- @ # Informatica - @ # Il KERNEL # @ -

Il ***Kernel*** è forse la parte più importante del sistema operativo. In sostanza si tratta di un codice che si occupa della gestione delle risorse presenti nel nostro computer hardware) e le rende direttamente accessibili agli altri programmi che girano sul PC.

Precisamente, il ***Kernel*** si occupa della gestione della CPU, della RAM, del File System , delle Periferiche di sistema, ecc.

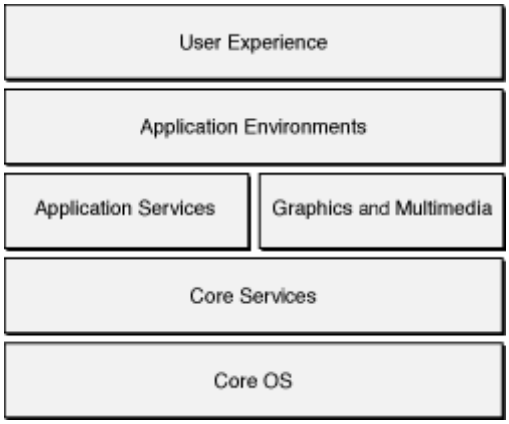
In parole molto semplici, quindi, il *kernel* è un mediatore che serve a far interagire il software con l'hardware.

Naturalmente, un kernel non è strettamente necessario per far funzionare un elaboratore. I programmi possono essere infatti direttamente caricati ed eseguiti sulla macchina, a patto che i loro sviluppatori ritengano necessario fare a meno del supporto del sistema operativo: questa era la modalità di funzionamento tipica dei primi elaboratori, che venivano resettati prima di eseguire un nuovo programma.

In un secondo tempo, alcuni software ancillari come i program loader e i debugger venivano lanciati da una ROM o fatti risiedere in memoria durante le transizioni dell'elaboratore da un'applicazione all'altra: essi hanno formato di fatto la base per la creazione dei primi sistemi operativi.

KERNEL ONION "ovvero Cipolle"

Il tipico modello cognitivo utilizzato per esporre architetture informatiche molto strutturate è la classica cipolla. Come in una cipolla, ci sono diversi strati, uno sopra l'altro, che partono dal cuore della cipolla e via via crescono verso l'esterno, l'ambiente operativo si può descrivere come una cipolla.



I diversi strati, chiamati layer, partono dal cuore. Il primo layer, quello più interno, è infatti chiamato Core OS: qui si trova il kernel, i driver dei vari dispositivi ed i comandi di basso livello del sistema operativo. Qui risiede

Darwin, una sorta di cappello per tutta una serie di tecnologie, che può essere considerato a tutti gli effetti un sistema operativo della famiglia Unix, nella sua variante BSD. Darwin è open-source (nel senso che il codice sorgente è disponibile a tutti per miglioramenti e modifiche) ed è utilizzabile anche da solo. Tra le varie tecnologie incorporate in Darwin c'è Mach, elemento di base di tutta la faccenda, che provvede ad una serie di servizi di base (memoria protetta, gestione del multi-tasking, memoria virtuale, supporto real-time) senza i quali nulla funzionerebbe; c'è il sottosistema BSD, che fornisce comandi di basi per la gestione dei processi, della politica degli accessi ed il supporto di base per la rete; ci sono i driver dei dispositivi (mouse, tastiera, eccetera); c'è il supporto dei vari file system (molti, e non solo quelli tipici di Apple, ma anche Unix e Windows, anche se mi irrita l'impossibilità di scrittura su NTFS); c'è il supporto per i servizi di base della rete.

Il layer successivo è chiamato Core Services. Si tratta in pratica di una libreria di funzioni a disposizione di tutti (non solo del sistema operativo, ma anche e soprattutto delle applicazioni) per la gestione di tipi di dati astratti (stringhe, ad esempio, ma anche URL, dati XML, le Preferenze, eccetera). è (dovrebbe essere) altamente ottimizzata, ed alla base di ogni possibilità di scambiare informazioni tra applicazioni e sistema operativo. In pratica, all'interno dello strato Core Services si trovano tutte quelli funzioni che non hanno a che fare con l'interfaccia grafica. Lo strato cerca di dare un'aspetto più pulito e maneggiabile alle funzioni messe a disposizione dal kernel. Se ad esempio nel kernel ci sono tutte le chiamate per gestire i file (aprire un file, chiudere un file, eccetera, ad un livello molto basso), nei Core Service si trova il File Manager, che fa le stesse cose in maniera più pulita, e combina opportunamente il tutto per rendere più facile la programmazione.

Il passo successivo è in realtà bipartito, perché si trovano due layer in parallelo. Il primo si chiama Application Services, e raccoglie tutta una serie di tecnologie che possono essere utilizzate in maniera uniforme da chiunque. In questo modo, applicazioni diverse riutilizzano gli stessi componenti software per eseguire le stesse cose (che so, visualizzazione di pagine HTML attraverso il WebKit, sistemi di ricerca attraverso il Search Kit, cose del genere), con grande beneficio per l'utente finale, che non deve imparare modi diversi per fare le stesse cose. Il secondo è chiamato Graphics And Multimedia, e come si evince dal nome raccoglie tutte le tecnologie per la visualizzazione 2D e 3D delle informazioni, compresa Quicktime (e quindi anche tutto l'audio). Si tratta di un layer bello corposo, in quanto rientrano appunto QuickTime, Open GL, ma anche Core Audio e Core Video, i sistemi di stampa, ColorSync, eccetera. Questi due layer forniscono in pratica tutto ciò che occorre ad una applicazione per interagire con un utente a livello schermo, audio e video.

L'ultimo layer di interesse è finalmente lo Application Environment, che chiamerò in italiano l'ambiente applicativo, all'interno del quale si possono sviluppare le applicazioni Macintosh. È da qui che, per la maggiornanza degli sviluppatori, si parte per lo sviluppo dell'applicazione. In realtà non esiste un

solo ambiente applicativo, ma ne esistono diversi, a seconda delle tecnologie utilizzate: Carbon, Cocoa, Java, Applescript, BSD/X11 e WebObjects (una volta, e per qualcuno è ancora disponibile, esisteva un ulteriore ambiente, chiamato Classic, dove programmare per Max OS 9). Questi ambienti applicativi forniscono i diversi mattoncini con cui si possono costruire le applicazioni. Un programmatore più decidere di usare i mattoncini di tipo BSD per fare la propria applicazione, un altro può scegliere i mattoncini di tipo AppleScript, un terzo Carbon. Io invece ho scelto i mattoncini di tipo Cocoa (ma ci saranno invasioni di campo, o meglio, di vari altri campi).

Costruire un ambiente applicativo è lo scopo finale di un sistema operativo. Un sistema operativo copre fondamentalmente i dettagli operativi del calcolatore sul quale risiede, nascondendo appunto le varie differenze hardware, interagendo con il mondo esterno, e mascherando il tutto attraverso una API, Application Programmatic Interface (qualcosa del genere), in pratica una collezione di funzioni che permettono il funzionamento delle applicazioni. Di più: chiamare funzioni di questa API è l'unico modo che hanno le applicazioni per interagire con il calcolatore.

Detto questo, appare chiaro che Mac Os X è un sistema operativo molto ricco: non fornisce un solo ambiente applicativo, ma sei! In realtà, mentre Carbon, Cocoa, Java e AppleScript sono in grado di costruire applicazioni sostanzialmente simili, quelle con la tipica interfaccia Macintosh, gli ambienti BSD/X11 e WebObject sono adatti solo per particolari applicazioni.