SISTEMI OPERATIVI

Dopo avere introdotto i principali elementi che costituiscono la struttura fisica del calcolatore (*hardware*), esaminiamo il *software*, ovvero tutto l'insieme di procedure (costituite da programmi, cioè sequenze di istruzioni del linguaggio macchina della CPU utilizzata all'interno del computer) che servono a controllarli.
A livello logico, un calcolatore può essere visto come una struttura "a gusci concentrici" avente come nucleo l'hardware. Esso è circondato da un insieme di gusci concentrici, i più interni dei quali operano più strettamente a contatto con esso, mentre in quelli più esterni sono privilegiate le interazioni con l'utente o, più in generale, con il mondo esterno. Il guscio più interno è il cosiddetto Sistema Operativo; più esternamente si trovano gli ambienti di sviluppo, che consentono all'utente di realizzare programmi utilizzando *linguaggi ad alto livello*, che usano cioè espressioni vicine al linguaggio naturale per descrivere le operazioni; infine, nella parte più esterna, troviamo i *programmi applicativi* e le cosiddette *interfacce utente*. Tutto il sistema di gusci esterni costituisce il software. Il sistema operativo, che si trova a immediato contatto con l'hardware ha lo scopo di sfruttare nel modo migliore le risorse della macchina rendendone l'uso il più efficiente possibile. Esso è anche la struttura di livello più basso con cui può comunicare l'utente. (v. Figura 1).

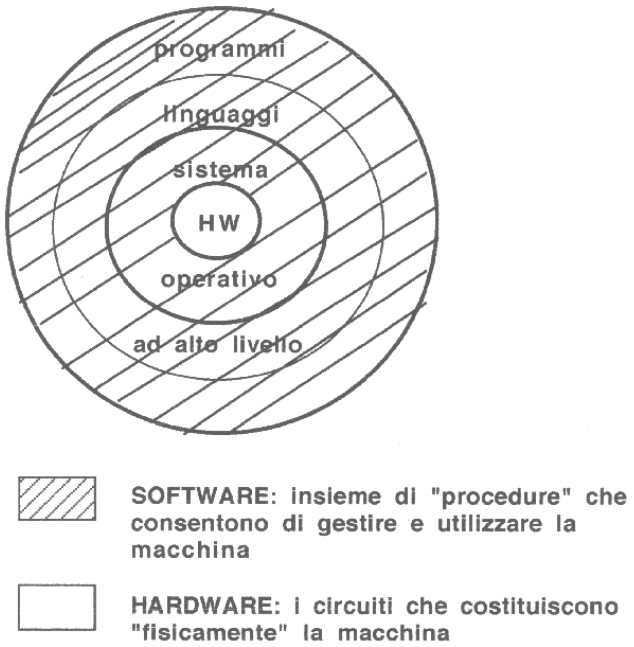

Fig. 1

Il rapporto fra utente e sistema operativo avviene, a sua volta, a diversi livelli, a seconda delle esigenze dell'utente stesso. E' possibile classificare i sistemi operativi in base al tipo di interazione che offrono all'utente. Nei sistemi *monoutente* una sola persona ha il controllo del computer e quindi interagisce con il sistema di calcolo sia per elaborare i propri dati che per modificare o impostare la configurazione di tutto il sistema. In un sistema *multiutente* più *utenti finali* possono contemporaneamente utilizzare i programmi applicativi o generare programmi propri tramite i linguaggi e i relativi compilatori, ma usano il sistema operativo principalmente per elaborare i propri dati. Diverso è il ruolo del *gestore del sistema* che, invece, interagisce essenzialmente col sistema operativo e deve adattarlo alle diverse possibili configurazioni, corrispondenti a diverse situazioni di uso, a seconda delle risorse disponibili e del numero di utenti che accedono contemporaneamente al sistema. Ad esempio, può stabilire diverse priorità, cioè decidere se certi utenti o certi programmi debbano essere privilegiati rispetto ad altri nello sfruttamento delle risorse della macchina. I sistemi operativi si classificano ancora in due ulteriori grandi classi: (a) sistemi operativi che offrono un ambiente di *monoprogrammazione*, nel qual caso in memoria centrale si trova il sistema

operativo (o le sue parti essenziali) ed, al più, un programma che deve essere eseguito, e (b) sistemi operativi che lavorano in *multiprogrammazione*, in cui, invece, oltre al sistema operativo, in memoria centrale si possono trovare più programmi contemporaneamente, che vengono eseguiti apparentemente in modo contemporaneo.

