Die Architektur von Mac OS X und iOS

Seminararbeit
von Pawel Wall
Uni Siegen
01.07.2011

Inhaltsverzeichnis:

Einleitung

Die Betriebssysteme Mac OS X und iOS von Apple haben heute eine sehr weite Verbreitung. OS X ist das Betriebssystem der Apple-Computer („Macs“). Das davon abstammende iOS findet man auf den Geräten iPhone, iPod touch, iPad und auf der zweiten Generation des Apple TV. Vor allem das iPhone genießt eine hohe Popularität und zählt mittlerweile zu den meistverbreiteten Smartphones. Diese Ausarbeitung beschäftigt sich mit der Architektur dieser beiden Betriebssysteme und beleuchtet die Hintergründe. Nach einer Darstellung der Vorgeschichte werden die Kernkomponenten von Mac OS X vorgestellt. Anschließend wird auf das jüngere iOS eingegangen.
Die Architektur von Mac OS X ist besser zu verstehen, wenn man seine Herkunft und das Zustandekommen betrachtet. Mac OS X wird als ein so genanntes „unixoides System“ bezeichnet. welches versucht, die Verhaltensweise (aber auch die Schnittstellen, siehe POSIX) des Betriebssystems UNIX zu implementieren. Vor allem von Apples Marketingabteilung wird oft der Eindruck erweckt, Mac OS X habe einen „echten“ Unix-Unterbau, teile sich also Code mit dem Ur-UNIX - was nicht direkt zutrifft. Es gibt nämlich unterschiedliche UNIX-Arten und Einteilungen, die unten genauer erklärt werden.
Bevor es zum geschichtlichen Umriss geht, wird noch der Begriff des Kernels definiert und seine Unterarten benannt.

<u>UNIX-Kategorien</u>

Genetisches Unix
Systeme die eine tatsächliche historische Verbindung zur ursprünglichen AT&T Codebasis aufweisen, deren Code also auf dem ursprünglichen UNIX basiert. Die meisten (wenn auch nicht alle) proprietären UNIX-Systeme fallen unter diese Kategorie. Auch die BSD-Systeme, die Abkömmlinge der in den frühen 1980er Jahren am der University of California, Berkeley entstandenen Systeme sind, zählen dazu. Heutige, von BSD abgeleitete Systeme, wie z. B. FreeBSD oder Darwin enthalten keinen originalen Unix-Quellcode mehr, werden aber ebenso in diese Kategorie gezählt.

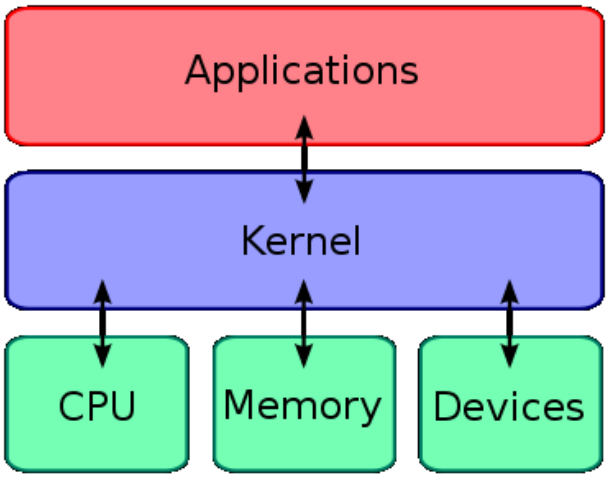
Funktionelles Unix
Zur Gruppe der „Funktionellen Unices“ gehören im Allgemeinen alle Betriebssysteme, die sich in gewissem Maße wie Unix verhalten. Als Beispiel können „Linux“ und „Minix“ herangezogen werden. Sie haben keine direkte Verbindung zur ursprünglichen Codebasis von AT&T, haben in Ihrem Verhalten aber Ähnlichkeiten zu Unix. Die allermeisten Open-Source-Implementierungen gehören zu dieser Kategorie, auch weil eine UNIX-Zertifizierung seitens der Open Group sehr kostspielig ist.

Warenzeichen „UNIX“
Diese Systeme - größtenteils kommerzieller bzw. proprietärer Natur - wurden von der Open Group zertifiziert die „Single UNIX Specification“ zu erfüllen. Nur sie dürfen das Warenzeichen UNIX® tragen. Die meisten dieser proprietäre Systeme sind mehr oder weniger Derivate des „System V“, bis auf wenige Ausnahmen - so erhielt z.B. IBMs z/OS (welches ansonsten kein echtes UNIX-System ist) das Warenzeichen UNIX durch die Implementierung einer POSIX-Kompatibilitätsschicht. Auch Mac OS X ist UNIX-zertifiziert.

Die Open Group ist ein Industriekonsortium mit über 300 Mitgliedern, das das UNIX-Warenzeichen lizensiert und neue Industriestandards für Unix entwickelt.

Kernel und Kerneltypen

Der Kernel ist die Haupt- oder Kernkomponente eines Betriebssystems. Es stellt die Brücke zwischen den Anwendungsprogrammen und der eigentlichen Datenverarbeitung auf Hardwareebene dar, auf die er direkten Zugriff hat. Der Kernel ist zuständig für die Ressourcenzuteilung (die Kommunikation zwischen Hardware und Softwarekomponenten), wodurch z.B. Parallelverarbeitung (also Multitasking) ermöglicht werden kann. Er regelt die Prozess- und Datenorganisation und bildet die unterste Softwareschicht des Systems. Auch die Ein- und Ausgabe wird gewöhnlich vom Kernel geregelt. Auch Mechanismen zur Kommunikation zwischen einzelnen Prozessen werden durch ihn zur Verfügung gestellt. Die Teile des Betriebssystems, die nicht zum Kernel gehören, werden als „Userland“ bezeichnet.



