Universitatea Politehnica Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Sistemul de operare Mac OS X Mountain Lion
- Sisteme de operare avansate -

Lăcătuşu Raluca - Cristina
Master IISC
An I

Bucureşti 2014

1. Arhitectura MAC OS X

Din punct de vedere al utilizatorului, sistemul MAC OS X este interfaţa sa cu utilizatorul, aplicaţii şi servicii. Pentru dezvoltatori, interfaţa este doar o faţadă, în spatele ei existând sistmeul de operare Mac OS X, o reţea complex de software care se ocupă de interacţiunile dintre cererile utilizatorilor şi a resurselor de calcul.

Inima acestui sistem software este kernel-ul. Kernel-ul oferă servicii de calcul de bază pentru sistemul de operare, cum ar fi manipularea întreruperilor, managementul proceselor şi gestionarea memoriei. Două tipuri de kernel constituie baza pentru majoritatea sistemelor de operare: kernel-ul monolitic şi microkernel-ul. Un kernel monolitic încapsulează aproape toate straturile sistemului de operare într-un program care rulează pe spaţiul kernel-ului. Un microkernel implementează un subset de servicii pentru sistmeul de operare, rulând pe spaţiul kernelul-ului şi este mult mai mic decât kernel-ul monolitic.

Serviciile suplimentare, implementate peste kernel, ca programele instalate de utilizator (care rulează în spaţiul de utilizare), au interfeţe bine definite. Pentru a efectua un serviciu care se află în afara spaţiului dedicate kernel-ului, kernel-ul comunică cu serviciul la nivel de utilizator prin transmiterea de mesaje. În general, un kernel monolitic este mai rapid, dar şi mai mare decât un microkernel.

MAC OS X-ul original a fost mai mult o colecţie de servicii de sistem care cooperau, a căror proiectare nu era împărţită între domeniile utilizator şi kernel. În plus, manipularea sarcinilor critice ale sistemului de operare, cum ar fi gestionarea memoriei şi a proceselor era învechită, ceea ce a condus Apple să dezvolte alternative pentru sistemele de operare viitoare.

De exemplu, în ziua de azi, suntem familiarizaţi cu sisteme de operare care folosesc multitasking-ul preventiv. MAC OS X a implementat o planificare numită multitasking de cooperare. Aceasta funcţionează după cum urmează: atunci când se execută un program, sistemul de operare încarcă programul în memorie, îl programează pentru execuţie pe UCP şi rulează programul numai dacă programul care rulează în acel moment cedează UCP-ul. Este responsabilitatea fiecărui program să cedeze UCP-ul când este nevoie – nu este responsabilitatea sistemului de operare.

Această planificare este suboptimală, să predea ocazional UCP-ul pentru a permite altor programe să ruleze, deoarece un program poate monopolize UCP-ul şi poate refuza altele programe care rulează. MAC OS X este construit pe UNIX şi, prin urmare, foloseşte multitasking-ul preventiv, kernel-ul gestionând politicile de planificare a proceselor.

O altă diferenţă între MAC OS X şi sistemele mai vechi Macintosh este gestionarea memoriei. MAC OS nu a pus în aplicare o protecţie a memoriei pentru

partiţiile sistemului sau pentru aplicaţii. Aplicaţiile sunt libere de a scrie în memorie, în afara spaţiului de adrese alocat şi acest lucru ar putea duce la blocarea altor aplicaţii, precum şi a întregului sistem. Sub MAC OS X, acest lucru nu este posibil: accesarea memoriei în afara spaţiului de adrese de către un program va duce la o defecţiune, iar procesul se va inchide, dar nu va bloca şi sistemul de operare sau alte procese.[1]

1.1 Straturile arhitecturii

Arhitectura MAC OS X este compusă din mai multe straturi, fiecare fiind responsabil pentru servicii de sistem diferite. Este important să se ţină cont de faptul că MAC OS X este construit peste kernel-ul de bază al UNIX-ului, care oferă serviciile sistemului şi suportă numeroase straturi ale aplicaţiilor cu care interacţionează utilizatorul.

Nucleul sistemului MAC OS X este Darwin, un sistem de operare open-source bazat pe Mach 3.0 şi 4.4BSD. Darwin este un sistem de operare complet, care nu are nevoie de componente de nivel superior Macintosh pentru a rula. Sistemul Darwin are două componente importante: kernel-ul mediu şi stratul de emulare BSD.

Mediul kernel-ului oferă servicii de sistem de operare de bază, iar stratul de emulare alimenreaza sistemul cu mediul de interacţionare cu utilizatorului BSD. Se poate instala Darwin pe un PowerPC sau pe o maşina x86 şi poate fi folosit ca un sistem BSD stand-alone. Componentele de sistem specifice Macintosh construite peste kernel-ul Darwin oferă sistemului MAC OS X servicii şi componente.

