Mac OS X Internals

ottobre 2002

Cobelli Andrea

cobellia@tin.it

Tipologia: ricerca bibliografica

1. Introduzione

Nel corso dell'anno 1999 Apple ha presentato il suo nuovo sistema operativo denominato Mac OS X (dieci), risultato dalla saggia combinazione di nove elementi di base.

Il famoso sistema operativo Macintosh, da sempre con interfaccia grafica a finestre, possiede ora un kernel Unix che lo dovrebbe rendere ancora più affidabile introducendo, tra le altre cose, il multitasking prelazionale e una nuova gestione della memoria fisica e virtuale.

[1][2]L'implementazione di Mac OS X di Apple si appoggia su DARWIN, il kernel open source che ingloba un certo numero di moderne tecnologie (di cui sono resi disponibili i sorgenti) e il sottosistema BSD.

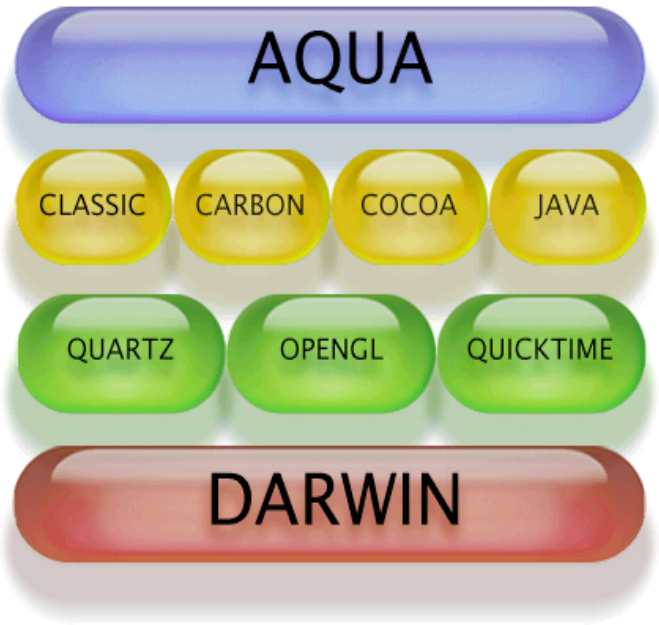


figura 1 - Livelli di Mac OS X

1.1. Darwin

L'evoluzione che inserisce un motore Unix all'interno del Mac OS inizia dall'acquisizione di NeXT (di Steve Jobs) da parte di Apple nel 1996, con l'obiettivo di unire la facilità d'uso di Macintosh alla stabilità Unix.

[2] Apple è da considerarsi la prima, tra le grandi aziende, a sviluppare e a mettere a disposizione, secondo una *public source license*[1] , le parti del proprio sistema operativo.

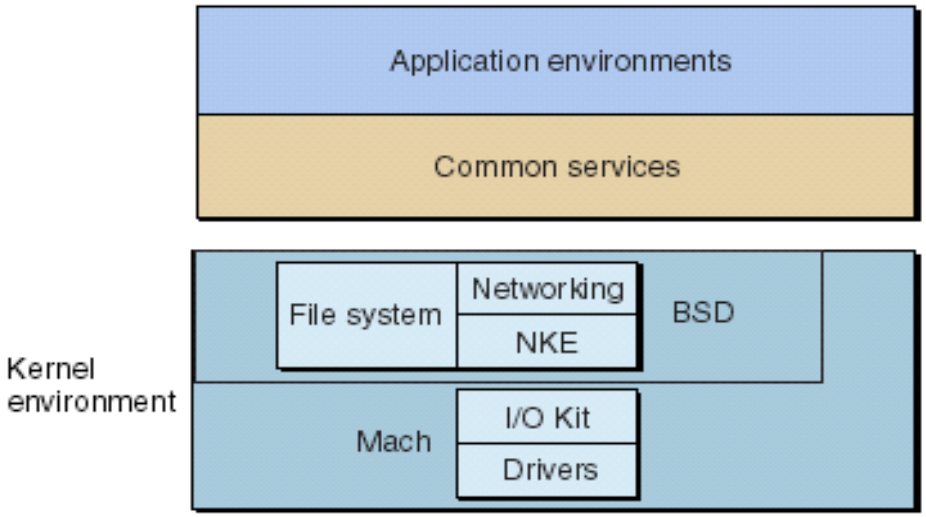


figura 2 - Livelli di Darwin

[1] APPLE PUBLIC SOURCE LICENSE, da accettare per poter scaricare i sorgenti di Darwin, riporta le condizioni per cui si può usare, copiare, modificare e distribuire liberamente il codice originale e/o modificato.

[2] Secondo quanto dichiarato da Apple la stabilità di Mac OS X inizia con il progetto open source Darwin, che include il *micro-kernel*[2] Mach 3.0 e i servizi del BSD Unix 4.4 ([14]FreeBSD), introducendo NKE (Network kernel Extension) per rendere più facile lo sviluppo di moduli per la gestione di protocolli; VFS (Virtual File System) facilitando la gestione di molti degli standard attuali per l'accesso alle periferiche; J2SE (Java 2 Standard edition) includendo l'accesso all'interfaccia grafica e rendendo nativo il supporto al multitasking prelazionato.

Darwin permette l'astrazione degli oggetti "memoria" facilitando la gestione di quella virtuale, aggiunge meccanismi avanzati per la protezione della memoria con una robusta architettura che alloca uno spazio di indirizzi unico per ogni processo

Inoltre Darwin, attraverso il micro-kernel Mach, contiene le funzionalità per: multitasking cooperativo[3] e prelazionato; multiprocesso *simmetrico*[4] ; supporto del real-time con garanzia di bassa latenza d'accesso alle risorse del processore; un framework di programmazione orientato agli oggetti per lo sviluppo di driver per le periferiche.

[2] Come sarà descritto di seguito, questo elemento implementa le chiamate di sistema essenziali, rendendo disponibile le funzioni accessorie nello spazio utente allegerendone il funzionamento.

[3] [4] La condivisione di informazioni, l'accelerazione del calcolo, la modularità e la convenienza sono elementi che spingono i task a collaborare. Mach prevede la cessione della CPU da parte di un task per favorirne un altro.

[4] [4] Ogni processore possiede una propria versione del sistema operativo facilitando l'intercambiabilità nei confronti dei tasks.

La tabella seguente mette in relazione le versioni rilasciate di Darwin con il prodotto commerciale Mac OS X di Apple Computer, Inc. [17]

tabella 1 - Versioni Darwin e Mac OS X a confronto

Version	Released	Corresponds With
Darwin 0.1	Marzo 1999	Mac OS X Server 1.0
Darwin 0.2	Estate 1999	Mac OS X Server 1.0
Darwin 0.3 (CD)	Agosto 1999	Mac OS X Server 1.0 Update 2
Darwin 1.0	Aprile 2000	Mac OS X Developer Preview 4
Darwin 1.2	Ottobre 2000	Mac OS X Public Beta
Darwin 1.3.1	Aprile 2001	Mac OS X 10.0.1
Darwin 1.4.1	Autunno 2001	Mac OS X 10.1

Spesso, ma non sempre, ad un nuovo rilascio del kernel Darwin corrisponde una nuova release di Mac OS X. Restano comunque implementazioni diverse che coinvolgono gruppi di sviluppo separati e che eseguono fasi di test indipendenti.

1.2. BSD Unix

[9] Mach è stato utilizzato come base sulla quale Unix e altri sistemi possono essere emulati. Questa emulazione viene fatta da appositi moduli software che girano nello spazio utente e per ciò sono stati resi intercambiabili.

Il principale server è quello che emula il sistema Unix BSD, sviluppato nell'Università di Berkeley in California a partire dagli anni '70. Lo Unix BSD ha origine [1] da una licenza d'uso rilasciata da AT&T di cui si forniva anche il codice sorgente, e dopo una serie di dispute legali con AT&T si arrivò alla stesura definitiva del FreeBSD 2.0 che permise l'utilizzo secondo le moderne licenze open source.

[14] Nell'implementazione di Mac OS X, il livello BSD è basato sul 4.4BSD-Lite2 del Computer Systems Research Group (CSRG) dell'Università di Berkeley. L'emulazione è composta da due parti principali, il server e la libreria di emulazione delle chiamate di sistema.

