Stadi evolutivi dei sistemi di elaborazione

Sistemi isolati

Stand alone. Elaborazione di tipo *batch*. Nessuna comunicazione diretta utente-macchina.

Sistemi centralizzati

Elaboratori di grosse dimensioni. Accesso remoto tramite *terminali passivi* (non intelligenti) collegati via linea telefonica. Tecniche di *time-sharing*.

Sistemi decentrati - Minielaboratori

Di nuovo decentralizzazione. Notevole potenza di calcolo ad un prezzo relativamente basso. "Rivolta dei mini" per superare la lentezza e la rigidità del servizio centralizzato.

Sistemi distribuiti

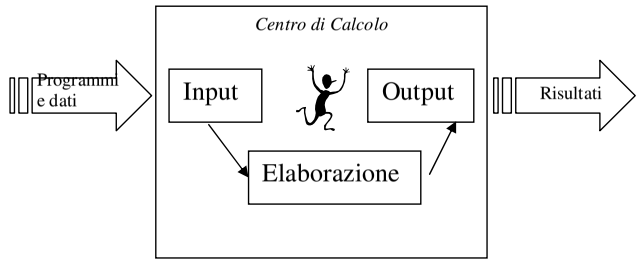
Concepiti come un insieme di unità dotate di capacità operativa autonoma ed al tempo stesso in grado di scambiare mutuamente dati e risorse attraverso una rete di comunicazione.

Intelligenza distribuita in un'ottica di cooperazione ed integrazione delle risorse.

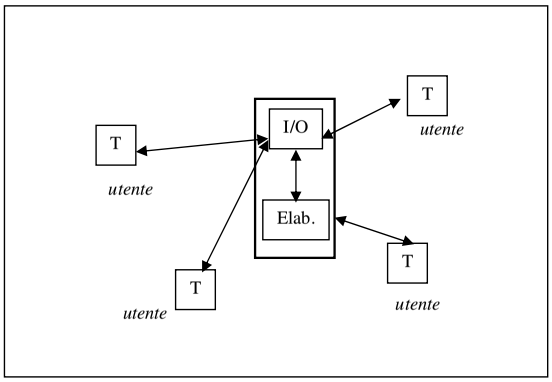
5

Stadi evolutivi e modalità d'uso dei sistemi

- il sistema *isolato* ('50-'60):
  - *batch*

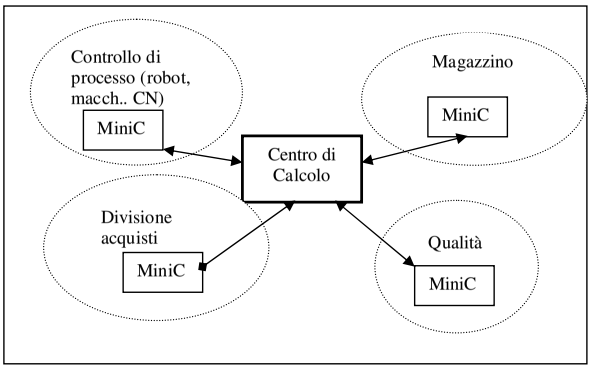


- il sistema *centralizzato* ('60-'70):
  - *remote job entry*
  - *teleprocessing*
  - *time-sharing*
  - *multiprocessing*
- facilitazione accessi
- distribuzione esperti



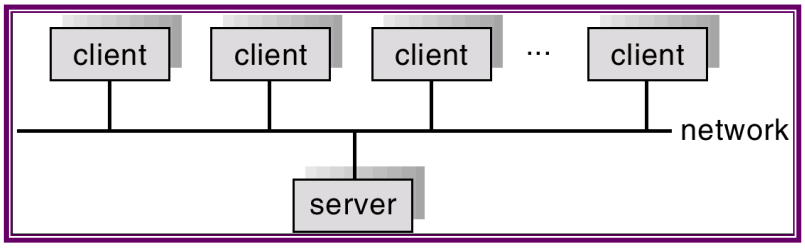
Stadi evolutivi e modalità d'uso dei sistemi

- i sistemi *decentrati* ('70 -):
  - *minicalcolatore*
  - *controllo real-time*
  - *data logging*



- autonomia locale
- disponibilità sw
- gestione dati non integrata

- i sistemi *distribuiti* ('80 -):
  - *multiprocessori*
  - *reti locali*
- *reti geografiche*
- - *basi di dati distribuite*
  - *protocolli di comunicazione*



Sistema	Rete	Distanza (m)	Velocità (bit/sec)
Distribuzione funzionale	integrata	1-10	10^7-10^{10}
Distribuzione locale	privata	10-100	10^6-10^9
Distribuzione geografica	privata/pubblica	>1000	10^3-10^7

- sottorete di comunicazione
  commutazione
  - di circuito
  - di messaggio
  - di pacchetto

Sistema di elaborazione distribuito

- E' costituito da nodi tra loro collegati in ciascuno dei quali sono presenti capacità di:
  - elaborazione
  - memorizzazione
  - comunicazione

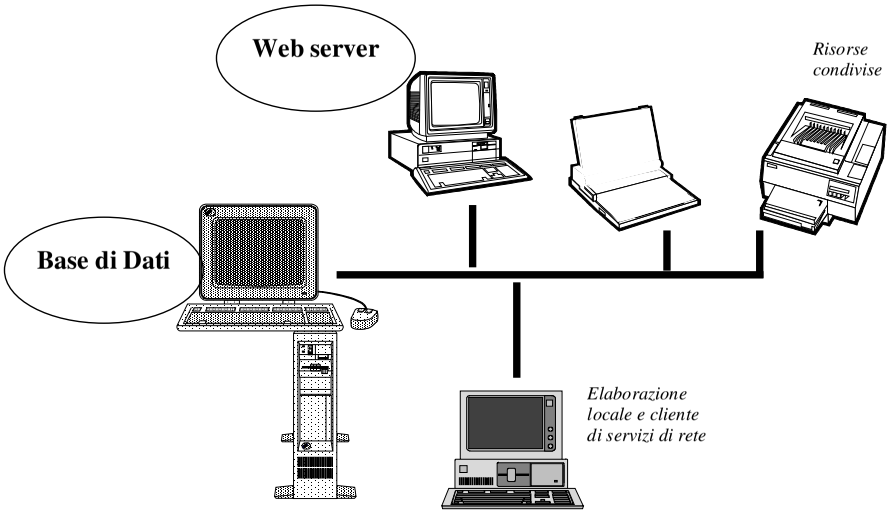
Vantaggi:

- *Tolleranza al guasto*
  Il verificarsi di un guasto non provoca l'arresto del sistema, ma solo una riduzione delle sue prestazioni. Si ha una minore vulnerabilità rispetto ad evenienze catastrofiche (naturali o dolose).

- *Prestazioni*
  Essendo l'elaborazione *di norma* effettuata nel posto stesso di utilizzazione, si ha un *miglioramento delle prestazioni* (tempo medio di risposta, throughput) rispetto al caso di elaborazione centralizzata.

- *Condivisione*
  La capacità di elaborazione, i programmi ed i dati esistenti nell'intero sistema sono, in linea di principio, *patrimonio comune* di tutti gli utenti.

Sistema di elaborazione distribuito



Evoluzione dei S.O.

