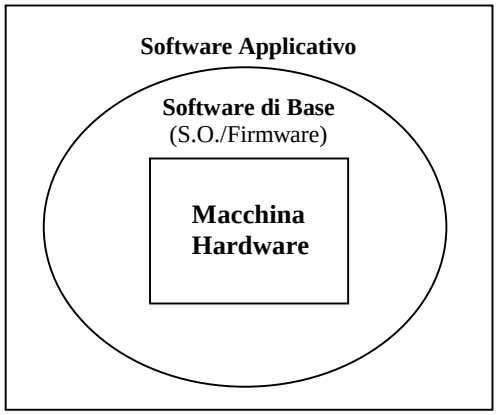
2. Sistemi Operativi

2.1 Software

2.1.1 Tipi di software

Un **programma** è una sequenza di istruzioni elementari che possono essere eseguite dal sistema di elaborazione; ogni programma lavora su di una serie di informazioni che costituiscono l'*input* e fornisce dei risultati che vengono detti *output.* Un programma in esecuzione viene detto *processo*.
Si definisce come **software** l'insieme dei programmi che possono operare sul calcolatore, cioè la componente *logica* di un elaboratore, in contrapposizione alla parte fisica detta *hardware*.
Il software di un sistema informatico viene normalmente suddiviso in due categorie:

- *Software di base*: dedicato alla gestione delle funzioni elementari dell'elaboratore; tale software lavora direttamente sul livello fisico (hardware) della macchina;
- *Software Applicativo*: dedicato alla realizzazione di particolari esigenze dell'utente e che riesce ad agire sull'elaboratore solo con il tramite del software di base.



2.1.2 Sistema Operativo

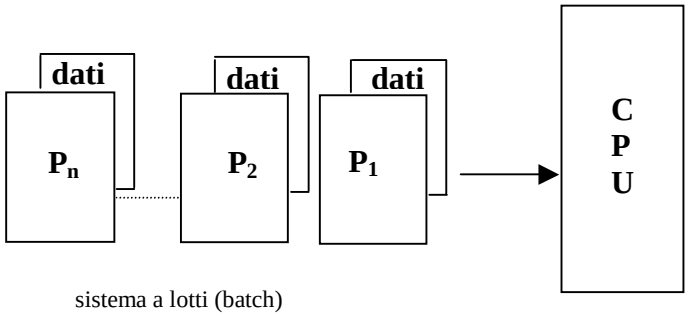
Il componente principale del software di base è il **sistema operativo**, il programma deputato a gestire le varie risorse fisiche dell'elaboratore svolgendo compiti differenziati a seconda della complessità del sistema posto sotto il suo controllo.
Il sistema operativo, in pratica, opera su due livelli:

a) gestisce le risorse del sistema di elaborazione:
   - processore
   - memoria centrale
   - memoria di massa
   - dispositivi di input/output

b) fornisce l'interfaccia uomo – macchina
   (crea un ambiente adatto alle esigenze di lavoro dell'utente)

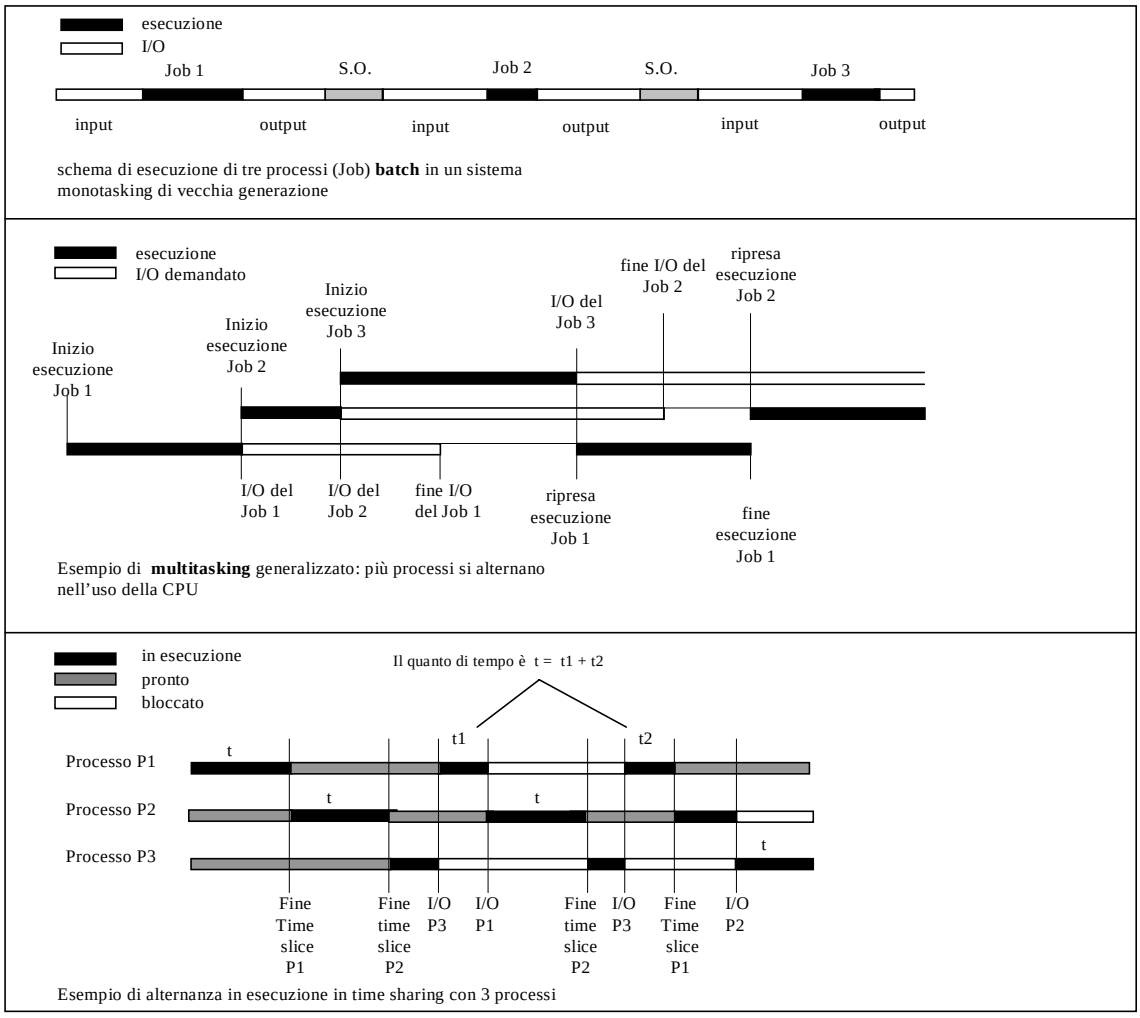
Il Sistema Operativo di un elaboratore è quindi l'insieme dei programmi che consentono le operazioni elementari della macchina quali, ad esempio, lettura e scrittura dalla e sulla memoria, gestione delle periferiche, supporto per i programmi applicativi e controllo degli errori e degli eventi particolari.