Es gibt unterschiedliche Kernelarten, die diese Aufgaben auf unterschiedliche Weise lösen. Während so genannte „monolithische Kernel“ den Code aller Betriebssystemfunktionen im selben Adressbereich ausführen, um die Performanz des Systems zu verbessern, führt ein so genannter „Mikrokernel“ die meisten solcher Prozesse als „Servers“, also Dienste im Benutzermodus aus. Im Folgenden wird genauer auf die drei Kerneltypen eingegangen.

Monolithischer Kernel
In einem System mit monolithischem Kernel laufen alle grundlegenden Betriebssystemdienste zusammen mit dem Hauptthread des Kernels, befinden sich also auch im gleichen Speicherbereich. Aus diesem Ansatz ergibt sich ein sehr mächtiger und performanter Hardwarezugriff. Unter Entwickler gilt die Implementierung eines monolithischen Kernels als einfacher, als die eines Mikrokernels. Der größte und wichtigste Nachteil monolithischer Kerne sind die Abhängigkeiten zwischen den Systemkomponenten: Ein kleiner Fehler, beispielsweise in einem Gerätetreiber kann zum Absturz des gesamten Systems führen. Des Weiteren kann die Wartung größerer Kerne sehr schwierig sein.

Mikrokernel
Beim Mikrokernel-Ansatz, stellt der Kernel selbst nur eine sehr grundlegende und minimale Funktionalität zur Verfügung. Das sind beispielsweise die Speicher- und Prozessverwaltung (inkl. Multitasking), sowie die Interprozesskommunikation. Andere Dienste, einschließlich solcher, die normalerweise von einem Kernel verfügbar gemacht werden (Netzwerk-Funktionalität, Geräteunterstützung etc.), sind als eigene Prozesse (Server) im Benutzermodus implementiert. Diese kommunizieren dann mit den jeweiligen Programmen, die diese Dienste angefordert haben. Die Wartung solcher Mikrokernel gestaltet sich leichter, als die von monolithischen; durch die vielen einzelnen Nutzerprozesse und Systemaufrufe werden mehr Kontextwechsel nötig, was einen Geschwindigkeitsnachteil gegenüber monolithischen Kernels mit sich bringt.

Hybrider Kernel
Die Idee hinter dem hybriden Kernel ist es, die Geschwindigkeit und den einfachen Aufbau des monolithischen Kernels mit der Modularität und Ausführungssicherheit des Mikrokernels zu kombinieren. Der Hybridkernel stellt also eine Kombination bzw. einen Kompromiss zwischen Beiden Extremen (Mikrokernel und monolithischer Kernel). Beispielsweise werden einige Dienste wie die Dateisystemverwaltung die Netzwerktreiber mit im Kernel ausgeführt, während bestimmte andere als Prozesse im Benutzermodus laufen. Mac OS X besitzt einen hybriden Kernel.

Von UNIX bis Rhapsody - Die Vorgeschichte von Mac OS X

UNIX
Immer wieder taucht im Zusammenhang mit Mac OS X der Begriff „Unix“ auf. Unix, ein Multitasking- und Mehrbenutzer-System, wurde im Jahr 1969 von einer Gruppe AT&T-Mitarbeiter in den so genannten Bell Labs entwickelt und hat eine große Bedeutung für die Computerwelt. Unix-Betriebssysteme sind heute in Servern, Workstations und mobilen Geräten weit verbreitet. Das Client-Server-Programmmodell war von größter Wichtigkeit für die Entwicklung für die Entwicklung des Internets. Weitere, wichtige Konzepte fanden in UNIX zusammen bzw. wurden erstmals darin umgesetzt: Ein (heute selbstverständliches) hierarchisches Dateisystem ebenso dazu wie das „Everything is a file“-Prinzip: Geräte, wie Disketten- und CD-Laufwerke, Bandgeräte, ja zum Teil sogar Prozesse und deren Eigenschaften werden in Dateien abgebildet, was eine einfache, einheitliche Schnittstelle für die unterschiedlichsten Applikationen ermöglicht. Mithilfe so genannter „Pipes“ können via Kommandozeileninterpreter mehrere kleine Programme hintereinandergeschaltet und kombiniert werden, anstatt ein großes Programm verwenden zu müssen, das alle diese Funktionen beinhaltet. Dieses „ein Programm für jede Aufgabe“-Konzept ist heutzutage vor allem unter Linux-Anwendern sehr beliebt und verbreitet. Unter Mac OS X spielt es für die meisten Endanwender keine größere Rolle, da hier das Ziel verfolgt wird, alle Funktionen über eine grafische Benutzeroberfläche zugänglich zu machen.
Ende der 1960er Jahre beschäftige sich ein Konsortium aus mehreren Firmen mit der Entwicklung des neuen Betriebssystems „Multics“, auch AT&Ts Bell Laboratories waren daran beteiligt. Doch das Projekt mit den Zielen „Mehrbenutzerbetrieb, gemeinsamer Zugriff auf Daten und die Aufteilung von Rechenleistung“ scheiterte und die Bell Labs zogen sich zurück.