A questo proposito, il concetto di multiprogrammazione è strettamente collegato a quello di *time-sharing*, un sistema che fa sì che le risorse del calcolatore siano assegnate per prefissati intervalli di tempo, in successione, ai diversi programmi in esecuzione in quel momento. Gli intervalli di tempo possono essere più o meno lunghi l'uno rispetto all'altro, a seconda della priorità di cui gode l'utente o il programma che l'utente utilizza, ma comunque sono di brevissima durata. In questo modo, si ottiene l'impressione che vi siano molti programmi che vengono eseguiti contemporaneamente, anche se, ovviamente, ogni istruzione che la CPU esegue appartiene ad un singolo programma e quindi, in uno specifico istante, è solo uno il programma effettivamente in esecuzione. Il time-sharing si può realizzare anche in monoprogrammazione; infatti non è detto che, istante per istante, debbano essere presenti in memoria tutti i programmi che vengono eseguiti, cioè si può avere time-sharing anche con un solo programma, volta per volta, presente in memoria insieme al sistema operativo, anche se è un caso limite. Anche in questo caso, anche se non in modo rigoroso, si può parlare di multiprogrammazione, perché l'effetto è anche in questo caso quello di più utenti o più programmi che utilizzano la macchina nello stesso tempo: la differenza è solamente relativa alla allocazione del programma, o dei programmi, in memoria.

Le operazioni fondamentali che il sistema operativo gestisce sono:
- l'esecuzione di programmi
- il controllo del trasferimento dei dati da e verso le periferiche
- la gestione dei file.

Per *file* si intende un insieme di dati logicamente correlati fra loro e raggruppati ai fini di archiviazione o di utilizzo per una successiva elaborazione. Un file può essere di due tipi, a seconda che contenga dati oppure un programma: un file è quindi una sequenza di codici binari che vengono raggruppati secondo un formato che varia da sistema operativo a sistema operativo e vengono memorizzati nella memoria di massa. Se i file contengono sequenze di istruzioni in linguaggio macchina sono detti *file eseguibili*. Questo significa che le sequenze binarie che contengono devono essere interpretate come istruzioni di un programma che può essere eseguito direttamente dalla CPU, cioè, come già detto, un programma in linguaggio macchina. L'accesso ai file, in un sistema operativo multiutente, può essere limitato, per cui certi file sono considerati proprietà di certi utenti o di certi programmi e quindi non tutti gli utenti o tutti i programmi possono avere accesso a tutti i file. Questo consente di garantire la privacy ai diversi utenti e/o ad evitare che un utente accidentalmente danneggi alcuni file critici per il funzionamento del sistema (*file di sistema*). L'accesso a questi ultimi è generalmente riservato al gestore del sistema. Un'altra limitazione all'accesso può essere relativa al modo in cui si accede al file, una volta che si è autorizzati a farlo. Tipicamente l'accesso può avvenire sia in *scrittura*, nel momento in cui si crea un nuovo file o in cui se ne modifica il contenuto, che in *lettura*, quando si accede al contenuto senza modificarlo. Tuttavia in alcuni casi si possono avere file accessibili in sola lettura, cioè l'utente abilitato ad accedere al file può leggere quello che è contenuto al suo interno ma non può modificarlo: è questo di solito il caso di alcuni file di sistema. L'accesso in sola lettura serve anche per proteggere dati che rivestano particolare importanza. L'accesso può avvenire anche in sola scrittura; ovviamente ci sarà sempre almeno un utente o un programma abilitato a leggere un file; ad es., se esiste un file che contiene informazioni private (come le password) di vari utenti, un utente può modificare le informazioni che lo riguardano, quindi è abilitato a scrivere in quel file, ma non lo può leggere, per proteggere la riservatezza di quelle degli altri. I casi di accesso in sola scrittura sono comunque molto più rari rispetto a quelli in sola lettura. La lettura dei file eseguibili in vista dell'esecuzione del programma che contengono implica anche il trasferimento in memoria, in posizioni prestabilite, delle informazioni (istruzioni) in essi registrate (*caricamento del programma*).

