*

Finally, if *FreeBSD* is great for managing network services, files systems and user management policies like security, Apple has chosen to also use *Mach* in *macOS* for managing process management service and memory management subsystem.

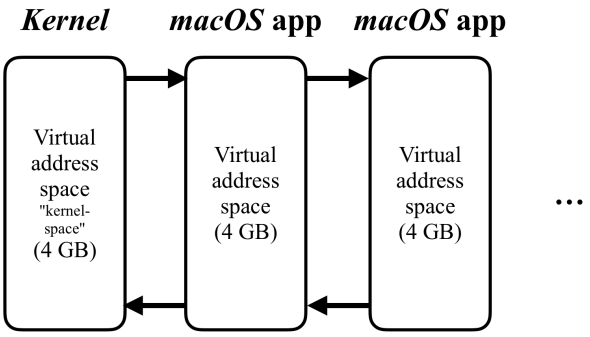
Darwin's Second Layer : *Mach 3.0*

XNU is also derived from *Mach*, an open-source operating system micro-kernel. One of Mach's design goals is portability, it manages memory and processor resources such as CPU usage, provides a protection for memory and handles scheduling. More precisely, it also provides support for real-time services, virtual memory and pagers. To understand how *Mach* works, here is a glossary provided by Apple explaining the concepts I am going to introduce :

- *Tasks* : The units of resource ownership, each task consists of a virtual address space, a port right namespace, and one or more threads (similar to a process) ;
- *Threads* : The units of CPU execution within a task ;
- *Address space* : In conjunction with memory managers, *Mach* implements the notion of a sparse virtual address space and shared memory ;
- *Memory objects* : The internal units of memory management. Memory objects include named entries and regions; they are representations of potentially persistent data that may be mapped into address spaces.

As I said earlier, *Mach* primarily manages memory and CPU resources. Memory is an important part of *macOS*, especially the virtual memory, that allows programs to be executed even though they are not stored entirely in memory, and the protected memory. Apple considers that the "virtual memory is seen as collection of VM (Virtual Memory) objects and memory objects […] with a particular owner and protections – These objects can be modified with objects calls that are available both

to the task and to the pagers". As we saw in the section *"The XNU Kernel"*, to protect and manage memory, the kernel assigns a unique address space to each process, or task. It also assigns the rights to access to the selected resources. *macOS* gives 4 GB of protected virtual address to each task to contribute to the stability of applications. *Mach* also gives a specific address space to *XNU* that we call the "kernel-space". This kernel-space is the virtual memory area in which all *XNU* threads reside. In kernel-space, a user cannot read or write to the kernel space address, else we get a *segfault* ("segmentation fault").



Each process, including XNU, gets its own 4 GB of virtual address space.

Under *macOS*, CPU resource management is handled by the *Mach* kernel using a preemptive multi-tasking, unlike *Mac OS Classic* which used a cooperative multi-tasking (also called *"non-preemptive multi-tasking"*) and where the responsiveness of all processes is compromised if a single process does not do cooperate. *Mach* allows each task in turn to use all the resources with preemptive multi-tasking. This means that preemption and memory protection lead to a more robust and stable environment.

Mach also provides in addition to memory management and CPU resources a powerful messaging system. This system lets the threads keep track of the data of the task and improves the interaction process of programs in the multi-processor environment. The messaging system also allows the system to start and stop tasks (that is what we call *"tasks scheduling"*).

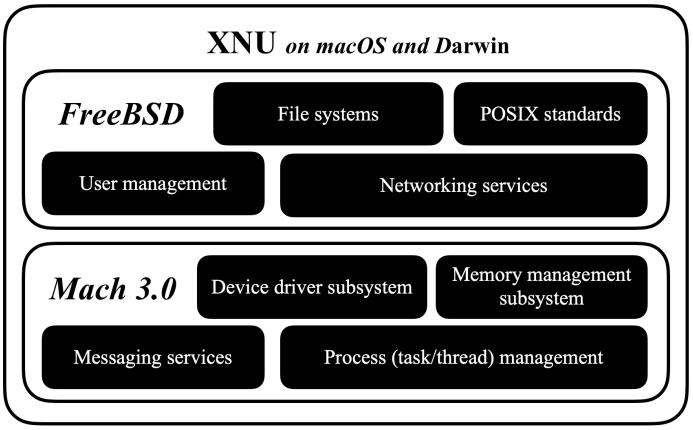
The I/O Kit

The final part of the *XNU* kernel is *I/O Kit*, a framework that provides an abstract view of the system hardware to the layers of *macOS*. *I/O Kit* enables drivers for printers and other devices (using USB, FireWire, etc.) to be written quickly and it implements most of the plug-and-play capabilities of the system. We will not go into the technical details of the framework but note however that even if *I/O Kit* is not a "kernel" itself like *FreeBSD* or *Mach*, it is an important and integral part of *XNU*.

Summary

Today, *XNU* is a very complete kernel that allows *macOS* to be a stable, efficient and *"friendly"* operating system. We saw in this report that *XNU* is based on two other kernels, *Mach 3.0* and *FreeBSD*. We have seen the main role of each of these kernels. More precisely than *Mach* managed memory and processor resources, and *FreeBSD* managed file, network and process management. *Darwin*, *XNU*'s operating system, is open-source, which means that all developers or non-developers have full access to the source code (available on *GitHub*). But *Darwin* is not just based on two kernels, it is also based on *I/O Kit*, a sophisticated framework that helps developers code device drivers for macOS (or even iOS). Some developers also consider that if we had to describe the *macOS*'s kernel in only three words, that would be "*I/O Kit – FreeBSD – Mach"*.

In conclusion, *XNU* is a powerful kernel that gave life to several operating systems, including of course *macOS*, but also *iOS*, an operating system used on several billion devices, or even *watchOS* and *tvOS*. *XNU* also relies on thousands of libraries and frameworks (like *I/O Kit*) but this report was primarily intended to present the role of the kernel itself. Here is a diagram that summarizes the role of *FreeBSD* and *Mach* in *XNU*.



References

- *Mac OS X: The Complete Reference*, by **Jesse Feiler**, Osborne/Mc-Graw-Hill (2001) – ISBN : 0-07-212663-9, available at *Griffith Library ;*
- *The Darwin Kernel Source*, by **Apple Inc.** (2019) – *www.github.com/apple/darwin-xnu ;*
- *Operating Systems A Systematic View >> Sixth Edition*, by **William S. Davis and T. M. Rajkumar**, Pearson Education Inc. (2005) – ISBN : 0-321-26751-6, available at *Griffith Library ;*
- *Kernel Programming Guide*, by **Apple Inc.** (2013) – *www.developer.apple.com/library/archive/documentation/Darwin/Conceptual ;*
- *The Documentation Archive*, by **Apple Inc.** (2016) – *www.developer.apple.com/library/archive/navigation.*