Putem să ne gândim la Darwin ca un sistem de operare bazat pe BSD, iar componentele Macintosh sunt puse peste acest sistem. Această clasificare permite observarea faptului că MAC OS X este construit peste Darwin, iar Darwin este un sistem UNIX complet.[2]

Componentele sistemului MAC OS X:

- Cel mai mic strat este Mach - kernel bazat pe BSD, numit şi mediul kernel-ului. Acesta oferă sistemul de servicii de sistem de operare de bază, cum ar fi gestionarea proceselor şi a memoriei, sisteme de fişiere, crearea de reţele, precum şi acces la dispozitiv şi controlul acestuia.
- Serviciile Core - Stratul de servicii de bază implementează un set central de rutine non-grafice care variază accesul la API-urile Macintosh . Acest strat include facilităţi pentru interacţiune cu fişierele de sistem, thread-uri, şi memorie, şi oferă rutine pentru manipularea şirurlori de caractere, accesul la resurse locale şi de la distanţă prin intermediul URL-uri, şi parsarea XML-urilor. Peste stratul de servicii de bază este stratul de servicii pentru aplicaţii.

Servicii aplicaţiilor aprovizionează programele care rulează în cadrul mediului de aplicare(cu excepţia BSD) cu interfaţă cu utilizatorul, ferestre, şi suport grafic. Acest strat include managerul de fereastră Mac OS X.

- Mediul aplicaţiilor, cum ar fi serviciile aplicaţiilor, este compus din diferite medii de aplicare care dau sistemului nivelul pentru utilizator. În prezent, Carbon, Cocoa, Clasic, Java, şi BSD formează acest strat, fiecare ca un mediu de aplicare separată. Fiecare oferă un mediu distinct de execuţie în care se pot rula programe şi se poate interacţiona cu straturile inferioare ale sistemului de operare. De exemplu, atunci când se execută un program Cocoa, ne aflăm în mediul specific Cocoa; atunci când se execută un program de Mac OS se va interacţiona cu mediul de aplicare Classic.
- Peste mediul de aplicare este Aqua, interfaţa cu utilizatorul Mac OS X. Aqua oferă sistemul Mac OS X şi programelor design-ul specific.[1]

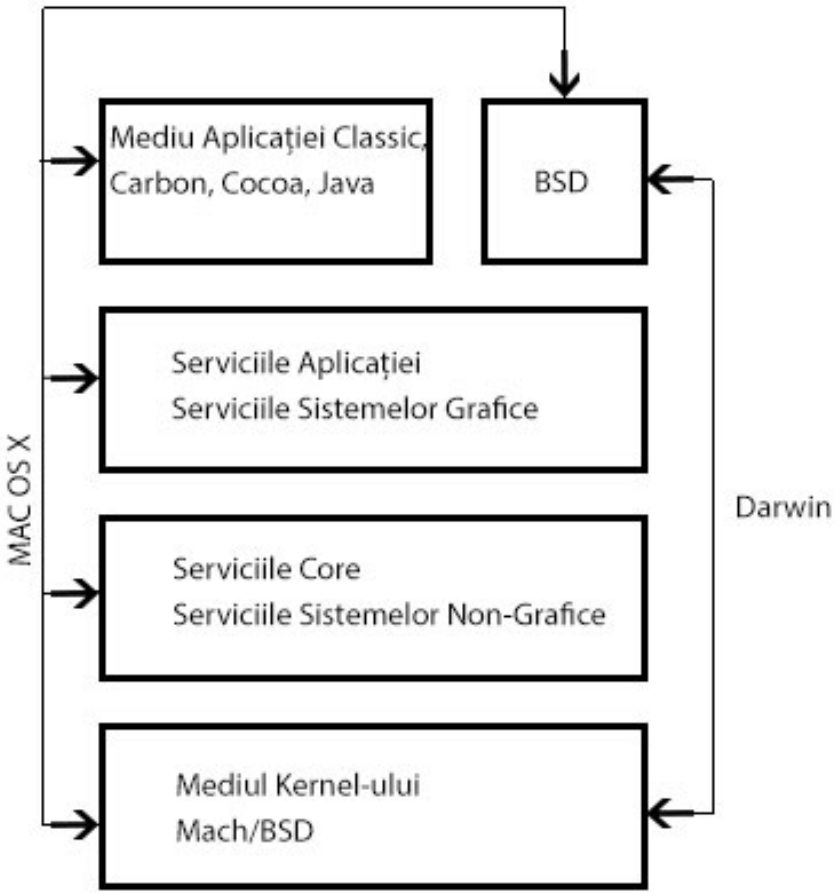

Figure 1.1 - Mac OS X este format din straturi, fiecare strat furnizând servicii pentru stratul de mai jos[1]

1.2 Mediul Kernel-ului

Mediul de kernel furnizează Mac OS X-ului serviciile sistemului de operare de bază. Acest strat este compus din două substraturi: nucleul de Mach şi stratul BSD, care cuprinde Mach. În cadrul acestor straturi sunt cinci componente principale: Mach, kit-ul I/O, BSD, sistemul de fişiere, şi reţeaua .