Quando il sistema inizia a funzionare, il server UNIX chiede al kernel di intercettare tutte le trap di chiamate di sistema e di vettorizzarle in un indirizzo dentro la libreria di emulazione del processo UNIX che crea la chiamata di sistema. Da quel momento, ogni chiamata di sistema fatta da un processo UNIX si risolverà nel passaggio temporaneo del controllo al kernel e, immediatamente dopo, nel passaggio alla sua libreria di emulazione. Al momento in cui il controllo viene dato alla libreria di emulazione, tutti i registri macchina hanno i valori che avevano al momento del trap; questo metodo di far rimbalzare il kernel indietro nello spazio utente vieene qualche volta chiamato meccanismo di trampolino.

La richiesta di esecuzione di una system call, ad esempio una *fork*[5], genera dei processi figli che ereditano automaticamente sia la libreria di emulazione che i meccanismi di trampolino[6].

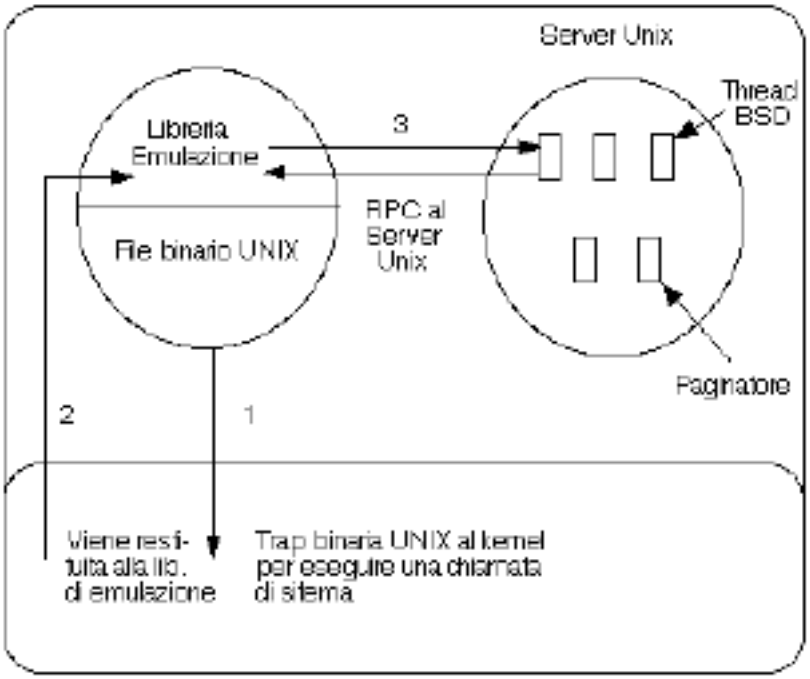


figura 3 - Emulazione di una system call

Il sottosistema BSD implementa molti servizi avanzati, tra cui: multitasking prelazionato con priorità dinamica; accesso multiutente; potente e robusto supporto di rete per il TCP/IP e gli standard SLIP, PPP e NFS; protezione della memoria; schedulazione indipendente per ogni thread[7]; possibilità di utilizzare le POSIX thread API per creare nuovi thread utente; supporto di CPU multiple.

[14] Di seguito saranno schematizzate le maggiori differenze dell'implementazione BSD interno a Mac OS X rispetto al progetto FreeBSD:

- La storica system call sbrk(), per la gestione della memoria, non è raccomandata in Mac OS X.

[5] La system call FORK chiamata da un processo (task in Mach) genera un processo figlio riconducibile al processo padre.
[6] Meccanismi che permettono il passaggio delle richieste da modalità utente a kernel e viceversa.
[7] Unità minima di esecuzione che deve essere attivato all'interno di un task.

- Mac OS X non supporta i dispositivi mappati in memoria (memory-mapped devices) attraverso l'utilizzo delle API mmap() (map files or devices into memory). Il sottosistema grafico mette a disposizione delle funzioni simili ma attraverso delle differenti API.
- La chiamata *swapon()* non è supportata; *macx_swapon()* è la funzione equivalente per il paginatore Mach.
- Mach prevede primitive IPC che differiscono da quelle tradizionalmente trovate in Unix (System V), che vengono emulate ma ne viene sconsigliato l'uso.
- Mac OS X gestisce i meccanismi per condividere le librerie in una modalità differente da quella prevista in BSD.
- Alcune modifiche ai criteri di sicurezza sono stae apportate per migliorare la configurazione di amministratori di sistema multipli e la gestione degli utenti.

Alcune nuove funzionalità sono state modificate appositamente nell'implementazione del BSD all'interno di Darwin, tra cui:

- miglioramento della cache lato file-system e delle operazioni di I/O "clusterizzato";
- miglioramento del supporto del file system Macitosh e UFS;
- sviluppo dell'estensione Apple per il file system ISO-9660;
- I/O per NFS multi-thread non sincronizzato;
- system call addizionali per il supporto delle proprietà estese del file system Mac OS;
- convenzione sui nomi per il *pathnames* per accedere alle estensioni del file system Mac OS.

1.3. Mach

[3] Mach, il fulcro di Mac OS X, è stato implementato dai ricercatori della CMU (Carnegie Mellon University) ed è un kernel orientato alla comunicazione tra le entità, supportando il calcolo distribuito e parallelo compatibilmente al BSD 4.4.

Mach è composto da due elementi principali:

- un ristretto ed espandibile kernel che gestisce lo scheduling, la memoria virtuale e le comunicazioni tra i processi; tale kernel esporta un numero limitato di astrazioni attraverso un'interfaccia integrata.
- un ambiente di supporto all'accesso distribuito ai file, la comunicazione trasparente tra processi e l'emulazione del BSD 4.4.

[4] Il micro-kernel Mach 3 trasferisce il codice BSD all'esterno, in tale modo il codice specifico Unix viene eseguito come server in modalità utente permettendo di rimpiazzare BSD con un altro sistema operativo e/o di eseguirne altri simultaneamente.

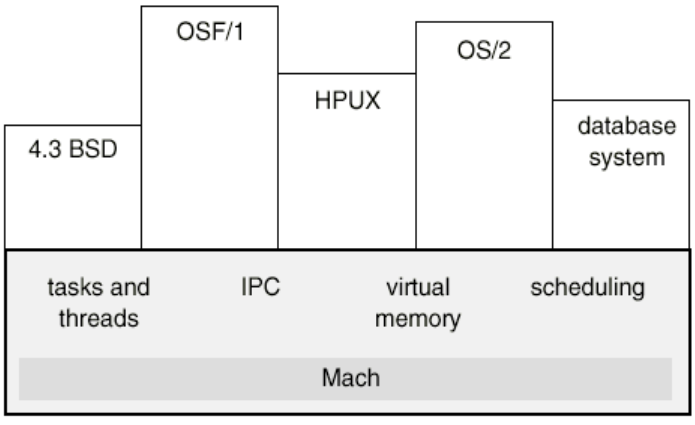


figura 4 - Server emulati nel micro-kernel Mach

Le system call di Mach sono compatibili verso l'alto con quelle del BSD e ne supportano tutti i comandi, tanto che molti dei programmi scritti per BSD sono riutilizzabili se ricompilati.

Il sistema Mach implementa inoltre le seguenti caratteristiche non incluse nel BSD 4.4:

- Task[8] multipli, ognuno con un grande spazio di memoria virtuale paginata.
- Flessibile condivisione di memoria tra i task.
- Esecuzione di thread multipli da parte dei task con gestione flessibile dello scheduling.
- Efficiente comunicazione tra processi basata su messaggi (IPC).
- Mappatura in memoria virtuale dei file fisici.[9]
- Approccio flessibile alla sicurezza e alla protezione.
- Supporto allo scheduling su multiprocessore.

[14] Comunque, in Mac OS X, il micro-kernel è agganciato con gli altri componenti del kernel all'interno di uno spazio degli indirizzi unico. Questa metodologia permette un decisivo miglioramento delle performance, infatti è molto più veloce eseguire una chiamata diretta tra componenti collegati (linked) che non inviare messaggi RPC tra task separati. Questa struttura modulare risulta essere più robusta ed estensibile di un micro-kernel compatto (monolitico) senza penalizzarne le performance.

[8] Ambiente di esecuzione che fornisce l'unità base per l'allocazione delle risorse.
[9] memory mapped file [3][11], in Mach tutto o una parte di un file su disco può essere caricato all'interno della memoria virtuale. Un eventuale page fault si traduce in una chiamata al file system dato che il riferimento tra memoria e file è lo stesso.