Gründe waren weniger die Ansätze und Konzepte als die Größe und Komplexität von Multics, das die Leistungsfähigkeit damaliger Hardware überstieg. Ein kleines Team der Bell Labs begann daraufhin mit den Arbeiten an dem Multiuser-Betriebssystem UNIX, das seinen Namen erst später erhalten sollte. Ein großes Ziel bei der Entwicklung von UNIX war die Möglichkeit des gemeinsamen Programmierens. Nachdem man es geschafft hatte, UNIX erfolgreich als Textverarbeitungssystem im Patentbüro der Bell Labs einzusetzen, konnten die Entwickler auf mehr finanzielle Unterstützung hoffen, UNIX wurde auf leistungsfähigere Rechner portiert und weiter ausgebaut. Kurz darauf begann man -ebenfalls in den Bell Laboratories- parallel mit der Entwicklung der imperativen Programmiersprache „C“, die bis heute eine weite Verbreitung erfahren hat. Es folgte die Portierung von UNIX in diese neue Programmiersprache. Das fertig übersetzte System erhielt den Namen Unix V4. Da Unix nun in einer „höheren“ Programmiersprache vorlag (und nicht mehr in Assemblersprache), konnte es mit vergleichsweise geringem Aufwand auf andere Systeme und Architekturen portiert werden. Derweil stieg das Interesse an UNIX stetig, auch außerhalb der Bell Labs.
Wegen seiner Monopolstellung auf dem Kommunikationsmarkt war es AT&T, der Mutterfirma der Bell Labs, verboten, in neue Märkte wie dem Softwarevertrieb vorzustoßen, was eine kommerzielle Vermarktung von Unix unmöglich machte - nur eine unentgeltliche Lizensierung kam infrage. So begann man, UNIX mitsamt Quell- und Maschinencode an Institutionen aller Art - auch Universitäten herauszugeben. Es begann vor allem im akademischen Bereich ein reger Austausch von Quellcode und selbstgeschriebenen Dokumentationen und Unix machte mehrere Versionssprünge bzw. -verzweigungen. Das Betriebssystem wurde um das oben beschriebene Konzept der Pipes erweitert, was Entwicklungszyklen verkürzte und zur Popularität beitrug. Mitte der 1970er Jahre erschien das erste kommerzielle UNIX, allerdings nicht unter AT&T.

BSD

An der kalifornischen Berkeley University hatte UNIX eine besonders hohe Beliebtheit. In ihrem noch relativ jungen Informatik-Fachbereich entstanden viele Funktionen und Verbesserungen, die Hochschule wurde sehr wichtig für die UNIX-Entwicklung. So wurde dort die Unterstützung von TCP/IP dort erstmals implementiert, entsprechender Code gelangte sogar in die offizielle AT&T-Version von UNIX. Eine eigene Distribution wurde aus der Taufe gehoben, die Berkeley Software Distribution, kurz „BSD“. Zwei Jahre später veröffentlichte AT&T die UNIX-Version V7. Es sollte die letzte Version mit freiem Quellcode sein. Alle weiteren Versionen waren nur noch wenigen Hochschulen zugänglich. Um Unklarheiten um unterschiedliche Versionen zu beseitigten, fasste AT&T den aktuellen Entwicklungsstand unter dem „UNIX System V Release 1“ zusammen und nachdem das Monopolverfahren fallen gelassen wurde begann die Kommerzialisierung von UNIX. Dieser drastische Schritt brachte die lebendige UNIX-Entwicklerszene fast zum Stillstand. Doch an der Berkeley Universität wurde weiter am eigenen, noch auf UNIX V7 basierenden BSD weitergearbeitet - mit Erfolg: viele Neuerungen erschienen zuerst in BSD und wurden dann teilweise in AT&Ts System V übernommen. Die darauffolgende Zeit, auch als „Unix Wars“ bekannt, war geprägt von Lagerbildungen und Standardisierungsversuchen, bei dem sich der Standard POSIX und die X/Open-Systemspezifikation besonders hervortaten. Das „System V Release 4“ vereinte letztendlich Funktionen von BSD, das aus ihm entstandene SunOS, Xenix (ein Unix von Microsoft, später „SCO UNIX“) und dem System V selbst sowie die X/Open-Spezifikation; die Lizenzgebühren stiegen. Wegen eines Urheberrechtsstreits mussten Teile von BSD neu geschrieben werden,

so dass sie keinen originalen UNIX-Code mehr enthielten. Die Version 4.4BSDlite war sogar vollständig von Unix-Code bereinigt und bildet die Grundlage für alle modernen BSD-Derivate.

NeXTStep

Nachdem Steve Jobs 1986 Apple verlassen hatte, gründete er 1986 die Firma NeXT. Sie entwickelte das unixoide Betriebssystem NeXTStep, es ist ein direkter Vorfahre von Mac OS X und damit auch iOS. Auf dem so genannten Mach-Mikrokernel (siehe unten) baute ein normales BSD-Unix auf. Das Betriebssystem kam mit präemptivem Multitasking, Multithreading und Speicherschutz. Die Bildschirmdarstellung erfolgte mittels „Display Postscript“ (eingeschränkte Erweiterung von PostScript, einer Beschreibungssprache von z.B. Druckdaten). Diesem ist das in Mac OS X eingesetzte „Quartz“ sehr ähnlich. Die standardmäßig eingesetzte Programmiersprache „Objective C“ und die mitgelieferten exzellenten Entwicklertools machten eine objektorientierte Entwicklung - auch von sehr komplexer Software - für und unter NeXTStep sehr einfach. Der erste Webserver der Welt stand 1990 dem Schweizer CERN-Forschungszentrum auf einem NeXTStep System zur Verfügung.

Der Mach-Kernel

UNIX war mit der Zeit ein komplexes System geworden und so begann man 1984 an der Carnegie Mellon University mit der Entwicklung des Mach-Mikrokernels - er sollte in BSD zum Einsatz kommen und UNIX kompatibel sein. Diese UNIX-Kompatibilität konnte der so genannte „Accent Kernel“, auf dem die Entwicklung von Mach aufbaute, nicht bieten. Aus diesem Übernommen wurde aber Konzept der „Inter-Process Communication“ (IPC), ein sehr universelles Pipe-Artiges System, mit dem jegliche Information zwischen zwei Programmen ausgetauscht werden konnte. Dies wurde mittels „Shared Memory“ realisiert, bei dem zwei oder mehrere Prozesse einen bestimmten Teil des Hintergrundspeichers gemeinsam nutzen können. Weitere wichtige Konzepte bzw. Abstraktionen von Mach:

Task

Ein Task ist ein „Container“ für die Ressourcen eines oder mehrerer Threads. Beispiele für Ressourcen sind: Virtueller Speicher, Ports, Prozessoren usw.