I primi calcolatori:

- Sono privi di S. O.
- Il programmatore è anche operatore interattivo ed ha visione diretta della macchina e disponibilità di tutte le sue risorse
- L'accesso da parte di più utenti è ottenuto mediante meccanismi di prenotazione
- problemi: complessità operazioni, inefficienza e rigidità della prenotazione

Prima generazione ('50-'60):

- *virtualizzazione dell'I/O*, librerie di controllo dei device
- separazione del programmatore dalla macchina tramite l'*operatore*
- problemi: *debug* (dump), *set-up dei job*

- riduzione dei tempi di set-up tramite:
  - gestione dei job di tipo *batch*
  - gestione periferiche con tecniche di *spooling*
- *automatic job sequencing* tramite *monitor residente*
-

=> nasce il S.O. come stratificazione successiva di funzioni volte ad aumentare l'efficienza e la semplicità d'uso della macchina

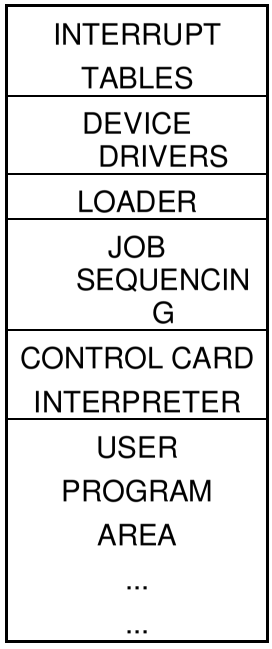
Monitor residente

- lavoro da console, caricamento manuale del programma in memoria: problema principale *job setup*
- necessita` di ridurre *idle time* => *automatic job sequencing*

monitor residente in memoria:

Il monitor richiede:

- schede di controllo (JCL) per interpretare le richieste dell'utente
- loader per compilatori, assemblatori, programma utente
- device driver, usate dal control card interpreter e dal loader per l'I/O, ma rese disponibili anche ai programmi applicativi tramite linking



Evoluzione dei S.O.

Seconda generazione ('60-'65):

- indipendenza tra programmi e dispositivi usati (*logical I/O*)
- parallelizzazione degli utenti tramite *multiprogrammazione* e *time-sharing*

Terza generazione ('65-'75):

- S.O. unico per una famiglia di elaboratori
- risorse virtuali (memoria)
- sistemi multifunzione (scientifico, gestionale)
- linguaggi di comando complessi

Quarta generazione ('75-):

- sistemi a macchine virtuali
- sistemi multiprocessore e distribuiti
- interfacce amichevoli per l'utente

Tecniche di gestione di un sistema di calcolo

- monoprogrammazione

- multiprogrammazione

Sistema monoprogrammato

- Gestisce *in modo sequenziale* nel tempo i diversi programmi.

  L'inizio della esecuzione di un programma avviene solamente *dopo il completamento del programma precedente*.

- Tutte le risorse hw e sw del sistema sono dedicate ad *un solo programma*.

- Bassa utilizzazione delle risorse:

$$\text{utilizzazione CPU} = \frac{T_p}{T_t}$$

  Tp = tempo dedicato dalla CPU alla esecuzione del programma

  Tt = tempo totale di permanenza nel sistema del programma

- throughput = numero di programmi eseguiti per unità di tempo

Sistema multiprogrammato

- Gestisce *simultaneamente più programmi indipendenti*, nel senso che ciascuno di essi può iniziare o proseguire l'elaborazione prima che un altro sia terminato.

- Le risorse risultano *meglio utilizzate* in quanto si riducono i tempi morti.

- Cresce la *complessità* del Sistema Operativo

- Occorrono algoritmi per la *gestione delle risorse* (CPU, memoria, I/O), nascono *problemi di protezione*, etc.

Gestione Batch

- Significa raggruppare i lavori o i programmi *in lotti* per conseguire una *maggiore utilizzazione delle risorse*, cioè un throughput più elevato

- Un concetto che si è evoluto nel tempo:
  - l'operatore raggruppa i programmi in lotti e li immette in tale forma nel sistema per un più razionale utilizzo delle risorse
  - i programmi sono inseriti *nella memoria di massa* e successivamente elaborati in multiprogrammazione

- Nel caso di multiprogrammazione il S.O. deve provvedere algoritmi per la scelta di quell'insieme di programmi che, in esecuzione contemporanea, massimizza il throughput

- La gestione batch può essere *locale* (unità centrale direttamente collegata ai dispositivi di I/O) o *remota* (è presente una trasmissione dei job e dei risultati ed eventualmente una memorizzazione intermedia)

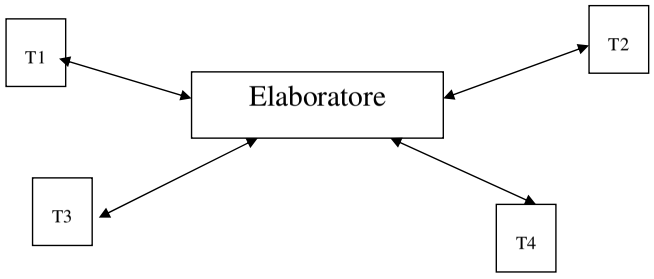
Time sharing

- L'elaboratore serve "simultaneamente" una pluralità di utenti, dotati di terminali, dedicando a ciascuno di essi tutte le risorse del sistema *per quanti fissati di tempo*

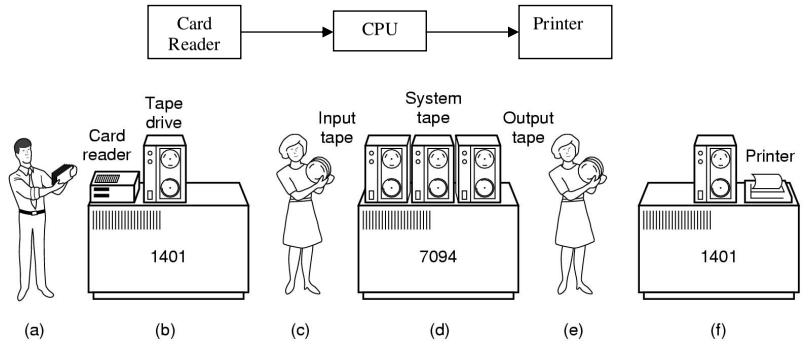
- Migliora i *tempi di risposta* (turn-around time) ma peggiora l'utilizzazione delle risorse

- Può essere presente sia in sistemi monoprogrammati che multiprogrammati

- Normalmente una modalità di gestione time-sharing è adottata nei sistemi *conversazionali*, in cui più utenti contemporaneamente "colloquiano" con il sistema



Evoluzione dei sistemi batch

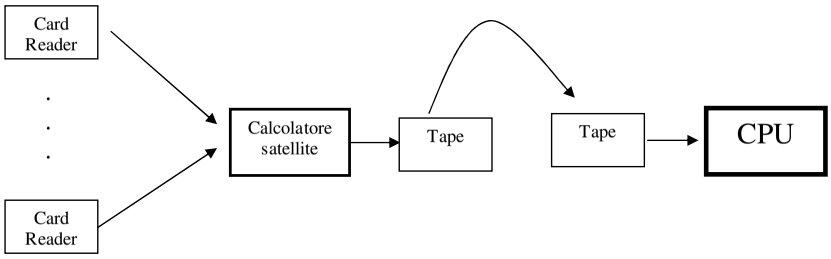


- operazioni di I/O *fuori-linea*:
  - il calcolatore principale non è più rallentato da periferiche lente
  - trasparente ai programmi applicativi

- calcolatori *satellite*:
  - con il compito di scrivere e leggere nastri
  - di potenza ridotta rispetto a quello centrale
  - primo esempio di sistema *multi-computer*

Evoluzione dei sistemi batch

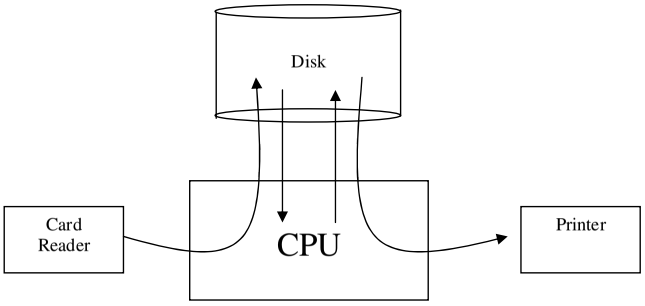
- Vantaggio ulteriore delle operazioni *fuori-linea* è la possibilità di utilizzare più *lettori di scheda* collegati con una stessa unità nastro in ingresso e più *stampanti* collegate con una stessa unità nastro in uscita



- Non ci può essere accesso contemporaneo da parte della CPU e del lettore di schede o della stampante allo stesso nastro

Spooling

- SPOOL: acronimo da Simultaneous Peripheral Operation On Line (NASA Houston Computation Center)



- Per accrescere la velocità di esecuzione dei programmi conviene utilizzare la memoria a disco per simulare i dispositivi di I/O: il disco viene usato come un buffer di grosse dimensioni a cui accedono sia il lettore di schede (e la stampante) che la CPU.