Il micro-kernel Mach sarà trattato con maggiore dettaglio nei prossimi capitoli, nei quali s'intende esplorare l'implementazione a basso livello degli algoritmi noti.

1.4. Processori PowerPC

[21] All'inizio del 1991 le tre compagnie Apple, IBM e Motorola decisero di unire i propri sforzi per sviluppare un'architettura comune. Si optò per l'utilizzo di una architettura preesistente come punto di partenza: l'architettura POWER (Performance Optimized with Enhanced Risc architecture), una architettura RISC[10] di seconda generazione sviluppata da IBM con registri dedicati a operazioni in virgola fissa (GPR) e registri dedicati a operazioni in virgola mobile (FPR) che gestiva, in un ottica di design Super scalare, separatamente le operazioni di controllo del Flow Program, di calcolo in virgola fissa e di calcolo in virgola mobile.

figura 5 - processore PowerPC G4

[10] Questi tipi di processori utilizzano un ridotto numero di istruzioni native rendendoli molto specializzati ed efficienti nell'esecuzione delle istruzioni.

Si è giunti alla produzione del processore MPC7400 conosciuto come G4, che da pochi mesi ha superato la frequenza di clock[11] di 1 GHz, ultima evoluzione del processore RISC della serie PowerPC.

Il processore G4 realizzato con tecnologia CMOS a 0.15 micron a 6 livelli di metallizzazione in rame, dispone di 10.5 milioni di transistor su un die di area 83 mm^2.

Il core è alimentato con 1.8 V e il consumo di potenza tipico è 5 W (massimo 11.5 W) rendendo interessante il G4 anche per applicazioni portatili.

In leggero contrasto alla politica RISC, fin qui adottata per i PowerPC, e a causa della crescente domanda di prestazioni in ambito audio, grafico e multimediale ha spinto Motorola a estendere il set di istruzioni con la tecnologia AltiVec[12] la cui implementazione è stata realizzata attraverso un'apposita unità di calcolo vettoriale a 128 bit chiamata AltiVec Unit che può agire indipendentemente dalle unità FP e interi portando fino a 16 le operazioni eseguibili in un clock.

[11] Spesso si confonde questo dato come velocità del processore. Le CPU che equipaggiano i Macintosh possiedono prestazioni maggiori alle famiglie Intel se si considera il numero di istruzioni processate per secondo (MIPS).

[12] Questa tecnica si avvale di 32 registri a 128 bit dedicati (in aggiunta a quelli per le operazioni interi e FP scalari) con il compito di eseguire operazioni vettoriali.

1.4.1. Gestione della Memoria del processore

Le due Memory Management Units per dati e istruzioni supportano indirizzi virtuali a 52 bit e indirizzi fisici a 32 bit (quindi fino a 4 GB (2^{32})). Il loro compito è tradurre gli indirizzi effettivi (o virtuali) calcolati nei relativi indirizzi fisici consentendo l'accesso alla memoria. Il G4 prevede un modello di memoria a 32 bit (pur essendo definibile nell'architettura PowerPC anche a 64 bit) e gestisce i 4 GB di indirizzi logici accessibili con pagine di dimensione fissa 4 KB e segmenti da 256 MB.

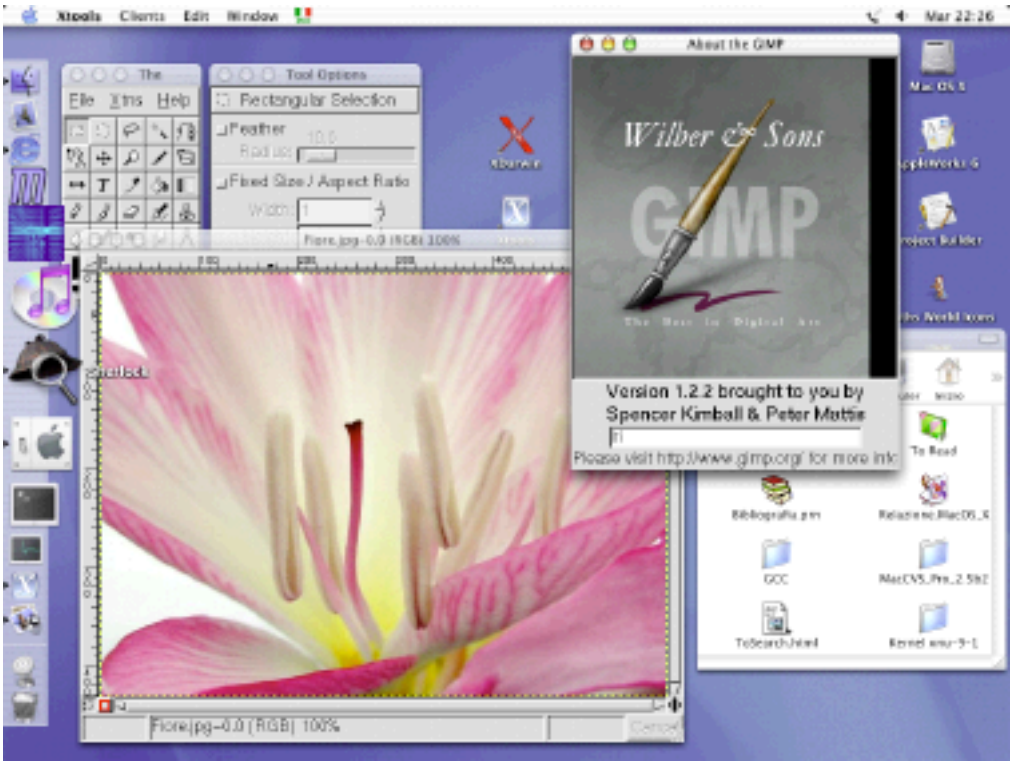
Per la traduzione ci si avvale, sia per dati che istruzioni, di un Translation Lookaside Buffer (TLB) a 128 ingressi, set-associative a 2 vie che salva gli indirizzi delle pagine recentemente trattate sfruttando un algoritmo LRU (least recently used): quando un cache miss provoca il riempimento di una linea, e tutte le linee dell'insieme contengono dati validi, la cache sostituisce la linea utilizzata meno recentemente.

1.5. Il sottosistema grafico

[15] Il cuore grafico è composto da una tecnologia Apple denominata Quartz utilizzata in Mac OS X per la grafica in 2D, OpenGL per il 3D e Apple QuickTime per la multimedialità.

X Windows non è stato incluso, anche se sono disponibili alcune porte di comunicazione, che ne consentono l'eventuale utilizzo. Apple ha formalmente incoraggiato gli sviluppatori all'uso della tecnologia Quartz come valore aggiunto ai sistemi Mac OS X.

[2] Mac OS X utilizza le tre potenti tecnologie grafiche, Quartz, OpenGL e QuickTime mettendo a disposizione primitive di sviluppo e primitive utente.



fiigura 6 - L'interfaccia grafica Aqua di Mac OS X

Quartz svolge il ruolo di window server e di rendering in 2D indipendentemente dal dispositivo utilizzato, basandosi su un modello bidimensionale indipendente dalla piattaforma (Portable Document Format - PDF).

[15] Quartz si può definire come la rappresentazione della terza generazione di sistemi grafici, dopo la scrittura diretta dei pixel e l'impiego di funzioni a basso livello.

Quartz utilizza vettori grafici e funzioni matematiche per la manipolazione degli elementi grafici, e aggiunge il *canale alfa* per l'effetto di trasparenza e gli effetti a scomparsa.

[2] *OpenGL* viene utilizzato da Apple per il rendering in tre dimensioni, attenendosi a uno standard che consente una grande portabilità del codice scritto.

QuickTime svolge il ruolo di potente motore multimediale per l'apertura, la modifica e la memorizzazione di file audio, video, animazioni, grafici. Il pacchetto è disponibile per un considerevole numero di piattaforme ed è in grado di gestire innumerevoli formati.

Tutto questo permette di rendere l'interfaccia grafica utente *Aqua* particolarmente semplice, piacevole e potente, secondo la tradizione Apple.

1.6. Sistema Macintosh

Mac OS X gestisce contemporaneamente ambienti diversi, tra cui *Classic* e *Carbon,* garantendo la compatibilità con applicazioni sviluppate per i sistemi precedenti, emulando una versione completa di Mac OS 9.1 oppure mettendo a disposizione delle API che favoriscono il funzionamento di programmi scritti per Mac OS 8 e successivi.