Mach

MAC OS X foloseşte Mach 3.0 microkernel (Mach 3.0 + OSF/îmbunătăţiri Apple). Porţiunea Mach din mediului kernel-ului este responsabilă pentru gestionarea proceselor şi a memoriei(inclusiv memoria virtuală şi memoria de protecţie), pentru multitasking-ul preventiv, precum si de gestionarea mesajelor între straturile sistemului de operare. Mach, de asemenea, controlează şi mediază accesul la resursele de calcul de nivel scăzut. Acesta îndeplineşte următoarele sarcini:

- Oferă infrastructură IPC(Inter Process Communication), metode(cozi de mesaje, RPC(Remote Procedure Call)) care facilitează comunicare sistemului de operare;
- Gestionează procesorul prin programarea execuţiei şi prevenirea thread-urilor care alcătuiesc un task;
- Suportă SMP(multiprocesare simetrică);
- Gestionează problemele referitoare la memoria low-level, precum şi memoria virtuală. Mach nu are cunoştinţă de sistemul de fişiere, networking, dar nici de lucrurile specifice sistemului de operare.

Mach implementează un set foarte mic de servicii de bază de sistem în spaţiul de adrese al kernel-ului, comunicând cu servicii suplimentare în spaţiul de utilizare prin interfeţe bine definite. Implementarea kernel-ului pentru Darwin integrează multe dintre aceste servicii.[1]

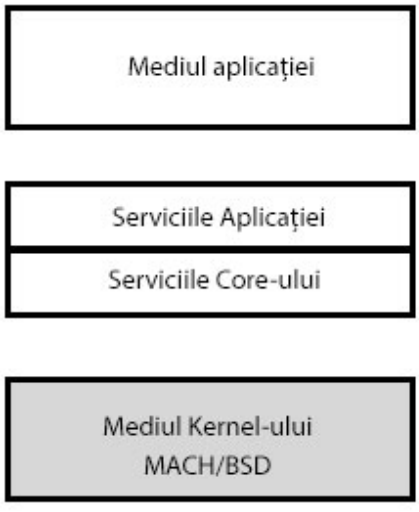

Figura 1.2 - Mediul kernel-ului MAC OS X furnizează sistemului serviciile necesare[1]

Există o diferenţă fundamentală între modul în care un kernel monolitic UNIX şi kernel-ul Mach utilizează şi implementează procese şi thread-uri. Într-un kernel UNIX, nivelul de bază de planificare este procesul, nu firul de execuţie. Toate thread-urile în cadrul procesului sunt legate de prioritatea şi de planificarea procesului şi nu sunt văzute de către kernel ca entităţi programabile. De exemplu, dacă sistemul de operare suspendă un proces, toate firele sale de execuţie sunt, de asemenea, suspendate.

Pe de altă parte, Mach împarte conceptul de proces UNIX în două componente: un task şi un fir de execuţie. Un task conţine execuţia unui program şi fire sale de execuţie. Pentru Mach, thread-ul este unitatea de bază de planificare, spre deosebire de un proces UNIX, care foloseşte procedeul ca unitate de programare. Sub Mach, prioritate de programare este gestionată pe o bază per-thread: coordonează sistemul de operare coordonează si programează thread-urile pentru unul sau mai multe task-uri.

Kit-ul I/O

Kit-ul I/O este un framework obiect-orientat folosit pentru dezvoltarea driverelor MAC OS X, implementat într-un subset a C++-ului. Dezvoltarea de drivere este un task specializat, acest lucru necesitând cunoştinţe detaliate, experienţă şi cod foarte specializat. Kit-ul I/O încearcă să crească reutilizarea codului şi reducerea efortului de învăţare pentru dezvolarea prin furnizarea unui framework care încapsulează funcţionalitatea funcţiilor de baza ale dispozitivelor în clase de bază, care sunt extinse pentru a implementa drivere specifice. [3]