Ad una parte del sistema operativo residente in memoria ROM, il *firmware*, è demandato l'**avvio della macchina** (*bootstrap*), che consente il caricamento in memoria del sistema stesso e la verifica dell'integrità delle componenti interne e dei collegamenti con le periferiche esterne.
I sistemi operativi si dividono in **monotasking** e **multitasking** in dipendenza del fatto che il processore possa eseguire un solo lavoro per volta o che sia in grado di gestire più lavori contemporaneamente.
I primi sistemi operativi adottavano una modalità *monotasking* detta **a lotti** (*batch*) che consisteva nel caricare tutti i dati necessari a far eseguire completamente un programma senza interventi dall'esterno ed attendere la sua fine prima di poter preparare il sistema per un nuovo lavoro.



sistema a lotti (batch)

In tempi più recenti il sistema operativo DOS, pur utilizzando la modalità *monotasking*, permetteva all'utente di interagire con il programma in corso fornendo dati e ottenendo anche risultati parziali.
Il grosso handicap di tale modalità è il fatto che la CPU resta inattiva ogni volta che il programma in esecuzione rimane in attesa di dati dalla periferia o li sta ad essa fornendo; tenendo conto che i tempi di trasferimento dei dispositivi periferici sono di parecchi ordini di grandezza più grandi di quelli di elaborazione pura, si comprende l'enorme spreco di tempo di CPU che risulta utilizzata solo per una piccolissima percentuale.
Per ovviare a questo problema sono nati i sistemi operativi **multitasking** che, pur eseguendo istruzioni di un processo per volta, ottimizzano il tempo di utilizzo del processore pur dando all'utente l'impressione di vedersi dedicate tutte le risorse del sistema per il proprio lavoro.
Il risultato è ottenuto eseguendo, a rotazione, piccole porzioni di tutti i processi in attesa di essere eseguiti; inoltre se un lavoro richiede una lunga operazione sui dispositivi periferici, la CPU viene destinata al successivo lavoro ed il controllo torna al processo sospeso quando l'operazione di input/output è terminata.
Questa tecnica viene chiamata **time-sharing** perché suddivide il tempo di CPU tra i vari processi che concorrono al suo utilizzo; gli **intervalli di tempo** (*time-slice*) sono talmente piccoli da non poter essere percepiti dall'utente.



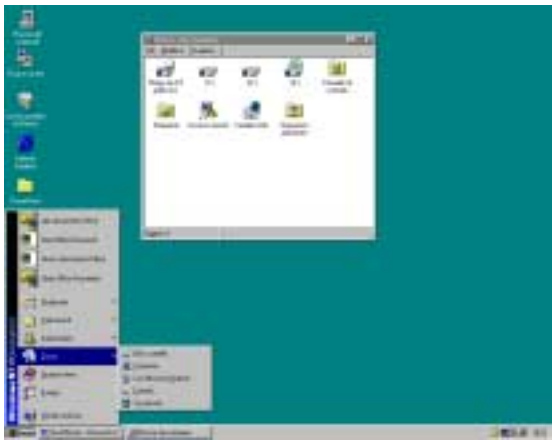
Esiste anche un'altra tecnica multitasking detta elaborazione in **tempo reale** (*real time*), che alterna i processi in esecuzione a seconda delle priorità ad essi accordate. Tale modalità viene usata quando alcuni lavori sono talmente importanti da non poter venire interrotti.

I sistemi operativi possono anche essere suddivisi in base al metodo con cui l'utente può interagire con la macchina, attraverso quella che viene detta **interfaccia** computer-utente; secondo tale criterio si possono riconoscere: sistemi ad interfaccia *testuale* (ad esempio MS-DOS), sistemi ad interfaccia *grafica* (ad esempio Windows) e sistemi ad interfaccia *fisica.*

In realtà Windows non nasce come S.O. autonomo ma come software da utilizzare tramite DOS ; solo con Windows 95 non si ha semplicemente una nuova versione di Windows, ma un vero e proprio S.O. autonomo che non necessita di DOS. Ulteriori sviluppi si sono avuti poi con Windows 98 e Windows 2000 fino ad arrivare al più recente Windows XP (nelle due versioni Professional , per uso appunto professionale, ed Home, per uso privato).