Threads

Ein Thread ist die Basiseinheit einer Ausführung innerhalb eines Tasks. Der Task nur bildet die Ausführungsumgebung für seine Threads, und *diese* werden *eigentlich* ausgeführt. Die Threads eines Tasks teilen sich Ressourcen, jedoch hat jeder Thread seinen eigenen Ausführungszustand und einen eigenen Programmzähler.

Port

Ports bilden die Basis für Machs IPC-Fähigkeiten. Ein Port nimmt die „Messages“, also die IPC-Nachrichten entgegen.

Message
Eine Message ist eine Ansammlung von Daten, die sich Threads verschiedener oder desselben Tasks mittels Ports zusenden können.

Der Mach-Mikrokernel sollte kein Dateisystem, keine Netzwerkfunktionalität oder sonstige Ein-/Ausgabefunktionen zur Verfügung stellen. Das sollten dann eventuelle, darüber aufgebaute Betriebssysteme über Benutzerdienste bieten. Spätere Versionen sollten dieses Konzept jedoch teilweise aufbrechen, so hatte Version 2.5 einige monolithische Implementierungen, bei dem sich Mach und BSD im selben Adressraum befanden. Bei Version 3 besann man sich jedoch auf den Mikrokernel-Ansatz: BSD wurde als Task im Benutzermodus ausgeführt, während der Mach-Kernel selbst nur sehr fundamentale Funktionen zur Verfügung stellte.

OpenStep/ OPENSTEP
1993 entwickelte NeXT zusammen mit Sun die objektorientierte API (Anwendungsprogrammschnittstelle) OpenStep, die auch auf nicht-NeXTStep-Betriebssystemen aufsetzen konnte. Es gab auch Versionen für Suns Solaris und Microsofts Windows NT. Die auf NeXTStep basierende OpenStep-Implementierung namens „OPENSTEP“ (alles groß geschrieben) erschien ein Jahr später. Die wesentlichen Merkmale der OpenStep API:

- Die API beschreibt nur die Bibliotheken und Dienste der höheren Schichten des Systems; nicht aber der Rest des Betriebssystems, wie ihn z.B. NeXTStep beinhaltet
- Sämtlicher Code, der Abhängigkeiten zum Mach Kernel vorwies, wurde entfernt, damit OpenStep auf jedem einigermaßen leistungsfähigen System ausführbar war
- Ein erheblicher Aufwand wurde betrieben, um das System „endianness-free“ zu machen. Endianness ist ein Problemstellung beim Speichern von ganzzahligen Werten im Arbeitsspeicher
- Primitive Objekte, wie z.B. Strings wurden in NeXTStep noch mit C-Datentypen repräsentiert, in OpenStep kamen neue, eigene Klassen (NSString, NSNumber usw.) zum Einsatz. Dies sollte die Plattformunabhängigkeit erhöhen.
- OpenStep benutzt „Reference Counting”, um zu allokierenden Speicher und die Lebensdauer von Objekten zu behandeln.
- Es kam das auf PostScript basierende Display PostScript zum Einsatz, eine vielseitige und leistungsfähige Methode, um Fenster und Grafiken auf den Bildschirm zu zeichnen

Mac OS
Das Betriebssystem, welches vor Mac OS X mit Apples Computern ausgeliefert wurde, war Mac OS. Aus ihm stammen die Mac OS X-Komponenten Carbon und QuickTime.

Rhapsody

Nachdem Apple 1996 NeXT aufgekauft hatte, begann man mit der Entwicklung eines Nachfolgers für Mac OS. Das neue Betriebssystem bekam den Namen „Rhapsody“. Das System baute auf OPENSTEP auf. Durch die virtuelle Umgebung „Blue Box“, auch später als deren Weiterentwicklung Classic Environment bekannt, war es möglich, Anwendungen des alten Mac OS auszuführen. Dazu wurde ein Mac OS emuliert, in diesem Fenster konnte die alte Software ausgeführt werden. Interaktionen zwischen diesen Programmen und nativen Rhapsody- bzw. OpenStep-Anwendungen waren nicht möglich. Ebenfalls mit dabei war eine Java Virtual Machine. Auch Apple-eigene Techniken wie QuickTime oder Carbon erhielten Einzug. Das Fenstersystem basierte zwar auch in Rhapsody auf Display Postscript, jedoch entsprach die Oberfläche optisch größtenteils der des alten Mac OS. Die oben beschriebenen Schwierigkeiten bei der Ausführung von Mac OS Software führten dazu, dass Rhapsody nie als Mac OS-Nachfolger veröffentlicht wurde. Aus Rhapsody entstand jedoch „Mac OS X Server“.

Es kommt alles zusammen: Mac OS X

Alle bis jetzt beschriebenen Systeme und Architekturen gehören zur Vorgeschichte von Mac OS X. In ihm werden sie vereint. Mac OS X v10.0 wurde im März 2001 veröffentlicht und ist eine Weiterentwicklung von Rhapsody. Als Kernel kam nun XNU zum Einsatz.