BSD

O altă componentă a kernel-ului Darwin este implementarea BSD-ului, care este bazată pe 4.4BSD. Componenta kernel-ului BSD este dezvoltată peste kernel-ul Mach modificat care rulează în spaţiul de adrese a kernel-ului. Această componentă furnizează servicii de reţele, sisteme de fişiere, politicile de securitate, API-ul kernel-ului FreeBSD şi API-ul POSIX pentru susţinerea aplicaţiilor pentru utilizator. Acesta oferă, de asemenea, aplicaţii cu interfaţă BSD pentru serviciile de bază ale sistemului de operare. În mod normal, design-ul microkernel-ului plasează multe dintre aceste componente BSD (cum ar fi sisteme de fişiere şi reţele) în spaţiul de utilizare, nu în spaţiul kernel-ului. Darwin nu este un microkernel pur. Pentru a rezovla problemele de performanţă, dezvoltatorii au modificat kernel-ul prin introducerea unor module de sistem BSD în spaţiul acestuia, rezervat în mod tradiţional pentru Mach.[1]

Sistemul de fişiere

Infrastructura sistemului de fişiere pentru Darwin se bazează pe un sistem îmbunătăţit de fişiere virtuale(VFS) şi include suport pentru HFS(sistem de fişier ierarhic), HFS+(sistem de fişiere ierarhic +), UFS(sistem de fişiere UNIX), NFS(sistem de fişiere de reţea) şi ISO9660.

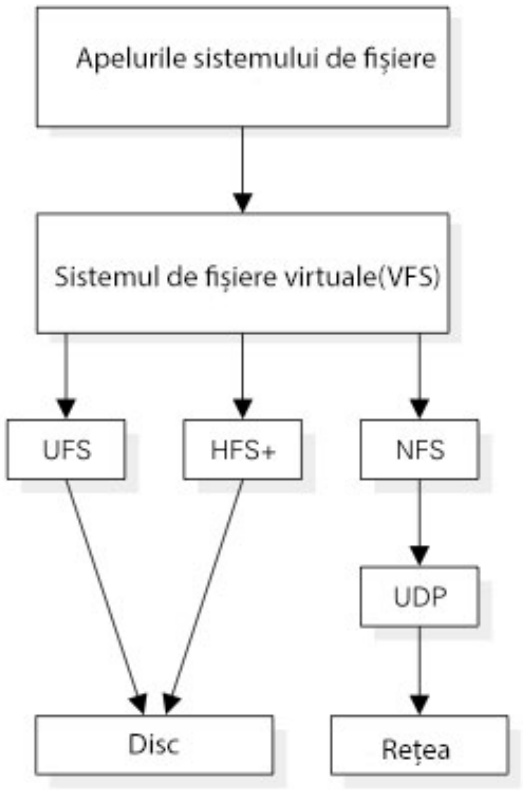

Figura 1.3 - Kernelul Darwin implementeaza un sistem Virtual de fisiere(VFS) care traduce apeluri de sistem in apeluri care pot fi intelese de sistemul de fisiere [1]

VFS este o component la nivel de kernel, care oferă o vedere abstractă a sistemelor de fişiere fizice, printr-o interfaţă comună. VFS acceptă apelurile de sistem legate de fişiere (deschide, închide, citire şi scriere) şi le traduce în apelurile corespunzătoare pentru sistemul de fişiere ţintă (a se vedea figura de 1.3). VFS este adesea menţionat ca ajutorul stivei de fişiere sistem, deoarece acesta poate interacţiona cu acesta şi poate adăuga multe tipuri de fişiere sistem, suportând augumentarea fişierelor de sistem deja existente cu cod customizat care furnizează numeroase servicii(ex: encriptarea).[5]

Networking

Infrastructura reţelei pentru Darwin se bazează pe 4.4BSD. Aceasta include toate caracteristicile pe care le poate avea un sistem BSD derivat, cum ar fi rutarea, stiva TCP/IP, dar şi socket-uri BSD. Această componentă se află in stratul BSD al kernel-ului.

Extensiile kernel-ului(KEXT) şi extensiile de reţea ale kernel-ului(NKE)

Extensiile kernel (KEXTs) dau dezvoltatorilor posibilitatea de a accesa structuri de date interne din kernel, dar şi un plus de funcţionalitate. KEXT sunt încărcate dinamic în spaţiul kernel-ului fără recompilarea sau relinkuirea kernel-ului. Deoarece KEXT rulează în kernel-ul, un modul rulat incorect poate bloca sistemul.

Extensii Kernel de reţea (NKE) sunt un exemplu special de KEXT. Ele permit dezvoltatorilor să fie conectate in stratul de reţea ale kernel-ului şi să implementeze noi feature-uri sau permit modificarea funcţionalităţii existente. Asemena KEXT-urilor, acestea sunt încărcate dinamic în spaţiu kernel-ului şi nu au nevoie de recompilarea sau relinkuirea kernel-ului pentru a fi executate.