- I trasferimenti lettore di schede-memoria di massa (spool-in) e memoria di massa-stampante (spool-out) sono effettuati da appositi programmi detti *di spooling*.

- Compare il concetto di insieme di programmi pronti per l'esecuzione su disco. Il S.O. può scegliere quale programma mettere in esecuzione (a differenza del caso dei nastri magnetici e delle schede).

Richiami e notazione

- Un sistema di calcolo è costituito da una o più CPU, memoria principale, memoria secondaria, dispositivi di I/O

- Memoria: un vettore M[0:n-1] (es. n=32MB) a cui la CPU accede tramite le istruzioni "LOAD N1" e "STORE N1"
- Una CPU contiene una serie di registri che referenzia per nome o in maniera implicita. Due registri speciali sono: IR (instruction register) e PC (program counter)

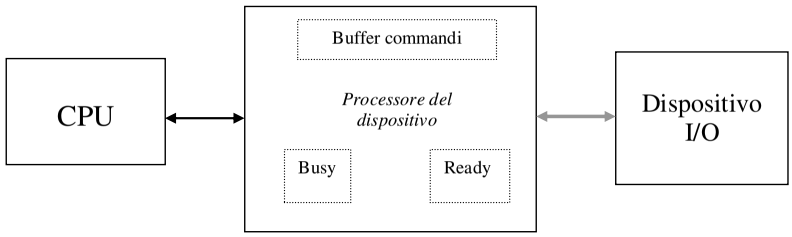
- Ciclo di Fetch-Execute:

  La CPU ripete continuamente, *in hardware*, il ciclo:

```
repeat
        IR := M[PC]
        PC := PC + 1
        execute(istruz. in IR)
until  CPU halt
```

  (L'execute di un jump determina la modifica del PC, etc.)

Un modello di interazione tra CPU e dispositivi di I/O



1. La CPU inserisce un comando nel buffer ed attiva il processor del dispositivo mettendo ad 1 il flag BUSY
2. Il processor del dispositivo, se non sta eseguendo un comando, ispeziona continuamente il flag BUSY
3. Non appena lo trova con il valore ad 1, lo azzera ed inizia l'esecuzione del comando contenuto nel buffer
4. Al termine dell'esecuzione il flag READY viene messo a 1 per avvertire la CPU che il comando è stato eseguito
5. La CPU azzera il flag READY e inserisce un nuovo comando nel buffer

Interazione tra CPU e dispositivi di I/O

- Nel sistema operativo considerato **la CPU non esegue nessun altro lavoro** durante l'attesa per il completamento di un comando ((*fase di busy waiting*)

- Il modello di comunicazione puo` essere espresso più precisamente come segue:

CPU

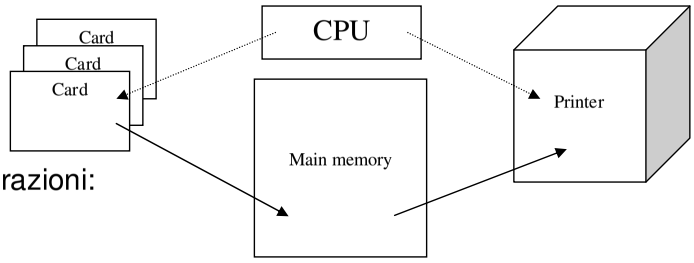
```
repeat
      invio comando;
      BUSY := 1;
      repeat  esamina READY      // fase di
            until  READY = 1     // busy waiting
      READY := 0;
until  transmission_completed
```

I/O Processor

```
repeat
      repeat  esamina BUSY
            until  BUSY = 1
      BUSY := 0;
      <esegui comando>
READY := 1;
until  device_halt
```

- Generalmente il tempo dedicato all'I/O costituisce una parte fondamentale del *tempo complessivo di esecuzione di un programma* (cioe` del tempo che intercorre tra la lettura del programma e la stampa degli ultimi risultati).
- Le prestazioni di un sistema di calcolo possono essere notevolmente migliorate riducendo il tempo dedicato alle attivita` di I/O.

Un semplice S.O.



- Sequenza di operazioni:

repeat
leggi il pacco di schede
compila
carica
esegui
stampa i risultati
until il calcolatore si ferma

- Compito del S.O. e` controllare la sequenza di passi e *gestire i dispositivi di I/O*.

- Trasforma il calcolatore in una *macchina virtuale* capace di compilare ed eseguire una sequenza di programmi.

Gestione I/O

- CR e LP possono essere in ogni istante in uno dei tre *stati*:
LIBERO, PRONTO, OCCUPATO

- Lo stato di CR dipende da:
interruttore, comando, pacco di schede (deck)

- Lo stato di LP dipende da:
interruttore, comando

LP	start	com	stato
0	0	0	LIBERO
1	0	1	LIBERO
2	1	0	PRONTO
3	1	1	OCCUP.

CR	start	deck	com	stato
0...3	0	0,1	0,1	LIBERO
4	1	0	0	LIBERO
5	1	0	1	LIBERO
6	1	1	0	PRONTO
7	1	1	1	OCCUP.

Programmi di controllo I/O

a) Lettura di un pacco di schede

INDCR = indirizzo dell'area di memoria riservata, destinata a contenere le informazioni sulle schede

SL = contatore del numero di schede lette

IC2 = indirizzo corrente dell'area di memoria

programma di controllo CR:

```
fissa INDCR
SL := 0
IC2 := INDCR
attendi fino a che CR diventa PRONTO
repeat
    invia comando (IC2)  /* leggi scheda e metti in */
    SL := SL + 1
    calcola il nuovo indirizzo IC2
    attendi mentre CR e` OCCUPATO
until CR diventa LIBERO
```


Programmi di controllo I/O

b) Stampa di linee

INDLP = indirizzo di inizio dell'area di memoria contenente le linee da stampare

LS = contatore linee da stampare

IC1 = indirizzo corrente area di memoria

programma di controllo LP:

```
IC1 := INDLP
repeat
    attendi fino a che LP PRONTA
    invia comando(IC1)   /* stampa linea da */
    LS := LS - 1
    calcola nuovo indirizzo IC1
until LS = 0
```

- Osservazioni:

  - In entrambi i programmi compaiono delle *fasi di attesa* che dipendono dalla *diversa velocita` della CPU* che esegue il programma di controllo *nei confronti dei dispositivi*.

  - Si e` introdotta una *forma di sincronizzazione* tra due dispositivi di per se` *asincroni*.

Programmi di controllo I/O - Prestazioni del S.O.

- La notevole differenza di velocita` tra CPU e dispositivi periferici fa si` che i programmi di controllo trascorrano la maggior parte del loro tempo in *attesa*.
- I tempi legati all'accesso alla memoria principale per depositare i caratteri della scheda (80) o prelevare i caratteri della linea (120) (a cura dei processori dei dispositivi) sono trascurabili
- E' l'attesa dell'*esecuzione fisica* del comando da parte del dispositivo che rallenta i programmi di controllo di I/O del S.O.