L'ambiente applicativo *Cocoa*, nativo nel sottosistema Mac OS X, facilita l'utilizzo di sistemi di sviluppo tipo RAD (Rapid Application Development) e le tecniche orientate agli oggetti.

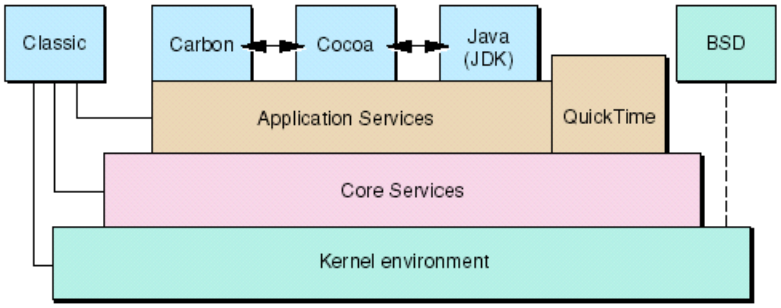


Figura 7 - Struttura degli ambienti applicativi di Mac OS X

2. Descrizione del problema

Nella stesura di questo documento s'intende trattare gli aspetti interni al kernel su cui è stato sviluppato il Mac OS X di Apple. Per quanto fin qui visto, lo studio riguarderà principalmente il micro-kernel Mach e il sotto-sistema BSD.

Si sono cercati riferimenti bibliografici per i sistemi adottati da Apple per i processi di scheduling della CPU e della gestione della memoria. Si sono inoltre analizzati i file sorgenti del micro-kernel trovando i riferimenti e le implementazioni degli algoritmi descritti.

2.1. Le scelte di progetto

[4] La filosofia di progetto Mach consiste nell'ottenere un kernel semplice ed estensibile, basato su meccanismi di comunicazione, garantendo un sistema di protezione efficiente.

Le astrazioni di Mach, di seguito riportate, rappresentano i fondamenti del sistema stesso:

- Un *task* viene implementato come un ambiente di esecuzione che fornisce l'unità di base per l'allocazione delle risorse; è costituito da uno spazio di indirizzi virtuali e un accesso protetto delle risorse attraverso porte, e può contenere più *thread* cioè l'unità base di esecuzione.
- Una *porta* è realizzata come un canale di comunicazione protetta dal kernel, sulla quale vengono scambiati messaggi di comunicazione tra i *thread.*

- Un *insieme di porte* raggruppa le porte che condividono una coda di messaggi comuni.
- Un *messaggio* rappresenta il metodo di base per la comunicazione tra *thread* ed è definito come un insieme tipizzato di dati.
- Un *oggetto di memoria* è una sorgente di memoria di cui i *task* possono associare porzioni al proprio spazio d'indirizzi.

Questa predisposizione, in Mach, per la comunicazione ne facilita la programmazione distribuita e parallela, fondendo all'interno del kernel, la combinazione di caratteristiche di memoria e comunicazione tra processi.

2.2. Gestione della memoria

[4] Mach collega la gestione della memoria e comunicazione IPC consentendo l'utilizzo dell'una per la realizzazione dell'altra. La gestione della memoria è basata sull'uso degli oggetti di memoria . Un oggetto di memoria è rappresentato da una porta (o da un gruppo) e i messaggi IPC vengono inviati a questa porta per richiedere operazioni (ad esempio page-in, page-out) sull'oggetto. Poiché viene fatto ricorso all'utilizzo di IPC, gli oggetti di memoria possono risiedere su sistemi remoti, e l'accesso a tali oggetti, può avvenire in modo trasparente.

[9] L'aspetto della gestione della memoria, che lo rende diverso dagli altri sistemi, si trova nel codice diviso in quattro sotto gruppi [11]: la *tabella delle pagine residente,* che tiene traccia delle informazioni delle pagine indipendenti dalla macchina, *pmap* che gestisce la MMU (dipende dall'hardware su cui gira), *codice kernel* interessato all'elaborazione dei page fault (gestione degli indirizzi e rimpiazzamento delle pagine) e *gestore della memoria esterna* che tratta la parte logica della memoria "concessa" ai task.

[11] La dimensione della pagina di memoria in Mach è un parametro di sistema impostato durante le fasi di boot, esso è tipicamente una potenza di due rapportata alla dimensione fisica (es. 512, 1K, 2K...) in dipendenza all'hardware su cui è implementato.

Sui sistemi PowerPC (PPC), utilizzati nei computer Apple, le pagine di memoria hanno dimensione pari a 4 Kbytes.

```
.............................ppc/vm_param.h...........................

#ifndef     _MACH_PPC_VM_PARAM_H_
#define     _MACH_PPC_VM_PARAM_H_

#define BYTE_SIZE 8                         /* byte size in bits */

#define PPC_PGBYTES     4096                /* bytes per ppc page */
#define PPC_PGSHIFT     12    /* number of bits to shift for pages */

#define VM_MIN_ADDRESS  ((vm_offset_t) 0)
#define VM_MAX_ADDRESS  ((vm_offset_t) 0xfffff000U)

#define VM_MIN_KERNEL_ADDRESS ((vm_offset_t) 0x00001000)

/* We map the kernel using only SR0,SR1,SR2,SR3 leaving segments alone
*/
#define VM_MAX_KERNEL_ADDRESS ((vm_offset_t) 0x3fffffff)

#define USER_STACK_END  ((vm_offset_t) 0xffff0000U)

#define ppc_round_page(x)     ((((unsigned)(x)) + PPC_PGBYTES - 1) & \
                              ~(PPC_PGBYTES-1))
#define ppc_trunc_page(x)     (((unsigned)(x)) & ~(PPC_PGBYTES-1))


#define KERNEL_STACK_SIZE     (4 * PPC_PGBYTES)
#define INTSTACK_SIZE         (5 * PPC_PGBYTES)

#endif            /* _PPC_VM_PARAM_H_ */
.....................................................................
```

```
.............................osfmk/vm_param.h.........................
/*
 *    The machine independent pages are refered to as PAGES.  A page
 *    is some number of hardware pages, depending on the target
machine.
 */

/*
 *    All references to the size of a page should be done with
PAGE_SIZE
 *    or PAGE_SHIFT.  The fact they are variables is hidden here so
that
 *    we can easily make them constant if we so desire.
 */

/*
 *    Regardless whether it is implemented with a constant or a
variable,
 *    the PAGE_SIZE is assumed to be a power of two throughout the
 *    virtual memory system implementation.
 */

#ifndef     PAGE_SIZE_FIXED
extern vm_size_t        page_size;
extern vm_size_t        page_mask;
extern int              page_shift;

#define PAGE_SIZE       page_size         /* pagesize in addr units */
#define PAGE_SHIFT      page_shift        /* number of bits to shift
                                                    for pages */
#define PAGE_MASK       page_mask         /* mask for off in page */

#define PAGE_SIZE_64 (unsigned long long)page_size    /* pagesize in
                                                    addr units */
#define PAGE_MASK_64 (unsigned long long)page_mask    /* mask for off
                                                        in page */
#else                                     /* PAGE_SIZE_FIXED */
#define PAGE_SIZE       4096
#define PAGE_SHIFT      12
#define     PAGE_MASK   (PAGE_SIZE-1)
#define PAGE_SIZE_64    (unsigned long long)4096
#define PAGE_MASK_64    (PAGE_SIZE_64-1)
#endif                  /* PAGE_SIZE_FIXED */
.....................................................................
```

2.2.1. Gestione della memoria virtuale

[13] Si introduce la definizione di *spazio di indirizzi virtuali* che s'intende come un range di indirizzi di memoria validi a cui un thread può riferirsi.

Uno spazio di indirizzi virtuali consiste in un 'pacchetto' di pagine di memoria sparse e indicizzate.

[4] Con l'obiettivo di migliorare la gestione dei messaggi, fondamento del proprio funzionamento, Mach utilizza il *remapping* della memoria virtuale, con il quale trasferisce i messaggi di grandi dimensioni. Queste tecniche di copiatura virtuale, modificano lo spazio degli indirizzi del task ricevente e presentano i seguenti vantaggi:

- Flessibilità nella gestione della memoria dei processi utente;
- Approccio generale, cioè applicabile a computer sia strettamente che debolmente accoppiati;
- Miglioramento delle prestazioni rispetto allo scambio di messaggi Unix;
- Gestione dei task migliorata con possibilità di migrazione. I task comunicano tra loro tramite le proprie porte, uno spostamento non modifica queste proprietà.