Nei sistemi ad **interfaccia testuale** i comandi vengono forniti dall'utente tramite stringhe di caratteri (parole) seguendo una grammatica ed una sintassi ben definite. I vantaggi sono la poca memoria richiesta, la velocità elevata e la disponibilità di tutti i comandi allo stesso momento, mentre gli svantaggi sono legati alla difficoltà di ricordarsi i comandi correttamente.

interfaccia testuale



Nei sistemi ad **interfaccia grafica** o **GUI** (*Graphical User Interface*) i comandi sono forniti dall'utente tramite la selezione dell'immagine che è legata alla operazione richiesta.
I vantaggi per l'utente sono la facilità d'uso e la totale trasparenza rispetto all'hardware sottostante, mentre gli svantaggi sono la pesantezza del sistema operativo (lento e grande) e la disponibilità dei comandi complessi solo dopo diversi passaggi.
Infine, nei sistemi ad **interfaccia fisica**, i comandi sono forniti tramite movimenti di componenti fisici del sistema e sono usati negli *embedded computer* (elaboratori che si trovano all'interno di macchine più complesse). Un esempio sono i robot utilizzati nelle fabbriche: per mezzo di questi sistemi vengono "appresi" (trasformati da movimenti in impulsi e quindi associati a comandi) i movimenti fisici che il robot deve effettuare per svolgere le attività cui è preposto e che costituiscono il "programma" associato a tali attività; al momento opportuno l'operatore lancerà tale programma ed il robot eseguirà le azioni ad esso associate.
I vantaggi sono nella spontaneità con cui sono forniti i comandi, mentre gli svantaggi sono legati al fatto che tali programmi sono strettamente dipendenti dal tipo di sistema informatico usato (normalmente non standard) e richiedono notevoli personalizzazioni.

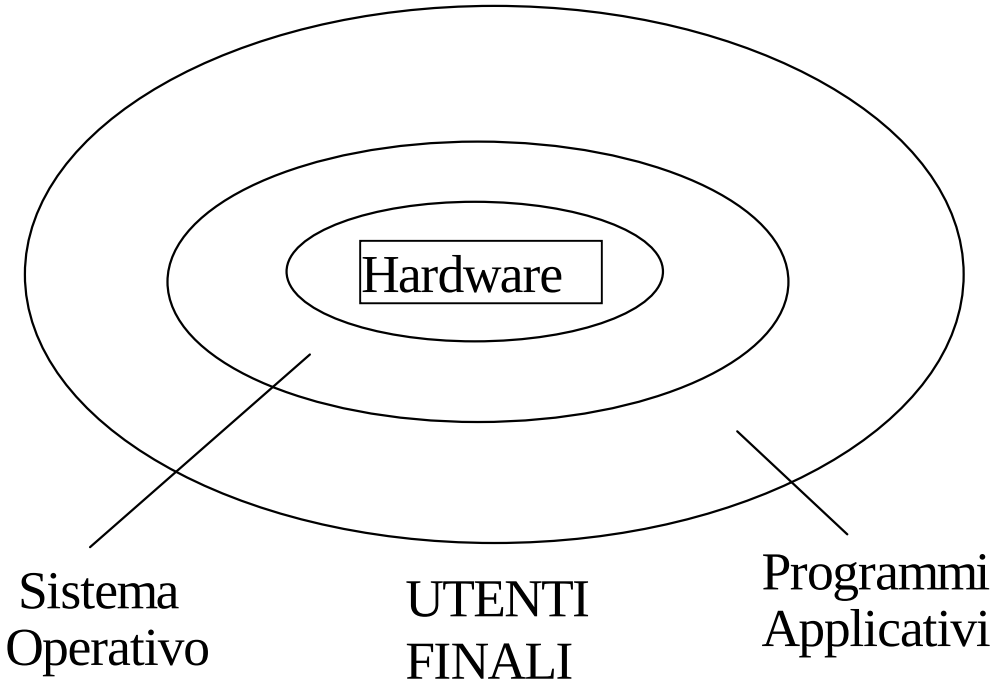
2.1.3 Software applicativo

Il **software applicativo** è formato da programmi costruiti per risolvere specifici problemi degli utenti; una delle caratteristiche fondamentali di tali programmi deve essere la facilità di utilizzo e quindi l'interfaccia deve essere amichevole (*user-friendly*) per consentire anche all'utente meno esperto di fruire efficacemente del prodotto.
Tale software è tradizionalmente suddiviso in *linguaggi di programmazione, software personale, strumenti di produttività e programmi "verticali"*.
I **linguaggi di programmazione,** o più propriamente *gli ambienti di sviluppo dei linguaggi*, agevolano la scrittura dei programmi applicativi e la verifica della loro correttezza ; tali programmi risentono in misura ridotta o nulla della struttura hardware del sistema sottostante e sono quindi facilmente *portabili* da un sistema informatico ad un altro.
Il **software personale** è formato da quei programmi che consentono anche ad un utente inesperto di fruire dell'aiuto del computer per attività tipiche della vita quotidiana quali scrivere e comunicare.
Tra tali programmi ricordiamo i sistemi di **videoscrittura** *(word-processing system)*, le **agende elettroniche** e i gestori della **posta elettronica** *(e-mail)*.
Tra i due tipi di software precedenti, l'uno dedicato all'utente esperto l'altro all'utente non smaliziato, trovano spazio gli **strumenti di produttività** che consentono di svolgere alcuni tipici compiti di programmazione in modo semplice ed efficiente.

Tipici strumenti di produttività sono i **fogli elettronici** *(spreadsheet)* ed i **sistemi per la gestione di basi di dati** *(database system).*
I **programmi "verticali"** risolvono problematiche altamente specialistiche quali la gestione di una biblioteca o il controllo di un impianto.