Împreună, aceste componente furnizează servicii de bază pentru Darwin, şi, prin extensie, Mac OS X. Un sistem complet Darwin adaugă o funcţionalitate BSD sau un mediu pentru aplicaţii peste core, furnizând setul de comenzi şi mediul de execuţie. Un sistem complet Darwin este o implementare a BSD-ului bazatpe UNIX care este capabilă să ruleze ca un sistem de operare stand-alone. Darwin poate fi rulat pe un sistem compatibil PowerPC sau x86.[6]

1.3. Serviciile Core

Stratul serviciilor Core se afşă peste kernel şi este responsabil pentru serviciile non-grafice ale sistemului (*fig. 1.4*). Operaţiile comune nu sunt incluse în API-ul Macintosh (Carbon şi Cocoa); în schimb, layer-ul serviciilor Core implementează un singur cod de bază care permite accesul la API-ul Macintosh. Dezvoltatorii folosesc API-urile Carbon şi Cocoa pentru implementarea aplicaţiilor. Aceste servicii sunt implementate în următoarele componente:

- *Carbon Managers* - un set de servicii grupate, care implementează rutinele aplicaţiilor;
- *Core Foundation* - o librărie care furnizează servicii low-level cum ar fi internalizarea sau servicii XML;
- *Open Transport* - un set de rutine folosite pentru protocoalele reţelelor;

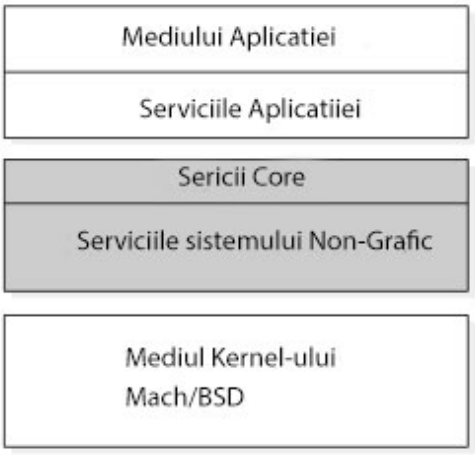

Figura 1.4. Straturile Serviciilor Core(stratul software peste Kernel) furnizeaza rutinele non-grafice, comune pentru API-ul Macintoch(Carbon si Cocoa)[1]

1.4. Serviciile Application

Layer-ul numit Application services alimentează sistemul cu serviciile grafice care sunt folosite pentru construirea interfeţelor şi a ferestrelor, precum şi pentru a efectua operaţii de desen, imprimare.

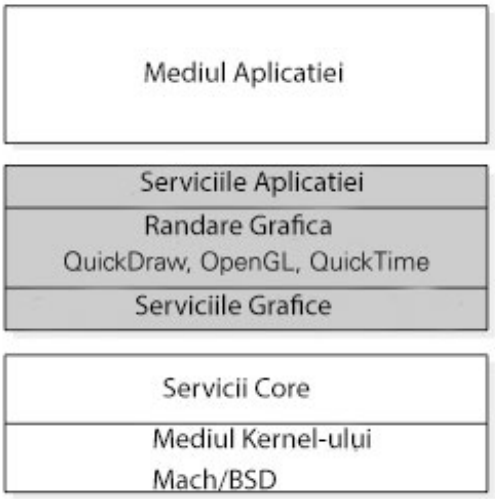

Fig. 1.5. Stratul Serviciilor Aplicatiei furnizeaza sistemului Mac OS X aplicatii grafice care randeaza QuickDraw, OpenGL si OuickTime[1]

Componenta principală a acestui layer este Quartz. Termenul Quartz defineşte tehnologia folosită pentru Mac OS X. Quartz este compus din două straturi: serviciile grafice şi bibliotecile de randare.

Serviciile grafice de randare care sunt definite peste serviciile core conţin următoarele biblioteci, care efectuează operaţiile de randare grafică:

- *Core Graphics Rendering library* - efectuează operaţii cu 2 dimensiuni. Această librărie este folosită pentru a desena şi a randa folosind vectorul de model.
- Quick Draw
- OpenGL
- QuickTime
- PDF

1.6. Application Environment

Stratul Application Environment oferă utilizatorilor de Mac OS X mediul în care aceştia pot dezvolt, implementa şi rula aplicaţii. Uneori, acest strat mai este cunoscut sub numele de Software Emulation, deoarece conţine medii de dezvoltare pentru sisteme de operare diferite. În practică se poate simula aproape orice sistem de operare pe acest cadru, cum ar fi Solaris, Windows sau MS-DOS. La momentul actual, 5 medii de dezvoltare sunt livrate cu Mac OS X: Classic, Carbon, Cocoa, Java şi BSD.[7]

2. Terminal

Terminalul este un emulator pentru MAC OS X. Îşi are originea în predecesorii OS X, NestStep şi Open Step. Un terminal emulator este un sistem bazat pe text. Oferă un mediu propice pentru shell-urile UNIX, care permit utilizatorilor să interacţioneze cu sistemul de operare, prin interfaţa linie de comandă. Terminal.app este folosit pentru a accesa sistemul de operare prin care se execută. Această aplicaţie este folosită de către utilizatorii care au nevoie să acceseze sistemul de operare la nivel low-level.