[13] Particolarmente interessante e avanzata risulta essere la copia logica di un range di memoria, che permette un'efficiente implementazione della system call POSIX Fork per la creazione di un nuovo task. Nell'utilizzo della Fork infatti, si copiano intere porzioni di memoria tra task, ovvero, vengono passati in un messaggio Mach, parti della memoria che vengono semplicemente condivise supponendo la loro iniziale non modificabilità.

[4] La gestione a oggetti, principio base di Mach, si estende alla memoria, in particolare a quella secondaria, applicandosi alle rappresentazioni di file, pipe o altri dati che risiedono nella memoria virtuale per lettura e scrittura.

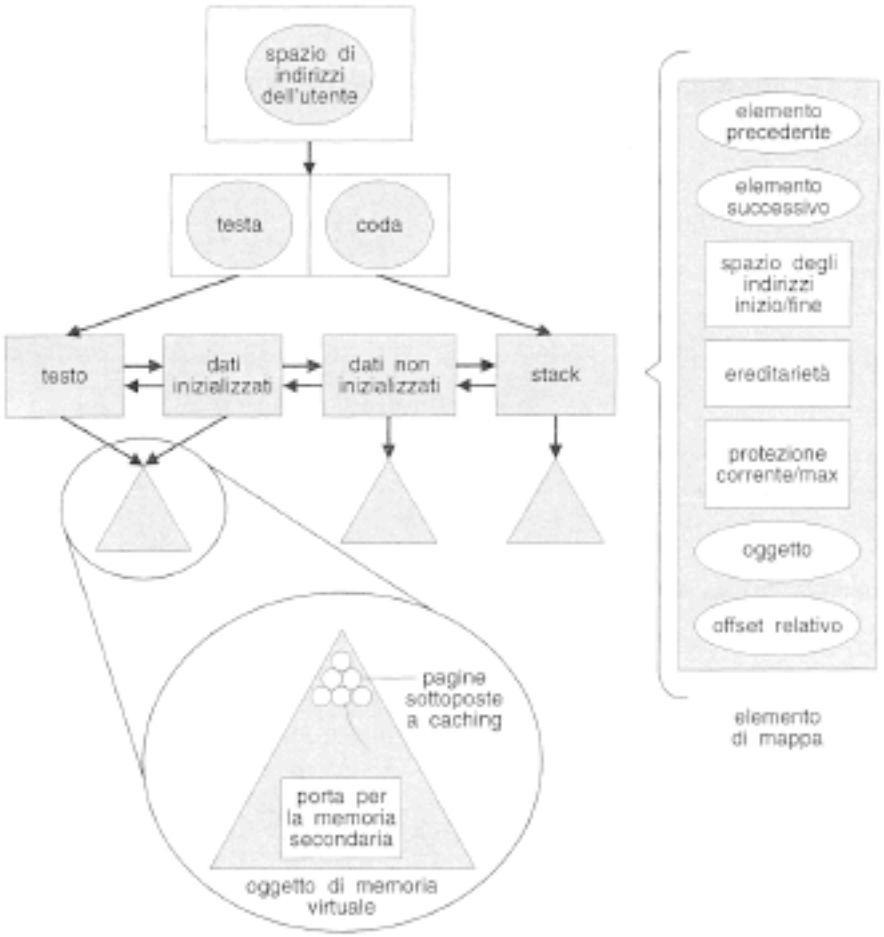


figura 8 - Gestione degli indirizzi degli oggetti di memoria

Ad ogni oggetto di memoria secondaria viene associata una porta di comunicazione e può quindi essere manipolato attraverso messaggi che sono inviati alla sua porta.

[9] Allo scopo di determinare quali indirizzi virtuali siano in uso e quali no, Mach fornisce un modo di allocare e deallocare sezioni dello spazio degli indirizzi virtuali, chiamati *regioni*.

[3][4][11] Dipendentemente dalla piattaforma Hardware su cui è installato, Mach può mettere a disposizione uno spazio degli indirizzi di 4 GB per ogni Task, che non vengono compattati, ciò richiederebbe una tabella delle pagine di dimensioni superiori a 1MB. Per gestire efficacemente gli spazi degli indirizzi sparsi, la tabella delle pagine risulta utilizzata solamente per le regioni correntemente allocate.

Al verificarsi di un *page fault* il kernel deve controllare se la pagina si trova in una regione valida. Questa procedura rende la ricerca più complessa ma semplifica la manutenzione e riduce lo spazio richiesto.

Mach conserva una cache delle pagine residenti in memoria di tutti gli oggetti mappati, al verificarsi di un page fault genera un messaggio alla porta dell'oggetto in memoria.

[9] Mach fornisce una verifica precisa e ben dettagliata delle pagine virtuali usate (per i processi che ne sono interessati), rendendo concreta la possibilità di supportare gli spazi degli indirizzi sparsi[13] .

In teoria, qualsiasi spazio degli indirizzi virtuale può essere usato, permettendo a un processo di avere una sezione di dati da 50 k distribuiti in 100 MB.

[13] Ogni thread può frammentare i propri dati all'interno della memoria a disposizione, indipendentemente dalla continuità della memoria libera. Anche se tale tecnica da un bassissimo coefficiente di ottimizzazione.

2.2.1.1. Algoritmo di scelta della pagina vittima

[3] Il micro-kernel Mach implementa la sostituzione delle pagine in memoria attraverso l'approssimazione dell'algoritmo ottimo denominata LRU (least recently used). Se non ci sono pagine di memoria fisica libera, e un task esegue un page fault, il kernel chiede al demone di paginazione [4] di copiare la pagina che non è usata per il periodo di tempo più lungo sulla memoria di massa.

```
.............................vm_pageout.c.............................
#ifndef VM_PAGEOUT_BURST_WAIT
#define VM_PAGEOUT_BURST_WAIT 30          /* milliseconds per page */
#endif                                    /* VM_PAGEOUT_BURST_WAIT */
#ifndef VM_PAGEOUT_EMPTY_WAIT
#define VM_PAGEOUT_EMPTY_WAIT 200         /* milliseconds */
#endif                                    /* VM_PAGEOUT_EMPTY_WAIT */

/*
 *    To obtain a reasonable LRU approximation, the inactive queue
 *    needs to be large enough to give pages on it a chance to be
 *    referenced a second time.  This macro defines the fraction
 *    of active+inactive pages that should be inactive.
 *    The pageout daemon uses it to update vm_page_inactive_target.
 *
 *    If vm_page_free_count falls below vm_page_free_target and
 *    vm_page_inactive_count is below vm_page_inactive_target,
 *    then the pageout daemon starts running.*/

#ifndef VM_PAGE_INACTIVE_TARGET
#define VM_PAGE_INACTIVE_TARGET(avail)    ((avail) * 1 / 3)
#endif                                /* VM_PAGE_INACTIVE_TARGET */
/*
 *    Once the pageout daemon starts running, it keeps going
 *    until vm_page_free_count meets or exceeds vm_page_free_target.*/
.....................................................................
```

[4]Mach 3.0, utilizzato nella prima versione di Mac OS X, adotta un proprio gestore di memoria di default che non può essere sostituito da un gestore utente ([14] a differenza delle altre implementazioni di questo micro-kernel in cui può esserne definito uno esterno).

[18] Di default il kernel Mach usa una politica di allocazione delle pagine di tipo globale, [4] ciò permette ad un task di selezionare un frame per la sostituzione dall'insieme di tutti i frame (anche se di un altro task).

[4] La politica di rimpiazzamento delle pagine viene realizzato da un task interno al kernel, *paeout deamon*, che utilizza un algoritmo di paginazione di tipo FIFO con seconda chance, che è l'approssimazione dell'algoritmo ottimo LRU.