2.1.4 Gerarchie di macchine virtuali

L'esperto che scrive un sistema operativo vede il sistema come un insieme di risorse fisiche da comandare direttamente, mentre colui che progetta un ambiente di programmazione vede la macchina come l'insieme delle funzioni messe a disposizione dal sistema operativo.
L'utente poi che progetta un programma applicativo tramite un linguaggio di alto livello non è obbligato a conoscere il metodo di funzionamento del sistema operativo né tantomeno dell'hardware sottostante, in quanto per lui l'elaboratore è semplicemente l'insieme delle funzionalità messe a disposizione dall'ambiente di programmazione.
Per l'utilizzatore di un programma applicativo, infine, il sistema appare virtualmente come l'insieme di comandi che può fornire alla macchina per soddisfare le sue esigenze; questi non deve preoccuparsi di come il software possa poi realizzare effettivamente le sue richieste tramite l'hardware.
In pratica viene a crearsi una *gerarchia* di tante *macchine virtuali* quante sono le categorie di utenti che vanno ad operare sul sistema informatico.
Lo schema, semplificato al massimo, che riassume questa stratificazione di macchine virtuali, è il seguente:



Questo gioco di *scatole cinesi* permette ai vari fruitori del sistema informatico di aver bisogno solo di un insieme limitato di competenze, legato alla macchina virtuale con cui

vogliono interfacciarsi e che, man mano, diminuiscono passando dalla struttura fisica alla parte più esterna.

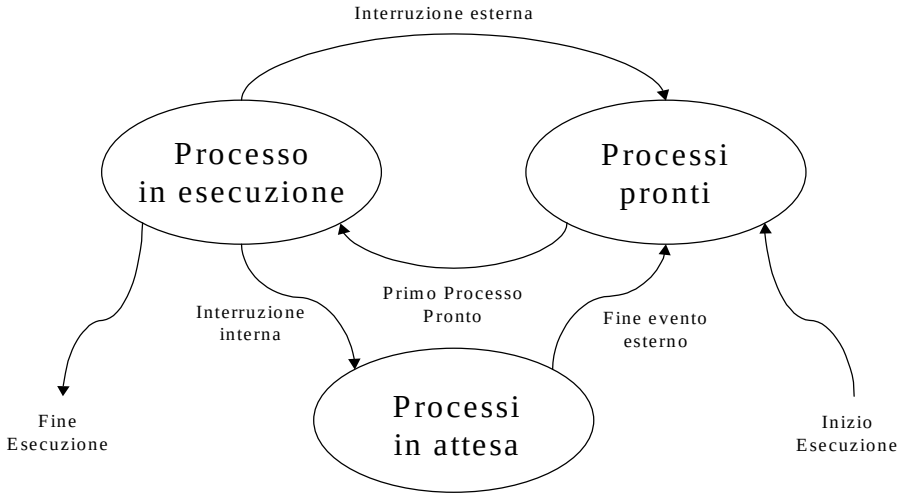
2.2 Funzionalità del sistema operativo

2.2.1 Gestione dell'unità di elaborazione

Il concetto di macchina virtuale è estendibile alla struttura del sistema operativo, in quella che viene chiamata architettura a *buccia di cipolla*. In tale astrazione il sistema operativo è visto come una serie di strati di moduli software; ogni modulo è un *livello* di *macchina virtuale* che può utilizzare tutte le risorse hardware e software delle *macchine* sottostanti (ma non di quelle soprastanti). Queste macchine ricoprono quindi la parte fisica fino ad arrivare al livello *visibile* dai programmi applicativi.
In relazione a quanto detto, una schematizzazione del sistema operativo può essere la seguente:

Il gestore dei *processi* o, più generalmente, il **gestore dell'unità di elaborazione** si occupa del modo in cui vari programmi, contemporaneamente residenti in memoria centrale, sono eseguiti, utilizzando il processore (o i processori) di cui è dotato il sistema in modo da ottimizzare tale risorsa critica.
Per **processo** si intende un programma nel suo divenire; ogni processo è quindi formato da una parte *statica* costituita dalle istruzioni del programma e da una parte *dinamica* determinata dai dati su cui il programma opera e dal suo stato di avanzamento.
In linea di principio, si può pensare che ad ogni programma sia associato un processo, anche se, in alcuni contesti, ciò non è necessariamente vero (un programma potrebbe essere *spezzato* in tre processi diversi che si occupano, ad esempio, dell'acquisizione, dell'elaborazione e della stampa dei risultati).
Se pensiamo ad un sistema operativo multitasking (la quasi totalità dei sistemi in uso), ogni processo può trovarsi in uno dei tre seguenti stati: *in esecuzione*, *pronto* o *in attesa*; solo un processo tra quelli presenti può essere realmente **in esecuzione**, cioè solo uno di essi può occupare, in un dato istante, la risorsa processore.

Gli altri processi possono o essere **pronti** per andare in esecuzione o essere **in attesa** del termine di alcune operazioni di input/output per poter essere poi considerati pronti.
Il percorso del processo in un sistema *time sharing*, dal suo caricamento in memoria centrale alla sua completa esecuzione, può essere riassunto come segue: il processo entra in una coda di processi *pronti* e, quando viene il suo turno, va *in esecuzione*; mentre si trova in esecuzione possono verificarsi degli eventi che ne consigliano la sospensione detti interruzioni (*interrupt*) e che fanno cambiare di stato il processo.
Le interruzioni possono essere **interne** al processo (il programma richiede un'operazione di input/output) o **esterne** (scade il periodo di tempo dedicato a quello specifico processo). Nel caso di interruzione interna il processo entra *in attesa* e verrà rimandato nella coda dei processi pronti solo quando il lavoro della periferica richiamata sarà terminato; se, invece, l'interruzione è esterna allora il processo viene direttamente spostato nella coda dei processi pronti.
Questo alternarsi di stati continua fino a che tutto il processo è andato in esecuzione e, quindi, il programma è terminato.