- Esempio: LP

	istruzioni macchina
inizializzazione	2
attesa fino a che LP pronta	?
invia comando	€ 5
decrementa LS	2
calcola nuovo indirizzo	5
test su LS	2

N.B.: tecnologia 1976 (oggi: processor clock > 1 GHZ)

- Il tempo di esecuzione medio di una istruzione macchina, assumendo un tempo di ciclo di 1 usec, e` circa 2,5 - 3
- Il controllo della stampa di una linea richiede circa 45 usec
- Velocita` di stampa: 20 linee / secondo
- Tempo di stampa di una linea: 1/20 sec = 50000 usec

Prestazioni

- Il tempo necessario per eseguire n programmi è:

$$t = I + CLE + O$$

ove: I = somma dei tempi di input
CLE = somma dei tempi per compile, load, execute
O = somma dei tempi di output

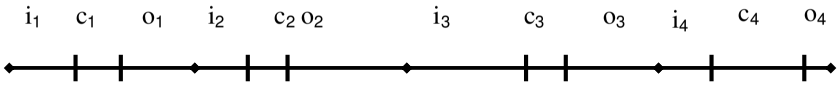
- Il throughput è:

$$\text{throughput} = \frac{n}{t} = \frac{n}{I + CLE + O}$$

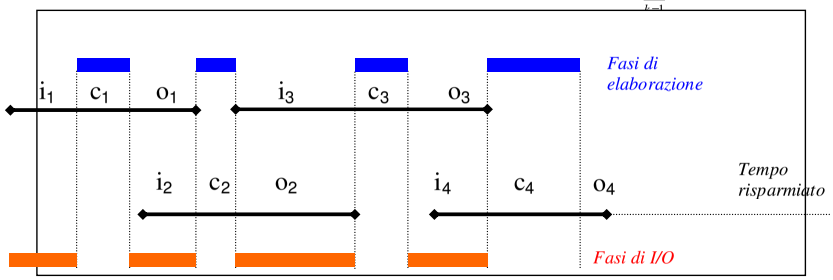
- Per migliorare il throughput occorre *sovrapporre* le tre fasi I, CLE, O nel tempo

- Tale sovrapposizione richiede che i dispositivi periferici e la CPU operino in modo *il più possibile indipendente*.

Sovrapposizione tra ingresso e uscita



- throughput di un sistema seriale: $\frac{n}{t} = \frac{n}{CLE + I + O} = \frac{n}{CLE + \sum_{k=1}^{n} (i_k + o_k)}$



E' una prima soluzione parziale

- throughput di un sistema con sovrapposizione di I/O:

$m_k = \max(I_{k+1}, O_k)$ k = 1,2,...,n-1 tempo di I/O: $M = i_1 + \sum_{k=1}^{n-1} m_k + o_n$

throughput: $\frac{n}{t} = \frac{n}{CLE + M}$

Programma di controllo I/O

- Per raggiungere l'obiettivo della sovrapposizione dell'ingresso di un programma con l'uscita di un altro occorre *un unico programma di controllo di I/O*, IOC (input-output control).

- IOC invia comandi ad ogni dispositivo PRONTO ed attende solo quando tutti i dispositivi sono OCCUPATI.

- Il programma riceve come dati di ingresso il numero di linee da stampare e l'indirizzo di memoria in cui sono contenute.

- Il programma termina quando tutte le linee sono stampate e CR è vuoto (LIBERO).

- A differenza del caso precedente, il programma non attende se uno dei dispositivi è pronto.

Programma IOC

INDLP = indirizzo di inizio area di memoria contenente le linee da stampare

INDCR = indirizzo di inizio area di memoria in cui inserire le schede dati

IC1 = indirizzo corrente relativo a INDLP

IC2 = indirizzo corrente relativo a INDCR

```
programma IOC:
        fissa INDCR
        IC1 := INDLP, IC2 := INDCR
        SL := 0
        repeat
            attendi mentre CR and LP OCCUPATE
            if CR PRONTO then
                invia comando (IC2)
                SL := SL + 1
                calcola il nuovo indirizzo IC2
            fi
            if LP PRONTO then
                invia comando(IC1)
                LS := LS - 1
                calcola nuovo indirizzo IC1
            fi
        until LS = 0 and CR LIBERO
```


Funzionamento con sovrapposizione delle attività di I/0

- Programma di controllo del sistema di calcolo da parte del S.O.:

```
LS := 0
repeat
        fase di I/O
        fase di elaborazione
until halt
```

- Dove la fase di elaborazione è la seguente:

```
if SL > 0 then compile; load; execute fi
```

- Osservazioni:
  - Le due fasi, I/O ed elaborazione, sono sequenzializzate
  - La fase di attesa della CPU per il completamento delle operazioni di I/O si è ridotta ma è comunque presente (si ha attesa quando entrambi i dispositivi sono occupati)
  - Si tratta inoltre di una attesa attiva (*busy waiting*) che disturba l'accesso dei processori dei dispositivi di I/O alla memoria
  - Per migliorare ulteriormente le prestazioni del sistema è necessario sovrapporre le fasi di I/O e di elaborazione

Sovrapposizione attività CPU e I/O

- Il coinvolgimento della CPU nelle operazioni di I/O è modesto. Per migliorare le prestazioni occorre che durante le fasi di attesa dei programmi di controllo (in cui viene completata l'esecuzione degli specifici comandi di I/O), la CPU possa eseguire altri programmi. Occorre cioè *sovrapporre anche la attività della CPU con le operazioni di I/O*.

- Nello schema proposto (programma IOC) la sovrapposizione non è possibile perchè la CPU *si dedica* alla esecuzione dei programmi di controllo I/O, rimanendo comunque impegnata (in busy waiting) per l'intera durata della più lunga tra le operazioni di I/O.

- La gestione delle operazioni di I/O richiede peraltro *il contributo* della CPU per l'invio dei comandi ai dispositivi.

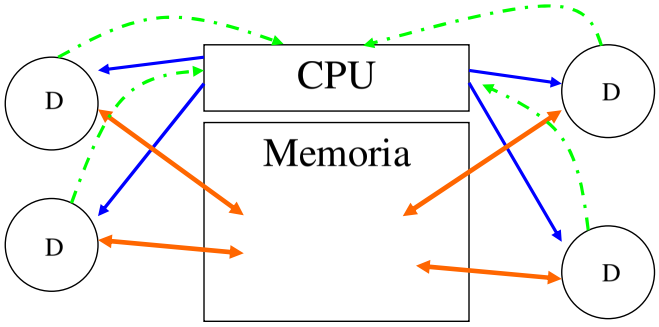
- La sovrapposizione potrebbe essere ottenuto inserendo nei programmi, ad intervalli regolari, la richiesta di esecuzione dei programmi di controllo. Questa soluzione è *inaccettabile* perchè fa ricadere sul programmatore un compito del S.O.

- La soluzione si ha attraverso il concetto di interruzione, realizzato tramite hardware.

Interruzione da dispositivo

- E' un segnale hardware inviato da un dispositivo di I/O alla CPU per indicare che un comando è stato eseguito

dati
comandi
interruzioni



- Ciclo base di Fetch-Execute *in assenza di interruzioni*:

```
repeat
        IR := M[PC]
        PC := PC + 1
        execute(istruz. in IR)
until  CPU halt
```

Modello di comportamento dell'interruzione da dispositivo

- Si può supporre che sia contenuto nella CPU un *vettore di bit* (registro di richieste di interruzione - IRR) in cui ogni bit *memorizza* l'interruzione *di un particolare dispositivo*.

- Il ciclo di Fetch-Execute diventa:

```
repeat
        if  IRR = 0 then
                IR := M[PC]
                PC := PC + 1
        else  IR := M[0] fi
        execute(istruz. in IR)
until  CPU halt
```

- M[0] indirizzo in memoria di una procedura, chiamata *interrupt routine*, la cui azione è la seguente:

```
interrupt procedure:
 begin
        salva lo stato del programma interrotto;
        x := indice del bit che ha causato l'interruzione;
        IRR[x] := 0;        /* azzera il bit di richiesta x-esimo */
        chiama il programma di controllo x-esimo;
        ripristina lo stato e continua il programma interrotto;
 end
```


Interrupt da timer e gestione a polling dell'I/O

- In alternativa alla gestione ad interrupt dei singoli dispositivi, è possibile utilizzare un unico *interrupt periodico* generato da un timer hardware (*real-time clock*) con frequenza programmabile dal S.O.