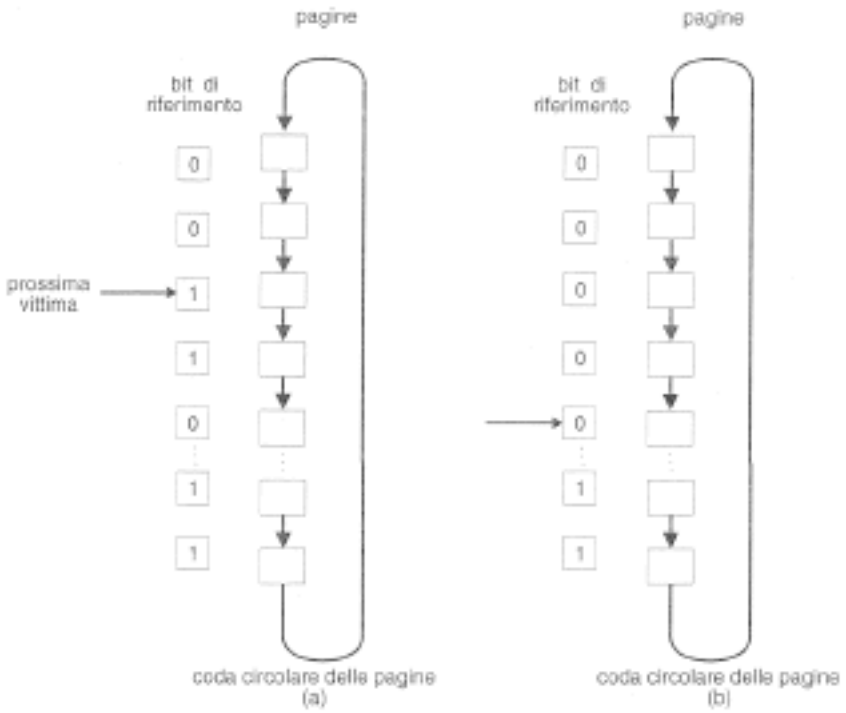


figura 9 - Algoritmo con seconda chance

Quando, per la prima volta, una pagina viene selezionata per la rimozione, ne viene solo modificato il *bit di riferimento*[14] che viene impostato a uno. Solo alla successiva selezione per la rimozione, se non è più stata utilizzata dal task, viene effettivamente rimossa. I bit di riferimento sono associati agli elementi della tabella delle pagine.

[14] Appositamente inserito per registrare gli accessi ai frame in memoria.

[19] Il kernel Mach 3.0 utilizza un meccanismo per il *bit reference* indipendente dalla piattaforma, scavalcando il limite che si avrebbe se l'hardware non lo supportasse. Questo meccanismo, pageout deamon, utilizza l'interfaccia pmap() per manipolare il *page reference* e per interagire con le architetture che non lo contemplano.

Il pageout demon, che gira all'interno dello spazio degli indirizzi del kernel, sposta le pagine dalla coda delle inattive a quella delle pagine libere e viceversa. Queste operazioni vengono sempre verificate per permettere la memorizzazione su disco delle pagine modificate che stanno per lasciare la memoria fisica.

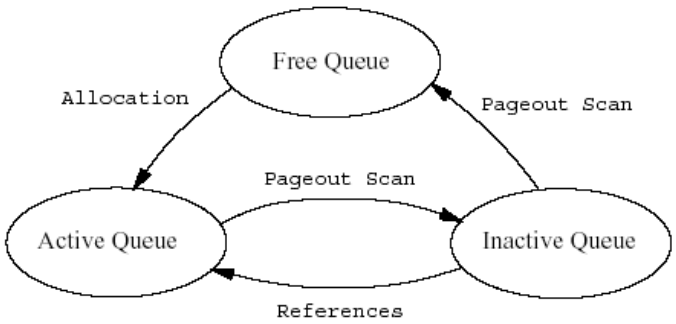


figura 10 - Demone di paginazione: gestione delle pagine

Una pagina inserita precedentemente nella coda inattiva può essere riportata nella coda attiva da un task che ne esegue il *reference*.

Per assicurare che siano sempre presenti un certo numero di pagine libere, il demone di paginazione viene eseguito ad intervalli di tempo predefiniti e indipendentemente dalle richieste eseguite dai tasks. Questo permette di avere sicuramente spazio libero e di non doverne liberare frettolosamente nel caso in cui alcuni task ne richiedessero. Le pagine che vengono liberate dal demone di paginazione, e non immediatamente necessarie, vengono solamente segnate come libere e non cancellate.

Questa procedura permette un immediato recupero della pagina nel caso sia da utilizzare prima della sua effettiva sovrascrittura.

[13] Quando un oggetto di memoria non viene utilizzato da molto tempo, il kernel normalmente libera tutte le pagine di memoria a esso riservate. Il gestore della memoria segna le pagine dell'oggetto come *rimovibile*, quindi le pagine così contraddistinte potranno cedere posto nel caso di page fault.

2.2.1.2. Ottimizzazione nella sostituzione delle pagine

Al fine di ottimizzare le operazioni che riguardano la sostituzione della pagine tra memoria fisica e virtuale, Mach prevede la gestione software del bit di modifica della pagina (*dirty bit).*

```
.............................vm_page.h.................................
/*    Each pageable resident page falls into one of three lists:
 *
 *    free
 *          Available for allocation now.
 *    inactive
 *          Not referenced in any map, but still has an
 *          object/offset-page mapping, and may be dirty.
 *          This is the list of pages that should be
 *          paged out next.
 *    active
 *          A list of pages which have been placed in
 *          at least one physical map.  This list is
 *          ordered, in LRU-like fashion.
 */
extern vm_page_t  vm_page_queue_free;     /* memory free queue */
extern vm_page_t  vm_page_queue_fictitious;     /*    fictitious    free
queue */
extern queue_head_t     vm_page_queue_active;   /* active memory queue
*/
extern queue_head_t     vm_page_queue_inactive; /*    inactive    memory
queue */
.....................................................................
```

```
.............................vm_map.c.................................
      if ((p = vm_page_lookup(object, offset)) != VM_PAGE_NULL) {
            if (shadow && (max_refcnt == 1))
                  extended->pages_shared_now_private++;
            if (p->dirty || pmap_is_modified(p->phys_addr))
                  extended->pages_dirtied++;
            extended->pages_resident++;
            if(object != caller_object)
                  vm_object_unlock(object);

.............................vm_kernel.c..............................
      if ((mem = vm_page_lookup(copy->cpy_object,
                  (vm_object_offset_t)offset)) == VM_PAGE_NULL)
          panic("kmem_io_object_trunc: unable to find object page");
            /*
             * Make sure these pages are marked dirty
             */
      mem->dirty = TRUE;
      vm_page_lock_queues();
      vm_page_unwire(mem);
      vm_page_unlock_queues();

.............................vm_fault.c...............................
      if (result_page == VM_PAGE_NULL)
            vm_page_zero_fill(dst_page);
      else{
            vm_page_copy(result_page, dst_page);
            if(!dst_page->dirty){
                  vm_object_lock(dst_object);
                  dst_page->dirty = TRUE;
                  vm_object_unlock(dst_page->object);
            }
      }
.....................................................................
```

Utilizzando questo meccanismo si possono dimezzare le operazioni di scrittura e di lettura della pagine nel caso in qui queste non siano state modificate. Risulta cioè inutile sovrascrivere una pagina di memoria che non sia stata precedentemente modificata.

Tutte le operazioni eseguite sugli oggetti di memoria vengono protette da un semaforo mutex[15]. Se più tasks accedessero contemporaneamente a un'area di memoria, non si potrebbe più garantire la necessaria integrità e congruenza dei dati; si inserisce perciò [20] la sezione critica all'interno di operazioni di lock e unlock implementate attraverso l'uso di semafori binari.

[15] Semaforo software che garantisce la mutua esclusione.

Il sistema garantisce che le operazioni di blocco delle variabili dei semafori siano eseguite come operazioni *atomiche*[16], che non permetto a due o più tasks di accedere alla propria sezione critica protetta dallo stesso mutex.

```
.............................vm_object.h.............................
      #define vm_object_lock(object)  mutex_lock(&(object)->Lock)
      #define vm_object_unlock(object) mutex_unlock(&(object)->Lock)

.....................................................................
```

Le funzioni di mutex, le cui primitive sono contenute nel file *hw_lock.s,* sono implementate direttamente in assembler per il processore PowerPC.

Le richieste di allocazione delle pagine virtuali passano attraverso un'ulteriore ottimizzazione.