Liniile de comandă poti fi compuse dintr-un singur cuvânt sau pot conţine mai multe cuvinte cheie. Comanda poate fi numele unui program UNIX sau poate fi o comandă care este contruită într-un shell.

O comandă Unix poate avea şi argumente. Un argument poate fi opţiune sau un nume de fişier. Formatul general pentru o linie de comandă Unix este: *comandă opţiuni numeFişier.* Nu există un set de reguli prin care o comandă este executată sau contruită, dar există cateva reguli generale:

- Opţiunile sunt de obicei formate doar dintr-o singură literă, având ca şi prefix o cratimă. Opţiunile multiple pot fi setate individual într-o linie de comandă şi pot fi apoi combinate (*-a -b*). Unele comezi au opţiuni contruite din cuvinte sau fraze şi încep cu două cratime, cum ar fi *--delete* sau *--confirm-delete*;
- Comenzile trebuie să fie introduse cu litere mici. Unix este case-sensitive, astfel ca ”echo” şi ”ECHO” sunt diferite;
- Argumentul fişierului este numele fişierului care se doreşte a fi folosit. Cele mai multe programe Unix acceptă nultiple argumente de acest fel, separate prin spaţii. Unele comenzi, precum *who*, au argumente care nu sunt nume de fişiere;
- Între comenzi, opţiuni şi nume de fişiere trebuie să existe spaţii;
- Opţiunile apar înainte de numele de fişiere;
- În unele cazuri, o opţiune are un alt argument asociat. Comanda de sortare este un exemplu potrivit pentru acest caz: *$sort -o sorted -n sortme;*
- Liniile de comanda pot avea caractere speciale. Acestea pot include, de asemenea, mai multe comenzi separate. De exemplu, puteţi scrie două sau mai multe comenzi pe aceeaşi linie de comandă, fiecare separate prin punct şi virgulă (;). Comenzile introduse în acest fel sunt executate una după alta de shell.

Terminalul este un mod ideal pentru conectarea de la distanţă cu alte sisteme UNIX, chiar şi cu alte sisteme Macintosh. Utilizatorii pot deschide o conexiune remote din Terminal. De obicei, este recomandată folosirea serviciului Secure

Shell(SSH). [4]

3. Mach, BSD, XNU

Kernelul Mach a apărut la Universitatea Carnegie Mellon, și inițial s-a bazat pe un sistem de operare numit ”Accent”. Inițial a fost construit folosind kernel-ul 4.2BSD. Primele versiuni ale acestui kernel au fost monolitice, similar cu XNU, având cod BSD și Mach. Acest kernel a fost proiectat din nevoia de a avea suport multi-procesor. De asemenea, a fost proiectat ca un Micro-kernel.

Task-uri

O sarcină este o reprezentare logică a unui mediu de execuție. Sarcinile sunt utilizate pentru a împărți resurse de sistem între programele care rulează. Fiecare task are propria spațiu de adrese virtuale, dar și un nivel de privilegii. Fiecare task are unul sau mai multe thread-uri. Spațiul de adrese și resursele sunt împărțite între thread-uri.

Thread-uri

În Mach, un thread este o entitate independentă de execuție. Fiecare fir de execuție are proprii registri și politici de planificare. Fiecare thread are acces la toate elementele task-ului care îl conține. API-ul Mach-ului oferă o varietate de funcții pentru lucrul cu thread-uri. Prin acest API, se pot crea noi thread-uri, se poate înregistra conținut sau se poate șterge.[3]

Mesaje

Mesajele sunt folosite în Mach pentru a asigura comunicarea între fire. Un mesaj este compus dintr-o colecție de obiecte de date. Odată ce un mesaj este creat, acesta este trimis la un port pentru care task-ul are drepturile potrivite. Drepturile asupra porturilor pot fi trimise între task-uri ca și mesaje. Mesajele sunt trimise în coada destinației și sunt procesate de către thread. Pentru Mac OS X, funcția msg() este folosită pentru a trimite și a primi mesaje de la un port la altul. Declarația aceste funcții este:

```
mach_msg_return_t mach_msg
     (mach_msg_header_t       msg,
      mach_msg_option_t       option,
      mach_msg_size_t         send_size,
      mach_msg_size_t         receive_limit,
      mach_port_t             receive_name,
      mach_msg_timeout_t      timeout,
      mach_port_t             notify);
```

Porturi

Un port este un canal de comunicare cu kernel-ul. Aceasta oferă posibilitatea

de a transmite mesaje între thread-uri. Un fir de execuţie care are drepturile potrivite pentru un anumit port este capabil să transmită mesaje către acesta. Diverse porturi care au permisiunile necesare sunt capabile să trimită mesaje către un singur port în mod concurent. Totuşi, doar un task poate primi mesaje de la un port la un moment dat. Fiecare port are asociată o coadă de mesaje.