- La routine di servizio dell'interrupt andrà ad esaminare le *condizioni di attivazione* dei programmi di controllo I/O, provvedendo eventualmente ad effettuarne la chiamata.

- Le variabili dei programmi di controllo I/O *non sono inizializzate ad ogni attivazione* del programma, ma vengono trattate come variabili globali. Ad esempio il programma di controllo del CR quando attivato usa il valore corrente di SL.

- Ogni programma di controllo I/O può essere scritto ed eseguito indipendentemente dagli altri, pur mantenendo la sovrapposizione tra elaborazione e gestione dell'I/O.

```
timer interrupt procedure:
 begin
        salva lo stato del programma interrotto;
        if  CR READY then  call CR_control fi;
        if  LP READY and  LS > 0 then  call LP_control fi;
        ripristina lo stato e continua il programma interrotto;
 end
```

Programmi di controllo I/O (con timer interrupt)

```
CR_control:
        begin
                invia comando (IC2)
                SL := SL + 1
                calcola il nuovo indirizzo IC2
        end

LP_control:
        begin
                invia comando(IC1)
                LS := LS - 1
                calcola nuovo indirizzo IC1
        end
```

- Lo schema che fa uso di un unico interrupt da timer è particolarmente *semplice*: è sufficiente un unico livello di interruzione.

- Tuttavia è meno *efficiente* rispetto all'impiego di interrupt da dispositivi:
  - Può accadere che nessuna delle condizioni di attivazione sia verificata. Si ha pertanto un *overhead* dovuto (più che alla valutazione delle condizioni) al salvataggio e ripristino del programma interrotto.
  - Un dispositivo può rimanere in attesa del comando successivo (al più per un periodo del real-time clock).

Interruzioni e S.O.

- E' compito del S.O. gestire le interruzioni tramite le *interrupt routines*.

- Nel semplice modello di S.O. visto, *in cui un solo programma alla volta* è presente in memoria principale, la sovrapposizione tra le attività di I/O e di elaborazione produce uno *scarso beneficio*.

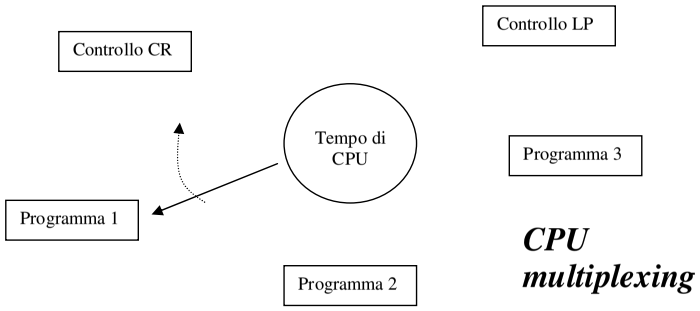
- Entrambe le attività sono relative infatti allo stesso programma, che in genere ammette un grado limitato di parallelismo tra di esse.

- Per avere un guadagno notevole nell'utilizzazione delle risorse occorre avere *più programmi presenti contemporaneamente in memoria* (multiprogrammazione), potendo così sovrapporre elaborazione ed I/O di programmi diversi.

Multiprogrammazione (circa 1965)

- Più programmi sono presenti contemporaneamente in memoria principale.

- Quando uno di essi attende per il completamento di una operazione di I/O, il controllo della CPU può essere assegnato ad un altro:



CPU multiplexing

- Occorre che l'insieme dei programmi presenti contemporaneamente in memoria principale sia scelto in modo da garantire la massima occupazione della CPU (e della memoria).

- Programmi *I/O bound* e *Compute bound*

Interruzioni e Multiprogrammazione

- *La routine di interruzione*, invece di ritornare al programma interrotto, può *scegliere* ora, in base ad un *determinato algoritmo*, a quale programma presente in memoria affidare il controllo della CPU.

- I programmi possono quindi essere attivati come conseguenza di una interruzione.

- Il S.O. deve provvedere alla scelta del programma cui assegnare la CPU sulla base di *algoritmi di scheduling*.

- Per evitare che i programmi *compute bound* mantengano il controllo della CPU per tempi molto più elevato di quelli I/O bound, è necessario introdurre anche un interrupt periodico (timer) che comunque determina l'invocazione dello scheduler.

- Un programma può richiedere l'accesso ad un dispositivo già impegnato (eventualmente nel servizio di una richiesta precedente dello stesso programma).

- Il S.O. deve pertanto *mantenere in una tabella lo stato dei dispositivi*, *sospendere* un programma che intende accedere ad una risorsa già impegnata, mettendo la relativa richiesta in una coda associata al dispositivo, *realizzare algoritmi di gestione per tali risorse*, scegliendo tra i programmi sospesi quello a cui attribuire la risorsa, *proteggere i dati di un programma*, ...

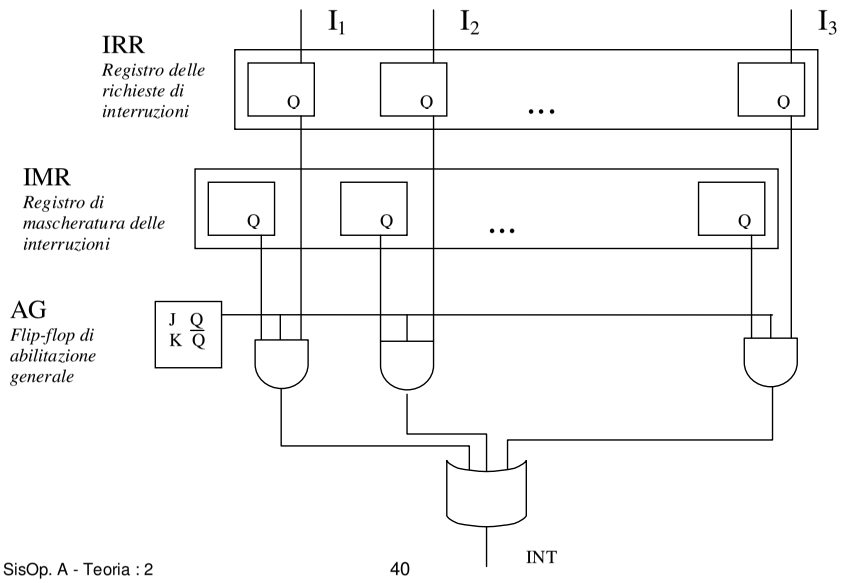
Interruzioni

- interruzioni hardware (asincrone):
  - *device interrupt*, per terminazione di un trasferimento dati o per condizione di errore rilevata dalle o nelle periferiche (es. parità)
  - *timer interrupt*, real-time clock per la misura del tempo
  - *powerfail sense interrupt*, per salvataggio urgente dello stato del sistema

- interruzioni software (sincrone)
  - *eccezioni* o *trap*, per tentativo da parte del programma di compiere azioni con effetti illegali, ad es. divisione per zero, overflow, opcode illegale (inesistente o privilegiato), violazione della protezione della memoria, etc. (rilevazione a livello hardware)
  - *programmate* o *supervisory call (svc)*, (es. INT n), per utilizzare le funzioni del S.O. o per trasferire il controllo al S.O.

Sistema delle Interruzioni

- A ciascuna causa di interruzione è associata un'azione che verrà effettuata dal programma di risposta alle interruzioni.
- Una causa non produce di per sè un'interruzione ma solo una *richiesta di interruzione*. Affinchè alla richiesta segua effettivamente una interruzione è necessario che la causa di interruzione sia *abilitata*.
- Abilitazione e disabilitazione possono essere selettive o globali.