Un task esegue una scansione delle pagine di memoria fisica libera e le ordina in base alla dimensione. Alla richiesta di pagine di memoria virtuale da parte del kernel, viene passato un riferimento a una lista di memoria fisica continua. Se tale spazio di memoria continua è sufficientemente ampia, viene allocato facilitando il lavoro del processo di working set di seguito descritto.

```
...........................vm_resident.c.............................
/*    Sort free list by ascending physical address,
 *    using a not-particularly-bright sort algorithm.
 *    Caller holds vm_page_queue_free_lock.*/
static void vm_page_free_list_sort(void)

/*    Check that the list of pages is ordered by
 *    ascending physical address and has no holes. */
int   vm_page_verify_contiguous( vm_page_t pages, unsigned int npages)

.....................................................................
```

[16] Durante queste fasi la CPU non sarà concessa a nessun altro task.

2.2.1.3. Contenimento del thrashing[17]

Una scarsa quantità di frame a disposizione dei task potrebbe portare ad un continuo lavoro di *page in*[18] e *page out* da parte del sistema che ne sarebbe pesantemente rallentato.

[14] All'interno del kernel è stato sviluppato il sottosistema addizionale per il rilevamento e la gestione del working set. Questo meccanismo, che introduce il principio di località, permette di evitare l'eventuale e deleterio processo di Thrashing che si può instaurare se i frame a disposizione dei task sono in numero insufficiente, causando un continuo page fault dei task coinvolti.

[14] Appena prima di una richiesta di una pagina, il codice che genera il fault richiede al sottosistema di working set quale pagina adiacente deve essere portata in memoria e, quindi, esegue una unica grande richiesta al sistema di paginazione.

```
.............................vm_fault.c.............................
            /*
             * Decide whether to scan ahead or behind for
             * additional pages contiguous to the faulted
             * page in the same paging block.  The decision
             * is based on system wide globals and the
             * expected page reference behavior of the
             * address range contained the faulting address.
             * First calculate some constants.
             */
            paging_offset = offset + object->paging_offset;
            cluster_offset = paging_offset & (cluster_size - 1);
            align_offset = paging_offset&(PAGE_SIZE_64-1);
            if (align_offset != 0) {
                  cluster_offset = trunc_page_64(cluster_offset);
            }
.....................................................................
```

[17] [4] Il concatenarsi di continui page fault rende il sistema inefficiente e la CPU inattiva in attesa di continue operazioni di I/O da parte dei task che si trovano con poche risorse per proseguire il lavoro.

[18] Operazione che acquisisce informazioni (dati e codice eseguibile) dal disco per portarli in memoria.

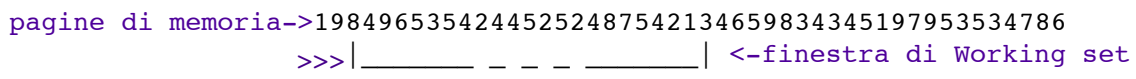
Poiché i file memorizzati nei dischi tendono ad avere un buon coefficiente di 'localizzazione' (cioè le tracce sono fisicamente vicine), e poiché lo spazio degli indirizzi locale è largamente rispettato sui supporti di memorizzazione di massa, questo genera un sostanziale miglioramento delle prestazioni complessive.

Dato che il sottosistema è basato sul precedente comportamento delle applicazioni, esso tende a caricare le pagine che probabilmente sarebbero necessarie successivamente.

Per poter usufruire del miglioramento portato dall'utilizzo del working set anche durante le fasi iniziali (di avvio) delle applicazioni, il sottosistema memorizza in un file (al primo avvio dell'applicazione) il working set utilizzato per poter essere successivamente caricato.

Questo file di working set d'avvio, è memorizzato nel file system e non è accessibile se non da root e dal kernel. Per Mac OS X 10.2 tale file è memorizzato nella posizione */var/vm/app_profile*.

La finestra di working-set viene avviata con una dimensione otto volte inferiore a quanto sarà la sua grandezza a regime. Un task[19] apposito, utilizzato per la gestione del working set, esegue una prima inizializzazione della finestra che poi viene allargata dinamicamente all'occorrenza dal gestore della memoria virtuale.



[19] TWS task working set.

```
.............................task_working_set.h.....................
           #define TWS_ARRAY_SIZE  8
           #define TWS_HASH_LINE_COUNT   32
           #define TWS_SMALL_HASH_LINE_COUNT   4

.............................task_working_set.c.....................
           task_working_set_create( task_t task, unsigned int lines,
                                unsigned int rows, unsigned int style){
           .............
           }
           task_expand_working_set( vm_offset_t tws, int line_count){
           .............
           }

.............................task.c................................
           task_working_set_create(new-task,
            TWS_SMALL_HASH_LINE_COUNT, 0, TWS_HASH_STYLE_DEFAULT);

.............................vm_fault.c............................
           task_expand_working_set( task->dynamic_working_set,
                                                 TWS_HASH_LINE_COUNT);

.....................................................................
```

2.3. Scheduler della CPU

[4] Il sistema di schedulazione della CPU sviluppato per il micro-kernel Mach si basa su un algoritmo *Round Robin* con code e livelli di priorità.

Data la sua natura multiprocessore, Mach predispone code locali (per un processore) e code di esecuzione globali.

Si cerca di seguito di schematizzare le caratteristiche principali del processo di scheduling:

- code *locali*, sono associate ai singoli processori e vengono assegnate ai processi, spesso kernel, che sono vincolati ad un certo processore;
- code *globali*, (sono 32) in esse viene utilizzata la priorità per distribuire i thread da eseguire. Le code globali hanno minore priorità rispetto a quelle locali;
- *priorità*, è associata ad ogni thread in esecuzione e il suo valore può essere compreso tra 0 e 127 ([12] nelle versioni più recenti si tende a far corrispondere il numero di code con la priorità, riducendo quest'ultima a un valore da 0 a 31). Il valore viene determinato attraverso la media esponenziale del tempo di CPU precedentemente usato, abbassando la priorità dei thread con precedenti tempi di esecuzione lunghi[20] ;

[20] [9] Ad ogni colpo di clock, la CPU incrementa il contatore di priorità del thread attualmete in esecuzione determinandone una discesa nella coda delle priorità che sarà abbassata.

- *quanto*[21] di tempo, è di tipo dinamico per migliorarne le prestazioni. Per non sprecare il tempo di *context switch*[22] , dato che i thread in esecuzione potrebbero essere in numero inferiore alle CPU a disposizione, la durata di un quanto di tempo è inversamente proporzionale al numero di thread in attesa di essere eseguiti. Il quanto di tempo calcolato è lo stesso per tutto il sistema;
- il *dispatcher*[23] globale, non esiste permettendo ai processori di consultare autonomamente le code di esecuzione locali e globali. Per accelerare la distribuzione dei thread nelle code, il Mach mantiene una lista dei processori inattivi.

Sulla falsa riga dello scheduler del BSD, la priorità dei thread nel sottosistema Mach viene ricalcolata continuamente ed è in [20] funzione dei cicli di CPU trascorsi e dal valore di Nice P(C, N)[24/A] .

Essendo C una funzione del tempo, si avrà uno spostamento verso il basso della priorità del task in esecuzione (dato che C aumenta).