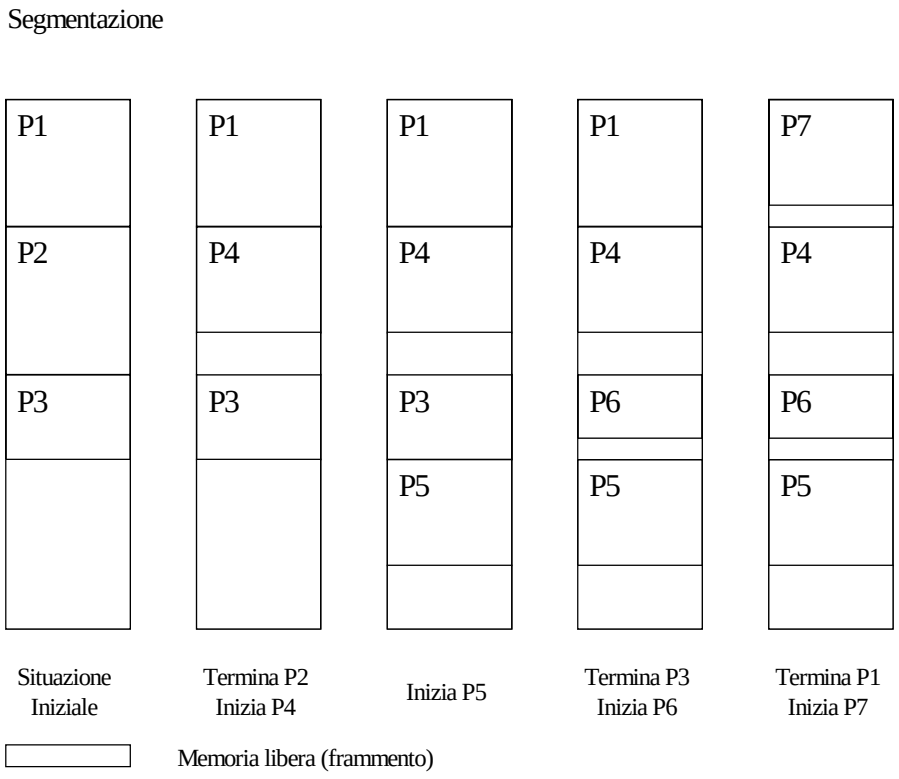


Il gestore dell'unità di elaborazione deve anche preoccuparsi della sincronizzazione dei processi, risolvendo le eventuali contese per una risorsa critica o valutando le esigenze di processi che cooperano per un fine comune.
Nel caso di sistemi multiprocessore, a questa parte del sistema operativo spetta l'onere di gestire il flusso delle istruzioni dei processi ai vari processori e tutte le interazioni fra i vari processi che possono essere contemporaneamente in memoria o in esecuzione.

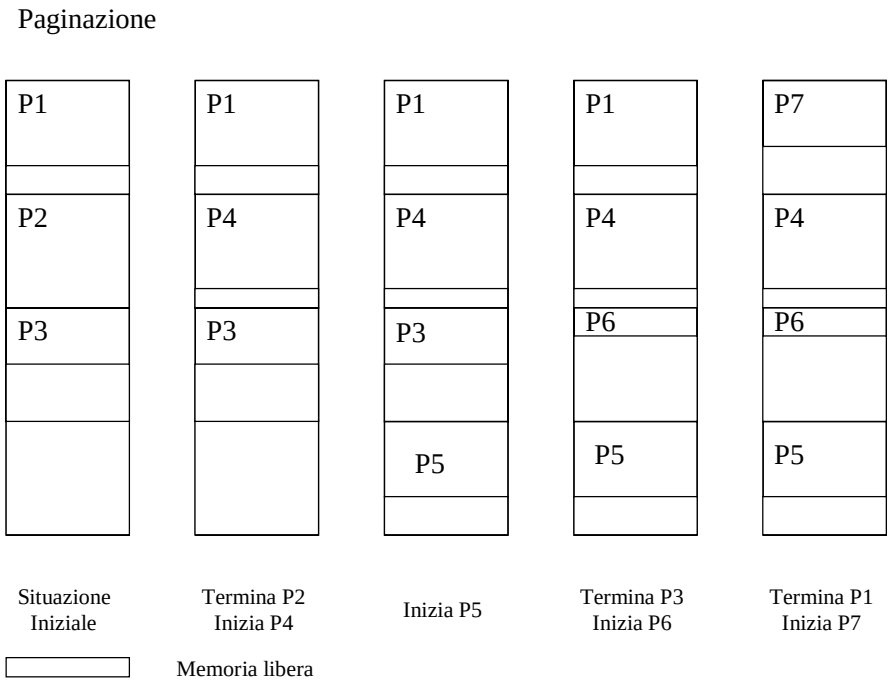
2.2.2 Gestione della memoria centrale

La memoria centrale, similmente all'unità di elaborazione, è una risorsa di sistema generalmente scarsa, da suddividere tra i vari processi in competizione. Il sistema operativo deve quindi occuparsi del modo in cui i singoli programmi occupano la memoria, in modo che ogni processo abbia l'illusione di una risorsa memoria *virtualmente* infinita.

In realtà, la memoria centrale è piuttosto deficitaria rispetto alle esigenze del processo o dei processi che vogliono andare in esecuzione; di conseguenza viene caricata in memoria centrale solo una parte di programma, mentre il resto permane nella memoria secondaria. Quando il processore richiede ulteriori istruzioni viene caricata un'altra parte del programma e così via fino ad esaurire il programma stesso. In questo modo gli utenti lavorano come se avessero in memoria centrale tutti i programmi da loro richiesti *vedendo* una memoria molto più grande di quella reale detta, appunto, **memoria virtuale**.
La dimensione della parte di programma da tenere in memoria centrale è un parametro **critico**: una piccola quantità richiederebbe molti trasferimenti dalla memoria secondaria per terminare l'esecuzione, mentre una grande quantità diminuirebbe il numero dei possibili processi presenti in memoria.
Oltre a dover risolvere il problema della memoria virtuale, il sistema operativo deve decidere come assegnare la memoria centrale ai vari processi; può scegliere infatti di suddividere la memoria in *segmenti* uguali alla parte di programma da caricare o in *pagine* di dimensione fissa più grandi di ogni singola parte di programma.
Ciascuna delle due tecniche presenta pregi e difetti: la prima, detta **segmentazione**, porta la memoria a mettere a disposizione segmenti sempre più piccoli arrivando a quella che si chiama *frammentazione* della memoria. La seguente figura illustra il fenomeno della frammentazione: in una memoria dove già risiedono in maniera ottimale alcuni programmi, la terminazione di alcuni e l'arrivo di altri porta a spezzettare la memoria in segmenti disponibili sempre più piccoli e, in definitiva, inutilizzabili.