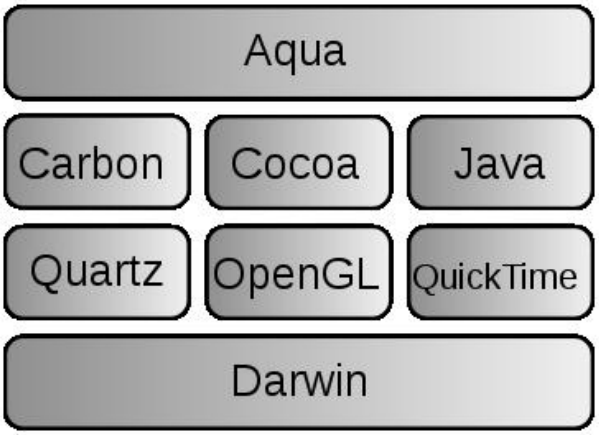
XNU

Wie oben erwähnt, heißt der Mac OS X Kernel „XNU“ (Akronym für „X ist Not Unix“). Er wurde ursprünglich für NeXTSTEP entwickelt, als Hybridkernel, der den Mach Kernel (v2.5) mit Komponenten aus 4.3BSD und einer objektorientierten API zur Programmierung von Treibern -genannt „Driver Kit“ - kombinierte. Nachdem Apple NeXT übernommen hatte, wurde die Mach-Komponente von XNU auf Version 3.0 aktualisiert. Da Apple Teile des Systems (siehe Darwin) unter quelloffener Lizenz veröffentlichen wollte, musste das System von sämtlichem originalen Unix-Code bereinigt werden. Wie oben unter „BSD“ bereits erwähnt, war dieser Schritt bei 4.4BSDlite/FreeBSD bereits vollzogen, weshalb es sich anbot, entsprechende BSD-Komponenten daraus zu übernehmen. Das Driver Kit wurde durch die API „I/O Kit“ ersetzt. XNU ist ein Hybridkernel mit den Eigenschaften eines monolithischen sowie eines Mikrokernels und versucht die Vorzüge beider Arten zu vereinen.

Mach realisiert Threads, Tasks, Speicherschutz, Echtzeitunterstützung und die Ein/Ausgabe per Konsole.
Die BSD-Teile von XNU realisieren die POSIX API, Sicherheitsrichtlinien, Benutzer- und Gruppen-IDs, den Netzwerk-Stack, Dateisysteme, Inter-Process Communication. Große Teile der BSD-Komponenten wurden mittlerweile geändert und weiterentwickelt, es gibt aber immer noch einen Austausch von Code zwischen Apple und dem FreeBSD Projekt.

Die Architektur von Mac OS X besteht aus vier Schichten:

- Benutzeroberfläche: *Aqua*
- Programmierschnittstellen: *Carbon, Cocoa und Java*
- Grafik-Subsystem: *Quartz, OpenGL und QuickTime*
- *Darwin*



Darwin

Darwin bildet das Basissystem von Mac OS X. Es enthält neben dem Kernel XNU noch ein BSD-Userland. Es steht unter einer freien Lizenz. Wichtige proprietäre Komponenten, wie die Aqua-Oberfläche, Carbon, Cocoa und QuickTime sind nicht enthalten. Die Rolle von Darwin ist am besten so zu verstehen, als dass Proprietäre Mac OS X im Kern sehr viele freie Komponenten (siehe FreeBSD) enthält und man dieses Basissystem darum wieder unter freier Lizenz veröffentlicht.

Quartz

Quartz bildet mit OpenGL und QuickTime die Grafikschicht von Mac OS X. Es besteht aus *Quartz 2D* und *Quartz Compositor*. Quartz 2D ist die Primäre Schnittstelle zum Rendern zweidimensionaler Elemente sowie Text. Quartz beherrscht Kantenglättung sowie Subpixel-Rendering - eine Technik zur Verbesserung der Schriftdarstellung auf Flachbildschirmen.
Quartz Compositor ist ein System zur Darstellung der Benutzeroberfläche, also dem (auf dem PDF-Format basierenden) Fenstersystem von OS X. Es wird neben von Quartz 2D auch OpenGL und QuickTime benutzt. In der OS X Version 10.2 kam Quartz Extreme hinzu. Quartz Extreme bewirkt eine deutliche Beschleunigung der Oberflächendarstellung, da nun mittels OpenGL die Grafikkarte direkt angesprochen werden kann: die einzelnen Fenster werden wie Texturen behandelt.

OpenGL

OpenGL steht für Open Graphics Library. Diese Spezifikation definiert eine Programmierschnittstelle, um 2D- und 3D-Computergrafiken zu produzieren. Sie ist programmiersprachen- und plattformunabhängig. Sie enthält etwa 250 Befehle, welche die Echtzeitdarstellung von komplexen dreidimensionalen Szenen ermöglicht. Die Basisoperation von OpenGL ist es, „Primitive“, wie Punkte, Linien und Polygone anzunehmen und diese in Pixel zu konvertieren. Dies erledigt die Grafikpipeline, auch bekannt als die „OpenGL State Machine“. Mittels GLSL kann diese um so genannte „Shader“ erweitert werden, damit werden verschiedene Zwischenschritte der Pipeline voll programmierbar.

QuickTime

Der Begriff QuickTime hat mehrere Bedeutungen: Das Dateiformat, die API sowie das Framework, letztere ist in diesem Kontext gemeint. Das Framework stellt folgende Funktionen zur Verfügung:

- Enkodieren und Transkodieren (umwandeln) von Video und Audio von einem ins andere Format
- Das Dekodieren von Video und Audio und das anschließende Senden des dekodierten Streams an Quartz Extreme zur Darstellung
- Eine Plugin-Architektur, um zusätzliche Codecs dritter zu unterstützen (z.B. DivX)

Carbon

Die Carbon API wurde entwickelt, um die Portierbarkeit von Anwendungen von Mac OS nach Mac OS X möglichst leicht zu machen. Anstatt wie noch in Rhapsodys Blue Box ein komplettes Mac OS zu emulieren, wurden dessen Schnittstellen direkt in der BSD-Schicht implementiert und verfügbar gemacht. Ausnahmen waren veraltete Schnittstellen oder solche, die elementaren Mac OS X-Konzepten, wie dem Speicherschutz im Wege standen. Beim Umstieg auf 64 bit, was mit der Mac OS X 10.6 geschah, wurde Carbon nicht mitportiert, somit seither nicht mehr unterstützt.