Vediamo cosa avviene al momento dell'accensione della macchina: abbiamo detto che, in quel momento, la memoria centrale è vuota, cioè i byte che la compongono assumono o tutti valore 0, o valori casuali. Perciò qualunque dato di cui avremo bisogno successivamente dovrà essere inizialmente letto da qualche altro dispositivo e trasferito in memoria, perché in memoria la CPU possa trovare almeno una porzione minimale di codice da eseguire, per poter iniziare a svolgere le operazioni da cui dipende il funzionamento del sistema. Quindi, al momento dell'accensione, è necessario caricare in memoria un insieme minimale di istruzioni eseguibili dalla macchina. Più in dettaglio, la CPU, non appena riceve tensione, va a leggere e a eseguire un programma che è allocato in una prefissata posizione di una memoria ROM, e quindi non volatile: viene quindi eseguito un programma che carica in memoria centrale (RAM) i processi che sono parte del sistema operativo e sono necessari al funzionamento della macchina. Per quanto riguarda il sistema operativo, di solito viene caricato in memoria il cosiddetto *nucleo,* costituito dalle procedure più elementari su cui esso fa affidamento per gestire il sistema. Questo processo di accensione e di inizializzazione della macchina viene detto *bootstrap*; di solito, tale processo comprende anche un test dell'hardware o, almeno, della memoria centrale. Il bootstrap si può essenzialmente distinguere in due tipi: bootstrap *a freddo*, effettuato a macchina completamente spenta, e bootstrap *a caldo*; in questo caso, quando viene eseguito, la macchina è già accesa ma si sono verificate o condizioni di errore che ne impediscono il funzionamento corretto, o la macchina si trova in una configurazione che non è più adatta a ciò che bisogna fare. In questo caso è possibile far ripartire la macchina, anche senza spegnerla, inviando un comando di *reset*, che è un comando software. Alla fine di questa operazione, il calcolatore si trova in una condizione uguale a quella successiva ad un bootstrap a freddo, senza però aver dovuto fare tutta una serie di procedure (test dell'hardware, inizializzazione, ecc.) che vengono di solito effettuate solo all'accensione della macchina.

Anche il sistema operativo ha una struttura logica a gusci concentrici, rappresentabile come in figura 2.

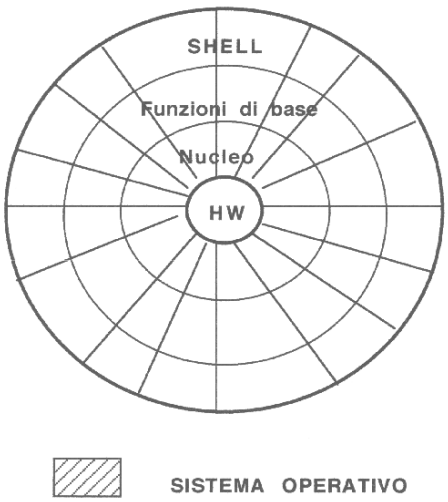

Fig. 2

Il sistema operativo può essere diviso in tre strati, che si collocano a livelli diversi rispetto all'hardware e al mondo esterno: un insieme di procedure di base, che costituisce il nucleo, a diretto contatto con l'hardware; una serie di funzioni elementari che interagiscono direttamente il nucleo e con le quali si possono poi costruire operazioni molto più complesse; infine c'è la vera e propria interfaccia tra utente e sistema operativo, cioè la *shell* (cioè conchiglia, in quanto strato più esterno). Le shell possono essere di due tipi: alfanumeriche, con comandi in linea (come nel sistema operativo MSDOS), o grafiche, a finestre, come ad es. il sistema operativo Apple (Mac OS) o in Windows. Nel primo caso l'utente impartisce comandi e li esprime mediante stringhe di caratteri, con opportune combinazioni che seguono una sintassi che attribuisce