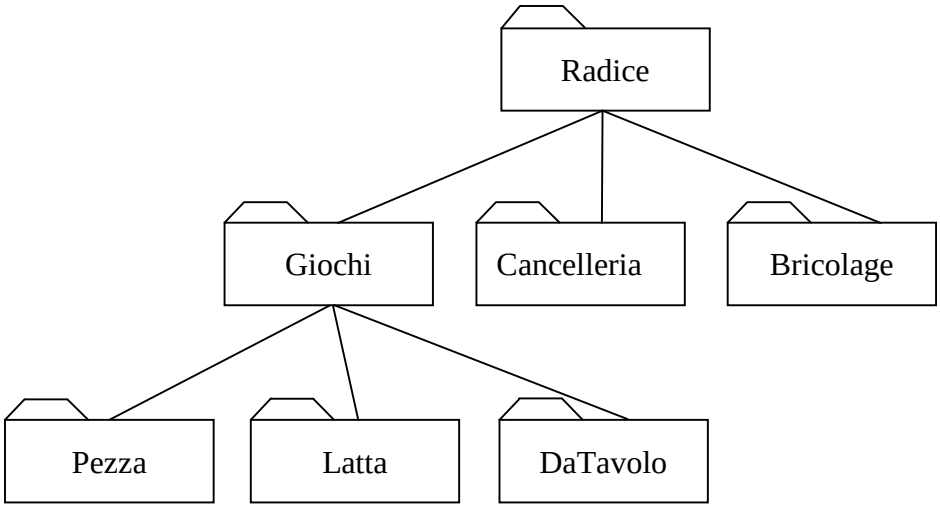
La seconda tecnica, detta **paginazione**, evita la frammentazione ma, in modo insito, spreca memoria visto che assegna al processo sempre più memoria di quella necessaria.



2.2.3 Organizzazione della memoria secondaria

Per poter organizzare correttamente l'enorme mole di dati che può essere contenuta nella memoria di massa si ricorre ad una unità astratta di memorizzazione detta **file**; un file è una sequenza di caratteri che raggruppa una serie di dati *omogenei*, come le istruzioni di un programma o il testo di una relazione. L'insieme di programmi del sistema operativo che si occupano della gestione della memoria secondaria, sia dal punto di vista *logico* che *fisico*, viene quindi detto **file system**.
Scopo principale di un *file system*, dal punto di vista dell'utente, è quello di consentire una organizzazione logica dei vari file raggruppandoli secondo criteri stabiliti dall'utente stesso.
Un *file system* deve, inoltre, dare anche la possibilità di associare ad ogni file un nome, deve gestire la corrispondenza fra tale nome e la posizione fisica realmente occupata (in definitiva trasformare le operazioni logiche su file in operazioni fisiche su blocchi di memoria di massa), fornendo gli opportuni metodi per accedere ai dati in modo da ottimizzare i tempi di reperimento delle informazioni; deve, infine, gestire dei meccanismi di protezione della riservatezza dei dati e lo spazio libero derivante da file cancellati. Il modo in cui un *file system* organizza i vari documenti contenuti all'interno della memoria di massa può essere spiegato ricorrendo al paragone dei **contenitori**. Immaginiamo che la memoria secondaria sia un grande contenitore dove riporre in modo ordinato alcuni oggetti presenti in una camera (i file); all'interno di questo contenitore possono trovare posto altri contenitori più specifici: ad esempio, uno per i giochi, uno per gli oggetti di cancelleria e

uno per gli arnesi del bricolage. A loro volta, tali contenitori potranno essere suddivisi in ulteriori scomparti (uno per i giochi di pezza, uno per quelli da tavolo, ecc.) e così via fino ad ottenere un livello di ordinamento sufficientemente preciso, dove riporre gli oggetti in modo che il loro reperimento sia agevole.
Il contenitore principale prende il nome di **radice** (*root*), mentre gli altri contenitori vengono detti **directory** e la loro dipendenza gerarchica può essere evidenziata da quella che si chiama *struttura ad albero*.