Cocoa

Cocoa ist die objektorientierte API zur Programmierung unter OSX und eine Weiterentwicklung von OpenStep. Daher haben alle Cocoa Klassen das Prefix „NS“, ein Erbe aus der OpenStep- bzw NeXT/Sun-Zeit. Eine Besonderheit von Cocoa ist seine gute Verwaltung von dynamisch allokiertem Speicher. Unter Mac OS X (aber nicht unter iOS) gibt es einen optionalen „garbage collector“, der nicht mehr benötigte bzw. referenzierte Objekte automatisch löscht. Cocoa besteht aus drei Frameworks:

Foundation

Im Foundation Framework wird eine Basissammlung an Objective-C-Klassen definiert. Neben grundsätzlichen, einfachen Objektklassen (Strings, Arrays, Klassen für Zeichenketten, Mengen, Listen etc.) bringt es weitere Funktionalitäten mit, die nicht direkt von Objective-C abgedeckt werden. Auch die XML-Unterstützung und die Undo-Funktion werden darin realisiert. Bei der Entwicklung waren folgende Ziele gesetzt:

- eine kleine Sammlung von nützlichen Basisklassen zur Verfügung zu stellen
- einheitliche Konventionen für Vorgänge, wie z.B. die Deallokierung von Speicher zu schaffen und dadurch die Anwendungsentwicklung zu erleichtern
- Unicode Strings, Objektzugriff und - Persistenz zu unterstützen
- eine gute Portabilität (auf andere Geräte und Architekturen) zu ermöglichen

Die Klassenhierarchie von Foundation hat ihren Ursprung in `NSObject`. Von ihr erben alle anderen (Unter-)Klassen. Daneben gibt es einige individuelle Klassen. Es gibt so genannte „Class Clusters“: Öffentliche, abstrakte Klassen fungieren als Interfaces für eine Vielzahl von nichtöffentlichen Unterklassen. Ein Beispiel: `NSString` und `NSMutableString` fungieren als Vermittler für verschiedene nichtöffentliche Unterklassen, die für unterschiedliche Speicherungsarten optimiert sind. Je nach dem, mit welcher Methode nun ein String erzeugt wird, erhält dieser seine Umsetzung in Form einer speziell dafür optimierten Klasse. Für den Einsatz unter iOS wurde ein Teil der Klassen aus dem Foundation Framework entfernt.

Applikation Kit

Das Applikation Kit („AppKit“) baut auf Foundation auf und enthält Objekte bzw. Klassen (ebenfalls Unterklassen von `NSObject`), die für eine grafische, ereignisgesteuerte Benutzeroberfläche von notwendig sind: Fenster, Flächen, Buttons, Menüs, Scrollleisten und Textfelder. Für den Programmierer regelt das AppKit Details, wie das direkte Zeichnen auf den Bildschirm und die Kommunikation mit der Hardware. Mit dem „Interface Builder“ stellt man Verbindungen zwischen Objekten der Benutzeroberfläche und Objekten der Programmlogik einer Anwendung. Für Programmierer ist es möglich, eigene (erbende) Unterklassen zu erstellen und diese mit zusätzlicher Funktionalität zu füllen. Für Anwendungsentwickler wichtige Klassen sind `NSApplication`, `NSWindow`, und `NSView`. Für die Logik des AppKits wichtige Klassen sind `NSResponder` und das in Foundation enthaltene `NSRunLoop`. Des Weiteren erzeugt und nutzt jede Cocoa-Anwendung eine einzelne Instanz von `NSApplication`: Es regelt die „main event loop“ - die wichtigste Ereignisschleife.

Core Data

Core Data ist eine Modellierungsschicht und enthält Methoden zur Beschreibung von Modellstrukturen. Mittels dieses Frameworks ist es möglich, einen kompletten „Object Graph“ zu serialisieren. Ein Object Graph ist der Zustand und die Relationen eines Objekt Systems zu einem bestimmten Zeitpunkt, also z.B. alle Objekte eines Programms zur Laufzeit. Im Unterschied zu einem UML Klassendiagramm, welches Relationen zwischen Klassen darstellt, erfasst der Object Graph ihre konkreten, erzeugten Instanzen. Durch diese Möglichkeit zur Speicherung der konkreten Objekte und seiner Beziehungen gestaltet sich die Speicherung von Anwendungsdateien sehr einfach - es muss kein eigenes Speicherformat implementiert werden. Nach dem Öffnen bzw. Wiedereinlesen kann mit den üblichen Methoden auf den Objekten weitergearbeitet werden. Ebenso ist eine Undo-Funktionalität implementiert, also das Rückgängigmachen von Schritten und Aktionen in der Anwendung. Core Data kann Objekte nach XML, SQLite oder auch binär serialisieren - jede Methode hat Vor- und Nachteile.