[21] Unità di tempo in cui la CPU processa attivamente il thread.
[22] Tempo utilizzato per portare un'unità di esecuzione dalla fase di attesa a quella attiva e viceversa.
[23] Attività del kernel che assegna i thread alle CPU.
[24/A] Dove P è la priorità, C è il contatore dei cicli di clock [C(t)] del task *running* , N è il valore impostabile dall'utente.

```
.................................sched.c.............................
#define NRQS            128                     /* 128 run queues per
cpu */
#define NRQBM           (NRQS / 32)             /* number of run queue
bit maps */

#define MAXPRI          (NRQS-1)
#define MINPRI          IDLEPRI                 /*     lowest     legal
priority schedulable */
#define IDLEPRI         0                       /*      idle      thread
priority */
#define DEPRESSPRI      MINPRI                  /* depress priority */
/*
 *    High-level priority assignments
 *
*************************************************************************
 * 127            Reserved (real-time)
 *                (32 levels)
 * 96       Reserved (real-time)
 * 95       Kernel mode only
 *                (16 levels)
 * 80       Kernel mode only
 * 79       System high priority
 *                (16 levels)
 * 64       System high priority
 * 63       Elevated priorities
 *                (12 levels)
 * 52       Elevated priorities
 * 51       Elevated priorities (incl. BSD +nice)
 *                (20 levels)
 * 32       Elevated priorities (incl. BSD +nice)
 * 31       Default (default base for threads)
 * 30       Lowered priorities (incl. BSD -nice)
 *                (20 levels)
 * 11       Lowered priorities (incl. BSD -nice)
 * 10       Lowered priorities (aged pri's)
 *                (11 levels)
 * 0        Lowered priorities (aged pri's / idle)
 ************************************************************************/

#define BASEPRI_REALTIME      (MAXPRI - (NRQS / 4) + 1)     /* 96 */
#define MAXPRI_STANDARD       (BASEPRI_REALTIME - 1)        /* 95 */
#define MAXPRI_KERNEL         MAXPRI_STANDARD               /* 95 */

#define BASEPRI_PREEMPT       (MAXPRI_KERNEL - 2)           /* 93 */
#define MINPRI_KERNEL         (MAXPRI_KERNEL - (NRQS / 8) + 1)/* 80 */

#define MAXPRI_SYSTEM         (MINPRI_KERNEL - 1)           /* 79 */
#define MINPRI_SYSTEM         (MAXPRI_SYSTEM - (NRQS / 8) + 1)/* 64 */

#define MAXPRI_USER           (MINPRI_SYSTEM - 1)           /* 63 */
#define BASEPRI_DEFAULT       (MAXPRI_USER - (NRQS / 4))    /* 31 */
#define MINPRI_USER           MINPRI                        /*  0 */

#define MINPRI_STANDARD       MINPRI_USER                   /*  0 */
.....................................................................
```

All'interno del range di priorità possibile, ne vengono gestiti quattro tipi che possono essere assegnati ad un thread: la priorità base, quella massima, la priorità assegnata per i thread del kernel e per il *real-time*[25] .

[9] Ogni coda di esecuzione comprende tre variabili: un semaforo *mutex*[26] , un contatore e una variabile di supporto alla scelta.

Il mutex viene usato per bloccare la struttura dati e per garantire che solo una CPU alla volta stia gestendo la coda; il contatore mantiene il numero di thread sulle code, se tale numero è pari a zero non ci sono task da eseguire e viene lanciato un apposito thread d'attesa; la terza variabile viene utilizzata come suggerimento su dove trovare il thread a priorità massima migliorando la ricerca alle code non vuote.

Mach fornisce delle ulteriori proprietà interessanti nella gestione della schedulazione dei thread come la possibilità di abbassare la propria priorità e di poter concatenare unità di esecuzione correlate.

La combinazione delle due permette ad un thread di nominare il proprio successore, eliminando le code di messaggi tra task (mittente chiede di schedulare il destinatario).

Una opzione, che spesso viene disabilitata, permette di richiedere l'esecuzione di un thread sulla stessa CPU in cui ha girato l'ultima volta, facilitando il recupero degli indirizzi che potrebbero essere ancora nella cache.

[12] Per garantire che non vi siano unità di esecuzione in attesa da molto tempo, *starvation*[27] , Mach ha un thread interno al kernel che ogni due secondi scansiona le code di esecuzione aumentando di priorità eventuali thread in questa situazione.

[25] Quest'ultima priorità, la più elevata, permette ad un thread di richiedere un numero definito di cicli di CPU per essere eseguito ininterrottamente.
[26] Tipo di semaforo binario per l'accesso esclusivo di una risorsa.
[27] Intesa come 'fame' di cicli di CPU di task che, avendo priorità bassa, non vengono mai eseguiti.

4. Conclusioni

In questo documento si sono ricercate le parti esaminate nel corso di Sistemi Operativi, inoltrandosi negli aspetti “teorici” del kernel Macintosh. La natura Open Source di Darwin ha facilitato l’analisi dei meccanismi interni, anche se la conoscenza necessaria per capirne i dettagli è stato il vero ostacolo affrontato.

4.1. Considerazioni personali

Il nuovo sistema operativo rilasciato da Apple, arrivato alla versione 10.2, appare come la mossa vincente per poter essere nuovamente un componente attivo nel mercato mondiale dei sistemi operativi. La grande stabilità che giornalmente si percepisce e la sua facilità d’uso, concretizzano, da un lato, la volontà di essere professionali e dall’altro quella di essere alla portata di tutti.

Certo le applicazioni si cominciano a trovare, anche se con maggiore difficoltà rispetto ad altre piattaforme, ma Apple ha imboccato la strada giusta.

Come riportato in precedenza, molto di ciò che e stato scritto per BSD può essere riutilizzato se ricompilato. Il porting di molti software è già un progetto reale che è stato spinto dalla stessa Apple

Aprendo una shell del terminale si possono testare i comandi per bash e tcsh, si possono avviare applicazioni per XWindows come Gimp e AbiWord e si possono progettare applicazioni Java.

L’interfaccia grafica, sobria e piacevole, porta con se un numero notevole di astrazioni che ne permettono l’avvicinamento anche da parte degli utenti meno esperti.

BIBLIOGRAFIA

[1] Inside MACOS X, System Overview - 2001 Apple Computer, Inc.

[2] An Overview for Developers - 2001 Apple Computer, Inc.

[3] Rhapsody Operating System Software - 1997 Apple Computer, Inc.

[4] Sistemi Operativi - Abraham Silberschatz & Peter Baer Galvin ed. Addison - Wesley

[5] Operating Systems - Stallings

[6] Unix Internals(the new frontiers) - Uresh Vahalia ed. Prentice Hall International

[7] Introduzione all'architettura di sistema UNIX - Prabhat K. Andeigh - ed. Jackson

[8] C manuale di Programmazione - Peter A. Darnell & Philip E. Margolins ed. Mc Graw Hill

[9] I Moderni Sistemi Operativi - Andrew S. Tanenbaum ed. Prentice Hall international

[10] Applied Operating System Concepts - A. Silbershatz & P. Galvin ed John Wiley & Sons

[11] Machine-Indipendent Virtual memory management for Paged Uniprocessor and Multiprocessor Architectures - R. Rashid & A. Tevanian & M. Young & D. Golub & R. Baron & D. Black & W. Bolosky & J Chew - Carnegie Mellon University

[12] Scheduling Support for Concurrency and Parallelism in the Mach Operating System - David L. Black - Carnegie Mellon University

[13] Mach 3 Kernel Principles - Keith Loepere - 1991 Open Software Fundation & Carnegie Mellon University

[14] Kernel programming - 2002 Apple computer, Inc.

[15] http://homepage.mac.com/gaggan/mosx.html -

[16] http://www.cs.nmsu.edu/~lking/index.html

[17] www.opensource.apple.com/news/qa20010925.html

[18] Managing Discardable Pages with an External Pager - I. Subramanian - Carnegie Mellon University

[19] Page Replacement and Referece Bit Emulation in Mach - R. P. Draves - Carnegie Mellon University

[20] Appunti di sistemi operativi - 2002 Massimo Poncino - Università di Verona

[21] Il processore Motorola G4 - S. Stocchi - Università di Parma

[22] Ics game - P.Cristofoli & D. Tortora - Applicando n.192 anno 2002

[23] Sempre più irranggiungibile - L. Bassini - Applicando n.189 anno 2001

[24] Sotto il vestito niente.... - F. Cervelli - Applicando n.189 anno 2001

[25] Un motore da F1 - F. Cervelli - Applicando n.185 anno 2001

[26] The Impact of Operating System Structure on Memory System Performance - J.B. Chen - Carnegie Mellon University & B.N. Bershad - Universiy of Washington

[27] Mach: A System Software Kernel - R. Rashid & D. Julin & D. Orr & R. Sanzi & R. Baron & A. Forin & D. Golub & M. Jones - Carnegie Mellon University

[28] MACH Environment Manager - M. R. Thompson - Carnegie-Mellon University

[29] MIG - The MACH Interface Generator - R. P. Draves & M. B. Jones M. R. Thompson - Carnegie-Mellon University

[30] MACH Kernel Interface Manual - R. V. Baron & D. Black & W. Bolosky & J. Chew & R. P. Draves & D. B. Golub & R. F. Rashid & A. Tevanian, Jr & M. Wayne Young - Carnegie-Mellon University

[31] The Mach cpu_server: An Implementation of Processor Allocation - D. L.Black - Carnegie-Mellon University

[32] In-Kernel Servers on Mach 3.0: Implementation and Performance - J. Lepreau & M. Hibler, B. Ford & J. Law, and Douglas Orr - University of Utah