Sistema delle Interruzioni

- Ciascun evento "Ii" causa di interruzione è memorizzato in un flip-flop IRRi ed abilitato da IMRi ed AG

  IRR registro di richiesta di interruzione

  IMR registro di maschera di interruzione

  AG flip-flop di abilitazione/disabilitazione generale delle interruzioni

- L'interruzione "i" si manifesta se:
  - AG = 1 (il sistema di interruzione è abilitato)
  - IRRi = 1 (si è presentata la causa di interruzione "Ii")
  - IMRi = 1 (l'interruzione "i" è abilitata nel registro di maschera)

- IMR e AG possono essere modificati mediante istruzioni speciali (IE, DI) o generali (out(io_address, value))

Processo delle Interruzioni

- Al verificarsi di una interruzione occorre:

  a) salvare tutte le informazioni necessarie per la ripresa del programma interrotto

  b) individuare la causa dell'interruzione

  c) eseguire le azioni richieste (servizio dell'interruzione)

  d) ripristinare lo stato del programma interrotto e riavviarlo

- A seconda della sofisticazione dell'hardware queste azioni impegnano in misura maggiore o minore il software (overhead).

a) - L'hw provvede in genere a salvare PC e PSW, mentre i registri generali vengono tipicamente salvati in software. E' necessario che il salvataggio dei registri avvenga ad interrupt disabilitati.

   - Normalmente l'hw cede il controllo alla routine di servizio con interrupt disabilitati (AG=0). In caso contrario il sw deve disabilitare gli interrupt.

b) - L'hw può trasferire il controllo sempre al medesimo programma di risposta, che provvederà quindi ad individuare la causa dell'interruzione tramite *skip chain*, oppure direttamente a programmi di risposta separati.

c) - Durante il servizio della interruzione il sistema delle interruzioni può essere riabilitato (AG), eventualmente selettivamente (IMR), purchè sia possibile il *nesting* delle interruzioni tramite *stack*.

d) - Il ripristino dei registri deve avvenire ad interrupt disabilitati. Un'apposita istruzione di ritorno dall'interrupt (RTI) ripristina i registri salvati via hw e riabilita le interruzioni.

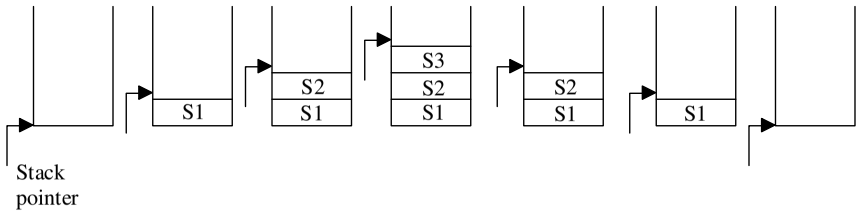
Livelli di priorità

- I diversi tipi di interruzione possono suggerire una gestione con diversi livelli di priorità.

- Durante il servizio di un interrupt di livello L le interruzioni di livello K<= L restano disabilitate, mentre le altre possono essere abilitate. Questa gestione può essere realizzata o agevolata dall'hw (programmable interrupt controller) o per via sw.

- Se all'atto del ritorno esistono più richieste pendenti si serve quella di livello più elevato.

Salvataggio dello stato tramite pila (stack)



S1: stato del processore salvato all'arrivo della prima interruzione

S2: stato del processore relativo all'esecuzione del programma di risposta della prima interruzione salvato all'arrivo della seconda interruzione

S3: stato del processore relativo all'esecuzione del programma di risposta della seconda interruzione salvato all'arrivo della terza interruzione

- Salvataggio, ripristino, gestione SP non interrompibili (hw o interrupt disabilitati)
- Programma di risposta ad interruzione di livello L:
  - salva lo stato del processore nella stack e modifica SP
  - abilita interruzioni di livello K>L
  - esegue programma di risposta
  - disabilita le interruzioni di tutti i livelli
  - ripristina lo stato del processore modificando SP
  - esegue ritorno da interrupt

Processi

- L'evoluzione dei S.O., guidata da esigenze di efficienza, ha portato alla presenza in memoria centrale di *più programmi* in esecuzione.

- Emergono nuove *funzionalità* richieste al S.O., quali la gestione dei programmi stessi e la protezione dalla mutua interferenza.

- Un nuovo *punto di vista concettuale*:

  - i programmi di controllo, analogamente ai dispositivi hw, sono largamente indipendenti l'uno dall'altro, ed interagiscono di rado ed in punti ben definiti con altre attività;

  - la CPU viene "trasferita" da un job all'altro (anzichè "ricevere" programmi in ingresso);

  - la specifica sequenza di stati della CPU è scarsamente significativa ed impredicibile a causa degli interrupt; il sistema va concepito in termini di *specifica funzionale* delle elaborazioni, del controllo dei dispositivi, delle strutture dati per lo scambio di informazioni.

Processi

- Un *processo* (o task) è l'unità funzionale in un S.O.

  Un processo è *controllato da un programma* e ha bisogno di un *processore per la esecuzione*.

- Alcuni processi dispongono di un processore *privato* e pertanto sono permanentemente in esecuzione (ad esempio i controllori delle periferiche).

- Altri processi condividono un processore *comune* (la CPU) (ad esempio i processi utente e di sistema).

-Ai processi che non dispongono di un processore privato associamo un *processore virtuale*, in grado di interpretare il linguaggio in cui il programma che controlla il processo è stato scritto.

Processi

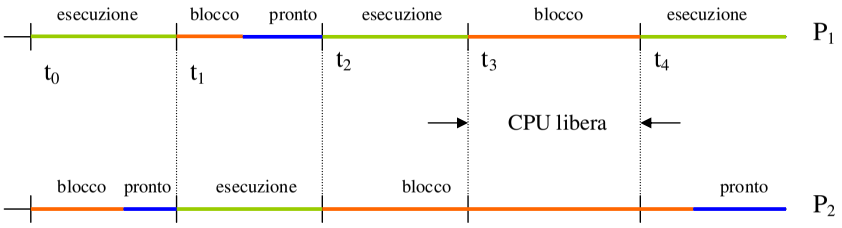
- In un sistema multiprogrammato la CPU esegue alternativamente, in un intervallo di tempo, **sequenze di operazioni appartenenti a programmi diversi** ; nel medesimo intervallo l'esecuzione di un programma può essere **sospesa e ripresa più volte**.

- A differenza dei sistemi uniprogrammati, nei sistemi multiprogrammati occorre distinguere tra l'attività della CPU e l'esecuzione di un particolare programma.

- Il termin **processo** viene usato per indicare **l'attività svolta dalla CPU per l'esecuzione di un particolare programma**.

- In un sistema multiprogrammato **sono presenti contemporaneamente più processi**, di cui **uno solo** in ogni istante può essere in **esecuzione**. Gli altri processi sono sospesi in attesa della disponibilità della CPU o del verificarsi di particolari condizioni che rendano possibile il proseguimento della loro esecuzione (ad es. completamento I/O)

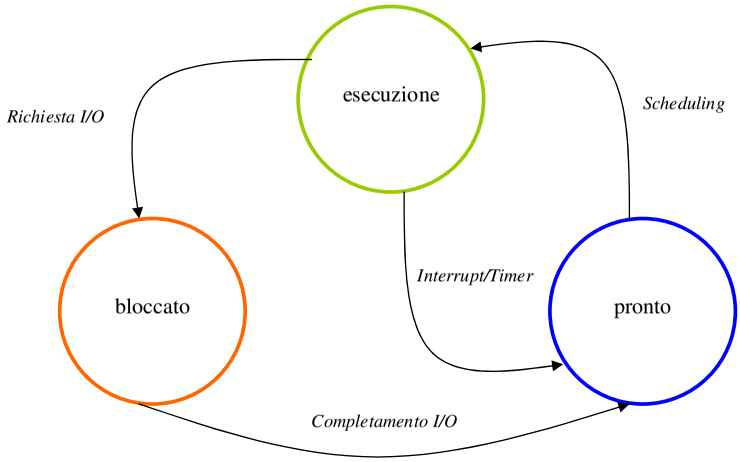
Sistemi multiprogrammati



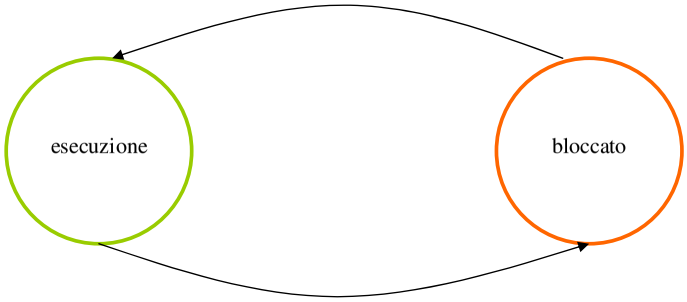
- Una possibile definizione di processo:
  "Attività svolta dalla CPU per l'esecuzione di un determinato programma"

Stato di un processo

- in esecuzione (running)
- bloccato (idle, waiting)
- pronto (ready)



- Se Numero di CPU = Numero di processi:



Gestione dei processi

- Processo => programma in esecuzione
  (es. job batch, programma utente time-shared, task di sistema, etc.)