Java

In Mac OS X wurde eine Java-Laufzeitumgebung (JRE) integriert. Diese enthält die Java Virtual Machine, mit der Java-Applikationen ausgeführt werden können. Jedes Java-Programm wird in seiner eigenen Java Virtual Machine (JVM) ausgeführt. Diese beaufsichtigt die Ausführung des Programms. Durch eine speziell auf das Betriebssystem zugeschnittene JRE bietet Java eine große Plattformunabhängigkeit, Java-Programme sind aber in der Regel nicht so performant wie speziell für das Betriebssystem kompilierte Anwendungen. Durch so genannte „Bridges“ ist es möglich Cocoa in Java (und in anderen Sprachen, wie Ruby oder Python) zu nutzen. Die Java-Bridge wird jedoch seit der OS X Version 10.4 nicht mehr aktualisiert.

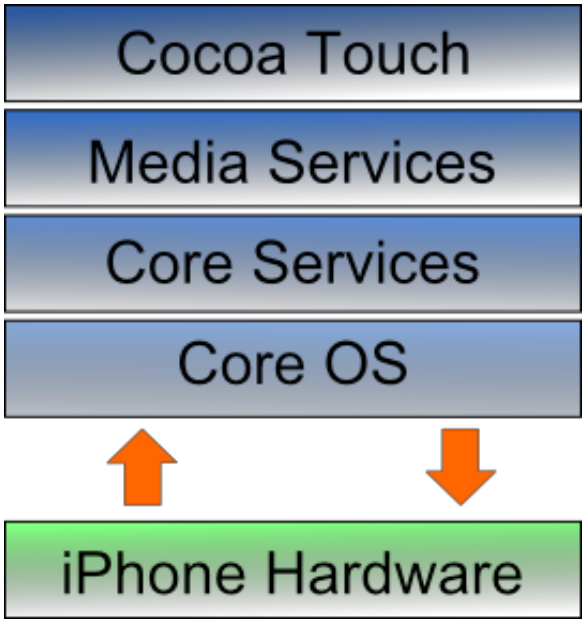
Aqua

Aqua ist die Benutzeroberfläche, die mit Mac OS X eingeführt wurde. Sie arbeitet mit Fenstern und wird stetig weiterentwickelt. Die Darstellung erfolgt mittels Quartz.

iOS: eine Weiterentwicklung von Mac OS X

Das Betriebssystem iOS, damals noch „iPhone OS“ genannt, wurde Anfang des Jahres 2007 zusammen mit dem neuen Smartphone namens iPhone vorgestellt. Es war ein so genanntes „Derivat“ von Mac OS X, ein Abkömmling. Die Besonderheit für Endanwender war, dass das Gerät bzw. Betriebssystem fast vollständig mit den Fingern über den Sensorbildschirm bedient wurde. So genannte Multitouch-Gesten ermöglichten sogar die Steuerung mit mehreren Fingern gleichzeitig. Auch die Texteingabe erfolgt mittels einer eingeblendeten Tastatur.
Was bei iOS im Vergleich zum „großen“ Mac OS X auffällt, sind die Einschränkungen, denen es unterliegt. Diese sind weniger entwicklungsgeschichtlicher Natur, sondern Bewusste Designentscheidungen, teilweise wettbewerblich motiviert. So können neue Anwendungen („Apps“) nur über den online verfügbaren „App Store“ bezogen werden. Erst seit iOS Version 4 ist bestimmten Programmen, die nicht zur Standardkonfiguration gehören, Multitasking bzw. paralleler Betrieb erlaubt. Mittels einem so genannten „Jailbraik“ können viele dieser Einschränkungen ausgehebelt werden. Auch wenn Apple dies nicht erlaubt, hat sich darum eine lebendige Gemeinschaft gebildet. In den Lizenzbestimmungen behält sich Apple sogar das Recht vor, installierte Apps auf iOS-Systemen aus der Ferne zu deaktivieren oder gar zu löschen.

Das iOS besteht aus mehreren Softwareschichten, jede von ihnen stellt Schnittstellen und Frameworks für die Entwicklung von Anwendungen zur Verfügung, die auf dem Betriebssystem laufen.



Die Einzelnen Softwareschichten von oben nach unten:

Cocoa Touch-Schicht

Die Cocoa Touch-Schicht ist die oberste der vier Softwareschichten und enthält die meisten der allgemein von Anwendungsentwicklern genutzten Frameworks. Cocoa Touch ist hauptsächlich in Objective-C geschrieben und basiert auf der Cocoa API von Mac OS X. Diese wurde an die Anforderungen des iPhones angepasst.
Die Cocoa Touch-Schicht stellt folgende Frameworks für Anwendungen bzw. die Anwendungsentwicklung zur Verfügung:

Das *UIKit Framework* bietet eine große und funktionsreiche auf Objective-C basierende API. Sie ist die wohl meistgenutzte bei der iOS-Programmierung. Wichtige Funktionen sind:

- Erstellen und Verwalten von grafischen Benutzeroberflächen (Textfelder, Buttons, Farben, Schriften usw.)
- Behandlung von Programmzyklen und Ereignisbehandlung (z.B. Interaktion per Touchscreen)
- Realisierung einer Programmübergreifenden Zwischenablage
- Datenverwaltung
- IPC (Kommunikation zwischen verschiedenen Prozessen)
- Das „Push Notifications"-Benachrichtigungssystem, bei dem das Gerät von einem Server Benachrichtigungen über Ereignisse in bestimmten Diensten erhalten kann. Die Realisierung geschieht in Verbindung mit dem *Push Notification Service* (siehe unten)
- Nutzung der Daten von Beschleunigungssensor (z.B. iPhone), Batterie, Gyroskop, Kamera etc.

Das *Map Kit Framework* stellt eine API zur Verfügung, mithilfe derer Kartenbasierte Funktionen in Anwendungen realisiert werden können, z.B. das scrollen durch eine Karte oder die Darstellung einer Karte, abhängig von der aktuellen geografischen Position des Geräts sowie das Überlagern der Karte mit zusätzlichen Informationen.