Apeluri către sistem

Pentru a combina Mach şi BSD într-un singur kernel, numerele syscall sunt împărţite între tabele diferite. Pentru un sistem PowerPC, unde instrucţiunea "sc" este executată, iar numărul syscall este stocat în r0 şi este utilizat pentru a determina care syscall să fie executat. Numerele pozitive syscall(mai mici decât 0x6000) sunt tratate ca şi syscall-uri FreeBSD. Codul de mai jos este extras dintr-o sursă xnu şi arată procesul descris mai sus:

```
oris r15,r15,SAVsyscall>>16
cmplwi r10,0x6000
stw     r15,SAVflags(r30)
beq--   cr6,exitFromVM
beq--   ppcscall
mr.     r0,r0
mtmsr   r11
blt--   .L_kernel_syscall
lwz     r8,ACT_TASK(r13)
cmpwi   cr0,r0,0x7FFA
beq--   .L_notify_interrupt_syscall
```

XNU

XNU este nucleul sistemului de operare cumpărat de Apple Inc. A fost dezvoltat pentru a fi utilizat pentru sistemul de operare Mac OS X şi a fost lansat ca software open source ca parte a sistemului de operare Darwin. XNU este acronimul pentru *X is Not UNIX.[2]*

Iniţial, XNU a fost dezvoltat de către NeXT pentru sistemul de operare NeXTSTEP. XNU a fost un kernel hibrid, fiind o combinaţie între versiunea 2.5 a kernel-ului Mach dezovltat de Universitatea Carnegie Mellon având componente precum 4.3BSD şi un API Objective-C pentru scrierea driverelor Driver Kit.

După ce Apple a achiziţionat NeXT, componenta Mach a fost upgradată la 3.0., componentele BSD au fost şi ele îmbunătaţite, fiind ataşat cod din proiectul FreeBSD, iar Driver Kit a fost înlocuit cu un API C++, folosit pentru driverele I/O Kit.

Ca orice kernel modern, XNU este un kernel hibrid, conţinând caracteristici specifice kernel-ului monolitic, dar şi caracteristici specifice microkernel-ului, încercând să obţină o performanţă ridicată. Astăzi, XNU rulează pe ARM, dar şi pe procesoarele x86-64.

BSD

Berkeley Software Distribution(BSD) este o parte a kernel-ului furnizează API-ul POSIX (apeluri către sistem BSD), procese UNIX, politici de securitate, id-urile

utilizatorilor și grupurilor, permisiuni, protocoale de rețea, codul sistemului virtual de fișiere, mai multe sisteme de fișiere locale cum ar fi HFS/HFS+, Network File System(NFS) pentru client și server, sistem criptografic, control de acces obligatoriu. Codul BSD prezent în XNU provine din kernel-ul FreeBSD. Deși o mare parte a codului a fost modificată, încă există cod comun între Apple și proiectul FreeBSD.[1]

K32/K64

XNU folosit începând cu versiunea Mac OS X 10.6 Snow Leopard(versiunea 10 Darwin) vine sub două forme: o versiune pe 32 de biți, numită K32 și o versiune pe 64 de biți numită K64. K32 poate rula aplicații pe 64 de biți. Ceea ce este nou în versiunea Mac OS X 10.6 este abilitatea de a rula XNU în spațiul asignat kernel-ului pe 64 de biți. K32 este kernel-ul implicit pentru 10.6 Server. K64 are câteva avantaje față de K32:

- poate gestiona mai mult de 32 GB RAM; pentru harta de memorie poate consuma o cantitate disproporționată din spațiul kernel-ului pe 32 de biți.
- dimensiunea buffer-ului cache-ului poate fi mai mare decât permite kernel-ul pe 32 de biți, crescând performanța I/O.
- performanța este crescută atunci când se utilizează dispozitive pentru rețea de o performanță ridicată sau când se utilizează GPU, deoarece kernel-ul poate mapa toate dispozitivele în spațiul de 64 de biți, chiar dacă au buffere DMA mult mai mari.

4. Managementul fişierelor

Fişierele şi directoarele sunt identificate prin numelelor. Un director este de fapt un tip special de fişier, astfel că regulile de denumire sunt aceleaşi ca şi pentru fişiere.

Numele fişierelor în UNIX pot conţine aproape orice caracter, în afară de caracterul „/”, care este rezervat pentru separarea fişierelor şi directoarelor în cale. Numele fişierelor sunt compuse din litere mare, litere mici, „.” şi „_”. Alte caractere(inclusiv spaţiul) sunt permise, dar pot îngreuna procesul, deoarece shell-ul acordă acestor caractere alte semnificaţii.