- Il processo è l'unità di lavoro di un sistema, che consiste di una collezione di processi: *processi del S.O.* che eseguono il codice del sistema, *processi utenti* che eseguono il codice di utente. Possono essere in esecuzione *concorrentemente*.

- Funzioni del S.O. (riferite ai processi):
  - creazione e cancellazione di processi
  - sospensione e ripresa di processi
- strumenti per la sincronizzazione e comunicazione
- strumenti per il trattamento di condizioni di deadlock

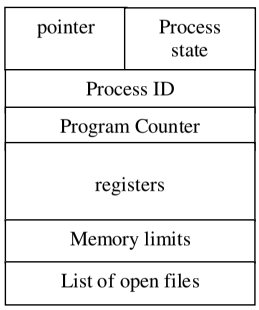
Gestione dei processi

- La possibilità che la CPU venga commutata in un qualsiasi istante da un processo ad un altro rende indispensabile ad ogni commutazione salvare tutte le informazioni contenute nei registri della CPU e relative al processo che è stato sospeso (PC, accumulatori, registri indice, etc.)

- Descrittore di processo o Process Control Block (PCB) : area di memoria, mantenuta all'interno del monitor del S.O. , associata al processo e contenente tutte le informazioni proprie del processo

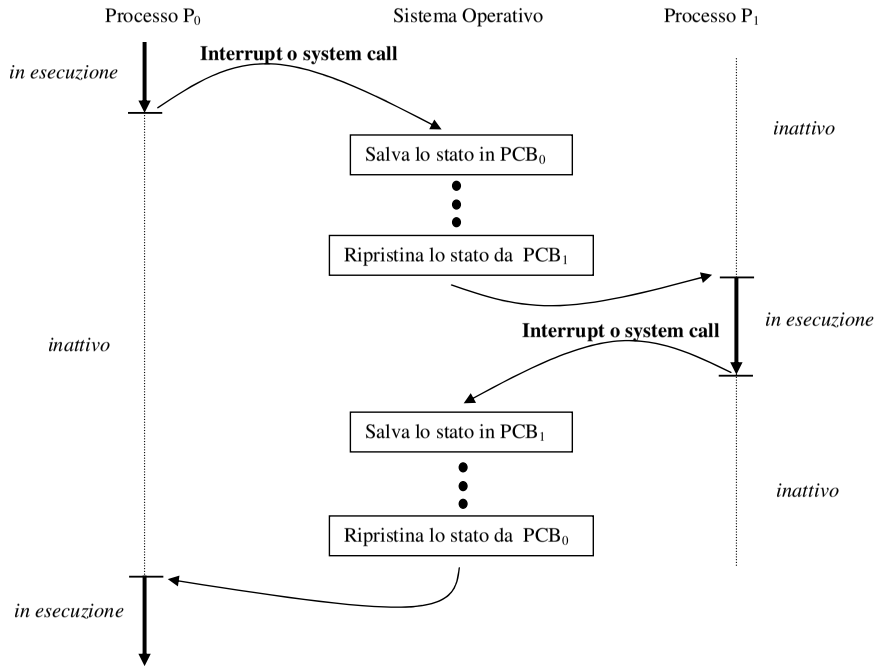
Process Control Block (PCB)

- Stato del processo : pronto, bloccato in esecuzione
- Program Counter (PC)
- Registri della CPU: accumulatori, registri indice, registri general-purpose
- Informazioni per la gestione della memoria : registri base e limite, tabella delle pagine (per il demand paging)
- Informazioni di accounting: ammontare del tempo di CPU o di tempo reale usato, limiti di tempo, numeri di account, etc.
- Informazioni sullo stato di I/O : richieste di I/O non soddisfatte, dispositivi assegnati, lista di file aperti
- Informazioni per lo scheduling della CPU: priorità del processo, puntatori a code di scheduling, parametri di scheduling.



*In **UNIX***
*parte di queste informazioni risiedono nella **user area** : una porzione di memoria nell'immagine del processo che è **accessibile solo in modo kernel***

Commutazione delle CPU tra processi



Algoritmo, Programma, Processo

Algoritmo: Procedimento logico che deve essere seguito per risolvere il problema in esame

Programma: Descrizione dell'algoritmo tramite un opportuno formalismo (*linguaggio di programmazione*) che rende possibile l'esecuzione dell'algoritmo da parte di un particolare elaboratore

Processo (sequenziale): La sequenza di eventi cui da` luogo un elaboratore quando opera sotto il controllo di un particolare programma (evento = esecuzione di una operazione)

Sistema di protezione

- Un processo potrebbe tentare di modificare il programma o i dati di un altro processo o di parte del S.O. stesso.

- Protezione: politiche (cosa) e meccanismi (come) per controllare l'accesso di processi alle risorse del sistema di elaborazione

- Esempi:
  - l'hardware di indirizzamento della memoria assicura che un processo possa operare solo entro il proprio spazio di indirizzi.
  - l'I/O system impedisce l'accesso diretto ai dispositivi, etc.

-All'hardware é affidato il compito di *rilevazione* di errori, come op code illegali o riferimenti in memoria illegali, possibili effetti di errori di programmazione o di comportamenti deliberatamente intrusivi.

- Tali errori vengono *segnalati* e affidati alla *gestione* al S.O. tramite il meccanismo delle trap.

- In presenza di errore o di violazione della protezione il S.O. provvede a terminare il processo, segnala la terminazione anomala ed effettua un dump della memoria.

Sistema di protezione

- Ciascun processo opera in un *dominio di protezione* che specifica le risorse a cui il processo può accedere e le operazioni consentite.

- Diritto di accesso (coppia ordinata <risorsa, diritti>): abilitazione alla esecuzione di una operazione sulla risorsa; un dominio di protezione è una collezione di diritti di accesso.

- Lista degli accessi di un oggetto: operazioni consentite da ciascun dominio.

- Lista delle *capabilities*: lista di oggetti e di operazioni consentite su tali oggetti riferita ad un dominio. Per eseguire una operazione su un oggetto il processo deve specificare la capability (indirizzo protetto mantenuto dal S.O.).

Protezione

- Il sistema di protezione richiede l'esistenza di *piu` modi* di funzionamento della CPU:
  - *supervisor mode* (system mode, monitor mode)
  - *user mode*

- Il passaggio dal modo user al modo supervisor avviene tramite *interruzione* :
  - esterna (asincrona)
  - interna (sincrona, trap), generata da una SVC (SuperVisor Call o System call).