ad ogni stringa un diverso significato, a seconda del suo contenuto e della posizione che occupa all'interno del comando. Nel caso delle shell grafiche (dette anche GUI=graphical user interface), a ciascun programma in esecuzione viene assegnata una zona del video detta *finestra*, all'interno della quale si trova di solito la cosiddetta *barra dei menu*, riservata a una serie di *pulsanti*, attivabili con il mouse, che consentono di far apparire un elenco (*menu*) di operazioni eseguibili. Ai lati della finestra si trovano le cosiddette *barre di scorrimento*. Siccome la finestra può essere posizionata e dimensionata a piacere sullo schermo, questo può far sì che non tutto l'output video del programma possa essere visualizzato; le barre di scorrimento orizzontali e verticali consentono di inquadrare di volta in volta una porzione diversa dell'output del programma nella finestra.
Sia in ambiente di mono che di multiprogrammazione è possibile aprire contemporaneamente più shell; la differenza è che, mentre in ambiente di multiprogrammazione le shell sono situate logicamente allo stesso livello, cioè nel momento in cui abbiamo più shell aperte queste sono contemporaneamente attive e si può passare a una shell da un'altra senza interromperne l'attività, in monoprogrammazione si esaspera la struttura a gusci concentrici nel senso che le shell vengono a sovrapporsi su livelli diversi, cioè nel momento in cui è attiva una shell le altre non sono visibili. Ad esempio, se abbiamo aperto tre shell, se vogliamo tornare alla prima dobbiamo chiudere la terza e la seconda, per tornare a lavorare con quella che ci interessa. Infatti, in monoprogrammazione può essere attiva una sola shell alla volta (quella più esterna, vedi fig. 3).

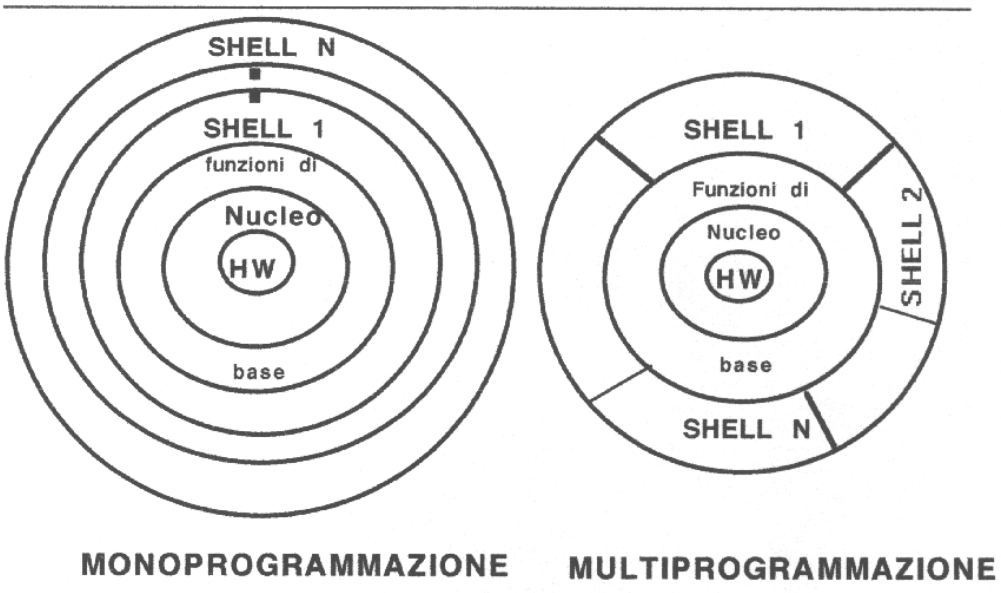

Fig. 3

In ambiente di multiprogrammazione si definisce il programma con cui l'utente può interagire in quel momento come programma che si svolge in *foreground,* cioè in primo piano; per contrapposizione, abbiamo i programmi che si svolgono in *background*, cioè in sottofondo, che sono quelli che, pur essendo eseguiti contemporaneamente a quello con cui sta interagendo l'utente, non sono influenzati dalle azioni che quest'ultimo compie. Un esempio di programma che spesso viene eseguito in background è il trasferimento di dati da un calcolatore a un altro, o dal calcolatore a una periferica lenta, come ad esempio la stampante. Poiché il trasferimento di dati è un processo piuttosto lento, a causa della lentezza dei dispositivi periferici rispetto alla CPU, bisogna essere in grado di continuare la propria attività mentre vengono eseguite operazioni che, in alcuni casi, possono richiedere anche qualche minuto. Per ottenere l'effetto desiderato è possibile lanciare sul calcolatore in foreground il programma con cui si vuole interagire e in background il programma di trasferimento dati. In questo modo il tempo di CPU è suddiviso fra programma in foreground (tipicamente con una priorità maggiore, che gli consente di sfruttare la CPU per una percentuale maggiore di tempo) e programma in background. Il trasferimento o la stampa diventano ancora più lenti ma è possibile continuare il proprio lavoro mentre il trasferimento viene eseguito invece di aspettare che tutto il processo di

trasferimento dati sia terminato prima di potere utilizzare un programma diverso da quello che sta gestendo il trasferimento stesso.