Deşi nu este recomandată folosirea spaţiului în denumirea fişierelor, mulţi utilizatori fac acest lucru. De asemena, este bine de ştiut că Finder nu agrează utilizarea „:” în numele fişierelor.

Dacă există spaţiu în numele fişierului, cuvintele despărţite vor fi interpretate de către shell-ul ca fiind comenzi. Ca acest lucru să fie evitat, putem pune numele fişierului între ghilimele, sau putem să adăugăm backslash-uri.

Exemplu: Pentru a redenumi un fişier care are un nume dificil de înţeles - prima comandă *rm* nu va merge, dar a doua va da rezultate. A treia comandă va exemplifica cum putem să facem *escape*.[4]

```
$ ls -l
total 2
-rw-r--r--  1 taylor staff 324 Jan 4 23:07 a confuing name
-rw-r--r--   1 taylor staff 64 Jan 4 23:07 another odd name
$rm a confusing name
rm: a: no such file or directory
rm: confusing: no such file or directory
rm: name: no such file or directory
$rm ”a confusing name”
$rm another\ odd\ name
```

Când avem un număr de fişiere într-o serie (ex: cap1.doc până la cap12.doc) sau fişiere cu caractere comune (ex: aegis, aeon, aerie) putem folosi metacaractere pentru a salva timp şi efort. Aceste caractere sunt *, ?, [].

Următoarele exemple arată cum pot fi folosite metacaracterele. Prima comandă listează toate intrările într-un director, iar celelalte folosesc metacaracterele pentru a afişa câteva intări.

```
$ ls
chap0.txt       chap2.txt       chap5.txt       cold.txt
chap1a.old.txt  chap3.old.txt   chap6.txt       haha.txt
chap1b.txt      chap4.txt       chap7.txt       oldjunk
$ ls chap?.txt
chap0.txt       chap4.txt       chap6.txt
chap2.txt       chap5.txt       chap7.txt
$ ls chap[3-7]*
chat3.old.txt     chap4.txt      chap5.txt      chap6.txt      chap7.txt
$ ls chap??.txt
chap1b.txt
$ ls *old*
chap1a.old.txt  chap3.old.txt   cold.txt        oldjunk
$ ls *a*a*
chap1a.old.txt   haha.txt
$ ls chap{3,6}.txt
chap3.txt        chap6.txt
$
```

Metacaracterele sunt folosite mai mult pentru afişarea fişierelor. Cele mai multe programe Unix acceptă mai multe nume de fişiere şi metacaracterele pot fi folosite pentru a numi multiple fişiere în linia de comandă. De exemplu comenzile *cat* şi *less* afişează conţinutul fişierelor, dar cat afişează tot conţinutul pe ecran, în schimb less afişează conţinut pe mai multe ecrane.

Metacaracterele sunt folosite şi pentru directoare. Pot fi folosite atât în căile relative cât şi în cele absolute. Folosirea metacaracterelor poate fi folositoare atunci când avem mai multe fişiere cu nume asemănătoare. Un exemplu clasic unde shell-ul este mult mai potrivit decât Finder-ul este acela în care încercăm să mutăm un subset de fişiere într-un director, iar acest subset trebuie să se potrivească unui pattern. De exemplu, dacă toate fişierele JPEG dintr-un folder trebuie să fie mutate într-un fişier numit *Imagini JPEG,* iar fişierele GIF şi PNG trebuie să ramână în fişierul iniţial, putem folosit comanda:[5]

```
$ mv *.{jpg,JPG} JPEG\ Images
```

Bibliografie

[1] Kevin O'Malley, "Programming Mac OS X: A guide for UNIX developers", Manning Greenwich, United States of America, 2003, ISBN 1-930110-85-5
[2] Dave Taylor, "Learning Unix for OS X Mountain Lion", O'Reilly Media Inc., United States of America, October 2012, ISBN: 978-1-449-33231-0
[3] Ben Simonds, "Master your Mac", Blender Master Class, USA, 2013, ISBN: 1-59327-477-7
[4] David Pogue, "OS X Mountain Lion: The Missing Manual", O'Reilly Media, Inc., July 2012, ISBN: 978-1449-33027-9
[5] Chris Seibold, "OS X Mountain Lion Pocket Guide", O'Reilly Media Inc., July 2013, ISBN: 978-1-449-33032-3
[6] Ole Henry Halvorsen, Douglas Clarke, "OS X and iOS Kernel Programming", Apress
[7] Galen Gruman, "OS X Mountain Lion Bible", John Wiley & Sons, Inc., ISBN: 978-1-118-4014309