Der *Push Notification Service* erlaubt es, den Benutzer über ein eingehendes Ereignis zu benachrichtigen, ohne dass die entsprechende Anwendung gerade aktiv sein muss. Es kommt beispielsweise bei Instant-Messaging-Programmen zum Einsatz. Wird eine neue Nachricht versendet, so macht diese einen Umweg über einen (Apple-)Server, welcher dann die Push Notification mit dem konkreten Inhalt an das Gerät übermittelt. So muss anstatt vieler einzelner Anwendungen nur ein Dienst im Hintergrund laufen.

Das *Message UI Framework* ermöglicht es, Emails aus beliebigen Programmen zu Versenden. Die dazugehörigen Benutzeroberflächenelemente, z.B. für die Adresseingabe werden ebenfalls bereitgestellt

Das iPhone als wichtigstes und erstes Gerät und somit iOS sind auf Kommunikation nach außen ausgelegt - für den Zugriff auf das Adressbuch und seine Daten kommt so ein eigenes Framework zum Einsatz, das *Address Book UI Framework*. Mit diesem kann man aus jeder

Anwendung heraus auf Adress- und Kontaktinformationen zugreifen, diese Anzeigen und auch bearbeiten.

Mittels des *Game Kit Framework* können Anwendungen Peer-to-Peer-Verbindungen zwischen mehreren Geräten herstellen. Diese können miteinander interagieren. Die Namensgebung rührt aus der Erwartung, dass vor allem so genannte Multiplayer-Spiele von dem Framework Gebrauch machen würden.

Media-Schicht

Die Aufgabe der Media-Schicht ist das Bereitstellen von Audio- (*AV Foundation framework*), Video- (*Media Player framework*), und Grafikfunktionalitäten(*OpenGL ES framework*) sowie das Zeichnen zweidimensionaler Elemente (*Core Graphics Framework*) und deren Animation (*Quartz Core Framework*).

Core Services-Schicht

Die Core Services-Schicht stellt viele Grundlagen zur Verfügung, auf denen höhere Schichten aufbauen. Dazu gehören Datenbankfunktionen (*SQLite library*), Zugriff auf den „App Store" bzw. Transaktionen innerhalb von Apps (*Store Kit Framework*) und Datenverwaltung (*Core Data Framework*).

Core OS-Schicht

Die Core OS-Schicht sitzt direkt über der Hardware. Sie enthält den Kernel (modifizierter Mach-Kernel) und stellt vielfältige Dienste zur Verfügung, z.B. Netzwerkfunktionalität (*CFNetwork Framework*), Zugriff auf externes Zubehör (*External Accessory framework*) sowie gewöhnliche, fundamentale Betriebssystemfunktionalitäten wie Sicherheitsfunktionen (*Security Framework* mit öffentlichen/privaten Schlüsseln, Sicherheitsrichtlinien und Verschlüsselungsmethoden) Speicherverwaltung, Dateisystemverwaltung und Threads (*LibSystem*).

Fazit

Im Verlauf dieser Ausarbeitung konnte ein Einblick in den Aufbau von Mac OS X bzw. iOS gewonnen werden. Es wurden die wichtigsten Kernkomponenten vorgestellt und ihre Funktionalität beleuchtet. Wie man sehen konnte, hat Mac OS X eine lange (Unix-) Vorgeschichte; Zustandekommen und Ursprung einzelner Komponenten reichen verhältnismäßig weit in die Geschichte der Betriebssysteme zurück und sind teilweise durchaus komplex und nicht geradlinig. Dabei hat man sich immer wieder Altlasten entledigt und neue Komponenten integriert. Mac OS X hat dadurch seinen UNIX-Charakter nicht verloren (noch immer wird nach Unix-Manier bei Sicherheitsrelevanten Operationen das Kennwort abgefragt) und ist zu einem modernen Multiprozessor- und Multiuser-System geworden. Unter Endnutzern und Fachleuten gilt OS X als stabil und sicher - auch im gegenüber dem großen Konkurrenten Microsoft Windows. Dieser Eindruck besteht nicht zuletzt aber auch wegen der langjährig geringen Verbreitung von Mac OS X und der damit einhergehenden geringen Attraktivität

des Systems für Autoren von Schadsoftware. Auch mit iOS als direktem Abkömmling ist man gut aufgestellt - an diesem messen sich die Konkurrenten von Android, Windows Phone & Co. Mit seiner Ausrichtung als „Internetbetriebssystem“ - die meisten iOS-Anwendungen setzen eine Netzverbindung voraus - folgt man den Trend zur Zentralisierung. Apples angekündigter Dienst „iCloud“ wird dieses Prinzip noch weiter treiben. Bedenklich bzw. möglicherweise hemmend für die Weiterentwicklung und das Aufkommen von Innovationen sind Apples restriktive Politik und die proprietäre Veröffentlichung seiner Betriebssysteme.

Quellen:
Amit Singh: Mac OS X internals: a systems approach (Hauptquelle, genutzt u. A. für Mach, XNU, Darwin, Rhapsody, Carbon, Quartz)

http://developer.apple.com (zweite Hauptquelle, genutzt u. A. für Cocoa)

http://en.wikipedia.org (u.A. OpenGL, Unix-Einteilung, Kerneltypen, Grafik Kernel, Grafik Mac OS X Architektur)

http://cm.bell-labs.com/cm/cs/who/dmr/hist.html
http://www.bell-labs.com/history/unix/ (UNIX-Geschichte)

http://www.eff.org/deeplinks/2010/03/iphone-developer-program-license-agreement-all
http://www.techotopia.com/index.php/The_iPhone_OS_Architecture_and_Frameworks (iOS, Grafik iOS-Architektur)