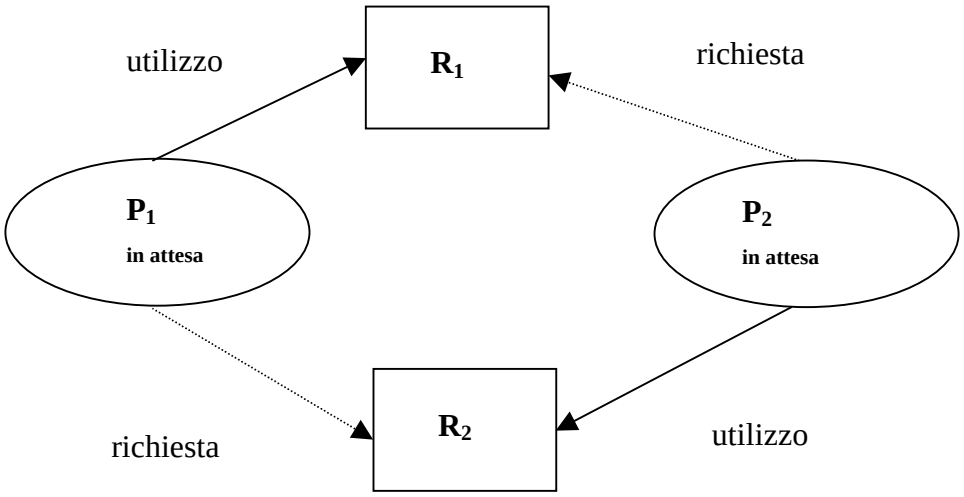
Il **nome completo** di un file (*path name*) deve quindi includere, oltre allo specifico nome del file, anche il **percorso** (*path*) necessario per raggiungerlo; un ipotetico file di nome *Orsetto* che si trovi nella directory *Pezza* sarà quindi individuato con la path name */Giochi/Pezza/Orsetto* (la radice, in quanto unica, può essere omessa).
Il sistema operativo deve gestire anche quello che si chiama **descrittore del file**, l'insieme cioè delle informazioni relative alla posizione fisica e logica del documento e al suo modificarsi nel tempo.
Un tipico descrittore di file conterrà sia dati utili all'utente quali il nome del documento, le date di creazione e di ultima modifica, il tipo di file e le protezioni attive, sia informazioni utili e disponibili solo al sistema operativo quali posizione fisica nella memoria secondaria e descrizione della struttura interna.
Normalmente, per specificare il tipo di file, si fa uso di quella che viene detta **estensione**; l'estensione è un gruppo di caratteri (usualmente 3) che segue il nome del file ed è separato da questo da un punto. Tali caratteri specificano, tramite un *codice*, il contenuto del documento; file con estensione *.COM* o *.EXE* conterranno dei programmi, altri con estensione *.DOC* dei documenti generati dal programma Word e così via (la successiva tabella riporta alcune delle estensioni più comuni).

Estensione	Tipologia del file
.doc	documenti di Word
.mdb	archivi di Access
.xls	fogli elettronici Excel
.ppt	presentazioni PowerPoint
.exe, .com	file eseguibili
.htm,.html	pagine web
.jpg, .gif, .bmp	immagini
.mid, .mp3, .wav	audio

2.2.4 Driver

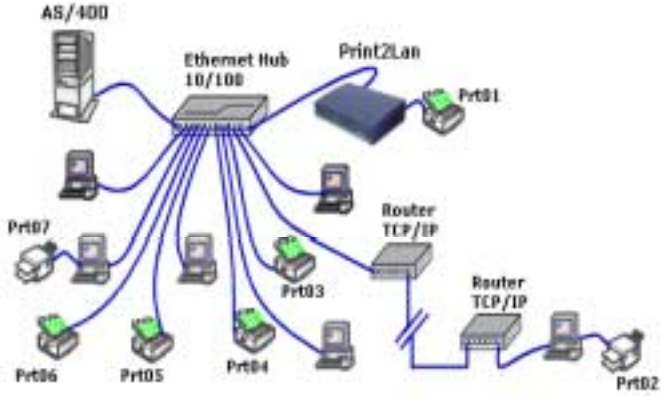
Altro delicato compito del sistema operativo è quello di gestire i rapporti fra l'unità di elaborazione e le varie periferiche in modo che l'utente che richiede una particolare azione su di un dispositivo periferico non sia costretto a conoscere l'esatto modo fisico in cui tale operazione viene eseguita.
Per far ciò esistono particolari programmi, detti **device driver** (o semplicemente driver), che racchiudono tutte le informazioni necessarie alla gestione di una particolare periferica; risulta evidente che per ogni dispositivo esterno deve esistere un particolare driver che ne conosca l'esatto funzionamento fisico. Ci saranno così un driver per gestire la tastiera, uno per gestire il monitor, uno per ogni tipo di stampante e così via.
Ognuno di questi programmi (che fanno parte del sistema operativo) deve inoltre gestire la comunicazione di segnali da e verso la periferica (*buffering*) e coordinare l'accesso alla risorsa, dato che quest'ultima potrebbe essere richiesta contemporaneamente da più processi.
Relativamente alla gestione dell'input/output, il sistema operativo deve assicurare una corretta **gestione degli errori** che si possono verificare nei vari dispositivi periferici (lettura impossibile su di un dischetto, carta terminata in una stampante, ecc.).
Nel caso di sistemi complessi, con molti processi in esecuzione in competizione per le risorse a disposizione, il sistema operativo deve vigilare sulla distribuzione delle risorse garantendo che non si possano verificare blocchi delle elaborazioni dovuti ad attese infinite.
Immaginiamo, ad esempio, la seguente situazione: il *processo* **P1** sta utilizzando la *risorsa* **R1** e viene posto in attesa per poter utilizzare anche la *risorsa* **R2** che è momentaneamente impegnata; il *processo* **P2**, attuale utilizzatore della *risorsa* **R2**, viene invece posto in attesa

visto che vuole servirsi della *risorsa R1*. Evidentemente i due processi sono irrimediabilmente bloccati visto che si è verificata una situazione di stallo non risolvibile facilmente, detta **deadlock** .



2.2.5 Gestione della coda di stampa

La gestione della comunicazione tra unità di elaborazione e stampante risulta essere particolarmente critica per la notevole diversità dei tempi di esecuzione delle operazioni proprie delle due unità. Senza un meccanismo di coordinamento di tale dialogo, l'unità di elaborazione potrebbe completare il suo compito (spedire una serie di dati alla stampante) in un periodo dell'ordine dei microsecondi ed attendere l'esecuzione effettiva dell'operazione (stampa su carta delle informazioni) per un tempo di 8 o 9 ordini di grandezza maggiore prima di poter continuare il suo compito.
Lo **SPOOL** (*Simultaneous Peripheral Operation On Line*) è la parte di sistema operativo che si occupa di ottimizzare tale comunicazione; l'unita di elaborazione, invece di inviare i dati alla periferica e controllare l'esecuzione dell'operazione, li registra sulla memoria secondaria in versione *printer-ready* e si volge ad eseguire altre operazioni.
Viene quindi generato un particolare *processo* che, nei momenti di inattività della unità di elaborazione (in *background*), si occupa di gestire il colloquio e l'eventuale **coda di stampa** che può venirsi a creare se le stampe richieste sono più di una.
Nei sistemi multiutenti dotati di molti terminali o nelle reti locali esistono delle stampanti ad alte prestazioni disponibili a tutti dette *stampanti di rete*; in tal caso si usa dedicare un computer (**spooler server**) per eseguire i processi di *spooling* comandati dai vari utenti svincolando così i singoli terminali anche dalla gestione della stampa in background.



2.2.6 Interprete dei comandi

L'**interprete dei comandi** rappresenta lo strato superiore del sistema operativo, responsabile di ricevere ed interpretare i comandi formulati dall'utente; queste istruzioni richiamano funzioni del sistema operativo o mandano in esecuzione programmi dell'utente o di sistema. In definitiva l'interprete dei comandi si occupa di ricavare, dai comandi forniti dall'utente, le effettive richieste da inviare ai moduli sottostanti.
Nei sistemi ad *interfaccia grafica*, l'utente è aiutato nel colloquio con l'elaboratore grazie al fatto che i comandi sono comunicati o utilizzando semplici *menù a tendina* o attraverso la selezione, tramite mouse, di disegni che simboleggiano l'operazione richiesta detti **icone**.
Quando è invece presente una *interfaccia testuale*, il dialogo tra utente e macchina è più faticoso: l'utente deve infatti conoscere un **linguaggio** elementare, dotato di propria *grammatica* e *sintassi*, per poter impartire ordini all'elaboratore.
In tal caso la logica di funzionamento dell'interprete dei comandi è la seguente:

- nella *linea di comando* l'interfaccia indica la disponibilità ad accettare comandi tramite una serie convenzionale di simboli detta **prompt**;
- l'utente digita il comando relativo all'operazione richiesta con gli eventuali *parametri* e *opzioni*;
- l'elaboratore verifica la presenza di *errori* sintattici: in caso positivo fornisce l'indicazione del tipo di errore, altrimenti esegue il comando richiesto;
- quando l'esecuzione è terminata, riappare il prompt sulla linea di comando e l'elaboratore è pronto per ricevere un nuovo comando.

2.2.7 Esempi di uso della linea di comando

I sistemi che fanno uso della **linea di comando** sono in rapida estinzione, scalzati da sistemi operativi ad interfaccia grafica che risultano essere facilmente utilizzabili (*user-friendly*) anche dall'utente inesperto; questi ultimi saranno oggetto di studio approfondito in altri ambiti (dedicati agli aspetti prettamente operativi).
Esistono, comunque, ancora due istanze di sistemi operativi ad interfaccia testuale diffusamente conosciuti: il *MicroSoft Disk Operative System* (**MS-DOS**) e il sistema **Unix**.
Nel primo caso l'interprete dei comandi è contenuto in un unico programma chiamato **COMMAND.COM** che è in grado di *capire* ed eseguire una cinquantina di comandi elementari.
La struttura generica di un comando in MS-DOS è la seguente:

comando [argomento] [/parametro]

dove **comando** è una qualsiasi istruzione o programma utente e le parentesi quadre indicano l'*opzionalità* del contenuto; uno spazio separa gli eventuali *argomenti* e una barra divide i vari *parametri*.
Il comando **COPY C:\ALFA.DOC A:\BETA.DOC**, ad esempio, *copia* il file di nome *alfa.doc* contenuto nell'unità *C:* (Hard disk) nel file *beta.doc* contenuto nell'unità *A:* (floppy disk), mentre il comando **DIR C:\GAMMA /W/P** mostra il contenuto della directory *gamma* in modo *compatto* (/W) e presentandolo *una pagina alla volta* (/P).
Altri comandi notevoli sono i seguenti:

- **TREE** *Dispositivo*: visualizza il contenuto di un dispositivo di memorizzazione nella forma *ad albero*.
- **DEL** *NomeFile*: cancella il file specificato.
- **REN** *VecchioNome NuovoNome*: cambia il nome ad un file.
- **TYPE** *NomeFile*: visualizza il contenuto di un file.
- **CD** *Directory*: imposta una nuova directory di lavoro (*directory corrente*).

Per poter operare su insiemi di argomenti, l'interprete MS-DOS, come molti altri, consente l'uso dei cosiddetti caratteri *jolly* (usualmente il carattere *asterisco*), detti anche **wildcard**, che possono sostituire uno o più generici caratteri. Usando il comando **DEL P*** cancelleremo, ad esempio, tutti i file il cui nome comincia per P ed è seguito da una sequenza qualsiasi di caratteri.
Nel sistema operativo *Unix*, l'interprete dei comandi prende il nome di **shell** ed è simile a quello dell'MS-DOS, ma in grado di riconoscere molti più vocaboli con una sintassi più raffinata.
I circa 300 comandi a disposizione sono composti da pochissimi caratteri (a volte uno solo) con molte opzioni che consentono una comunicazione estremamente *rapida* e *flessibile*; per contro il linguaggio risulta *molto ostico* ai non esperti.
L'interprete Unix, diversamente da quello MS-DOS, distingue caratteri maiuscoli e minuscoli (viene detto *case sensitive*) e quindi la sintassi è ancora più ricca e complessa.
La struttura di un generico comando di Unix è la seguente:

comando [-opzioni] [argomenti]

che risulta assai simile a quella MS-DOS, da cui si differenzia per l'uso del meno al posto della barra e per l'ordine invertito tra opzioni e parametri.
I principali comandi sono:

- **ls** *directory*: mostra il contenuto di una directory.
- **cp** *File1 File2*: copia il file1 sul file2.
- **who**: mostra gli utenti collegati al sistema.
- **cat** *File1 File2*: concatena il file2 al file1.