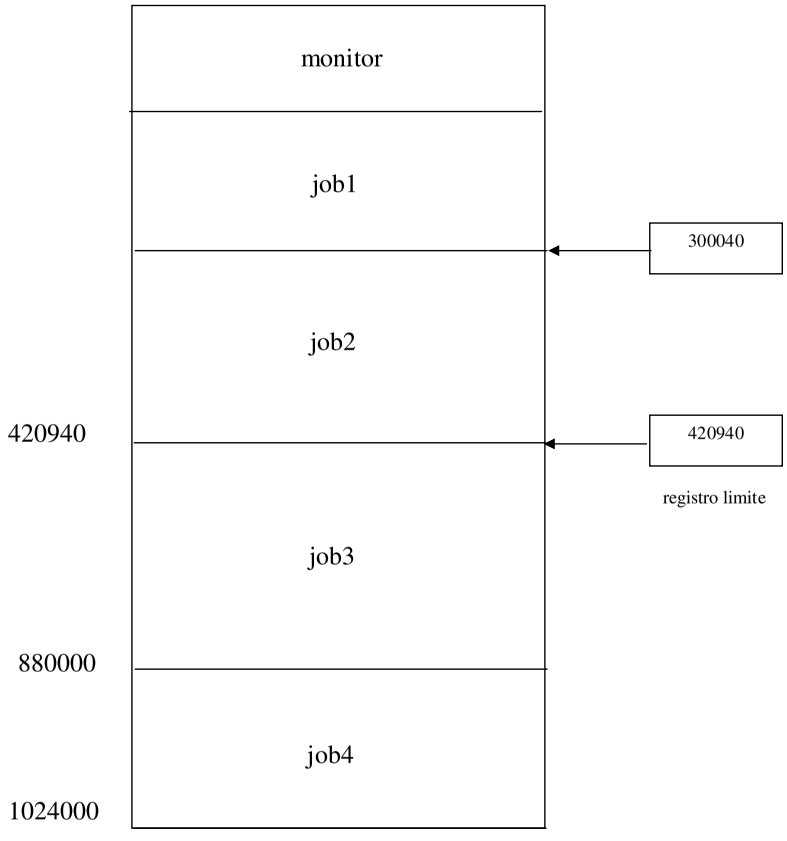
- Il passaggio dal modo supervisor al modo user avviene tramite una istruzione speciale di *cambiamento di modo* eseguita dal S.O. prima della cessione del controllo ad un processo di utente.

- Istruzioni *privilegiate* (eseguite *solo* in system mode):
  - I/O
  - modifica dei registri che delimitano le partizioni logiche di memoria
  - manipolazione del sistema di interruzione
  - cambiamento di modo
- halt

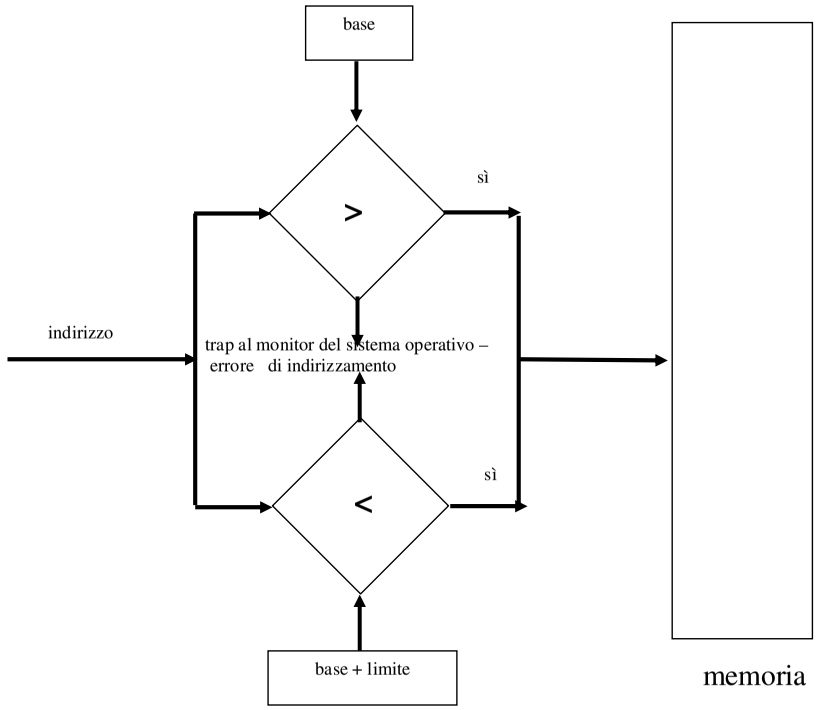
- Protezione della memoria (registri barriera, registri limite)
- Protezione della CPU (time limit)

Protezione della memoria con registri limite

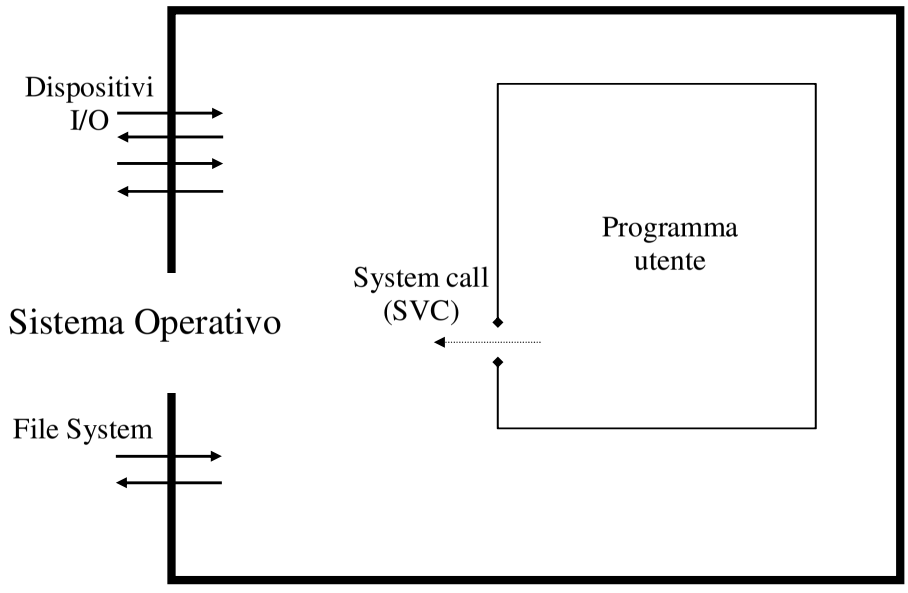
- Prima di mettere un processo in esecuzione, il S.O. ne confina lo spazio logico di memoria mediante registri limite:



Protezione della memoria con registri limite

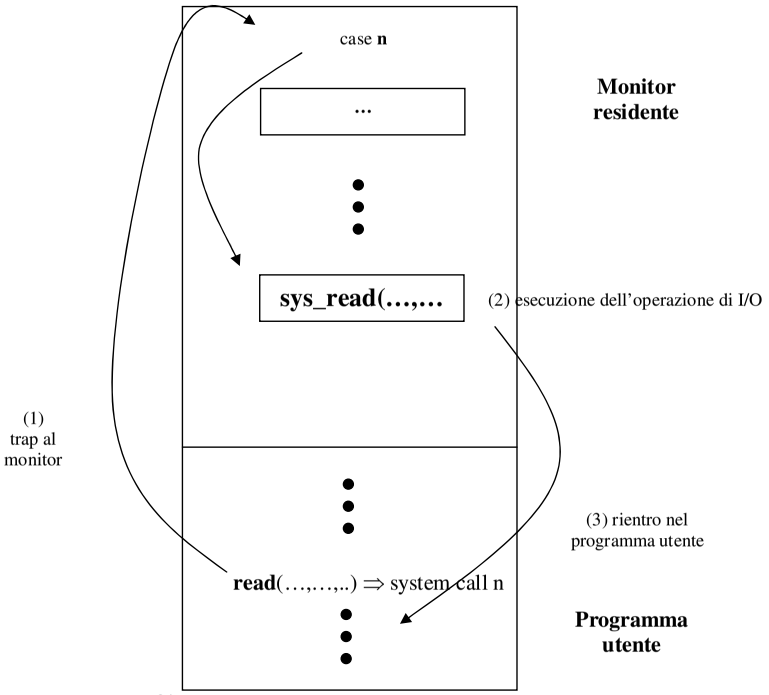


Confinamento del programma utente



Esecuzione delle operazioni di I/O

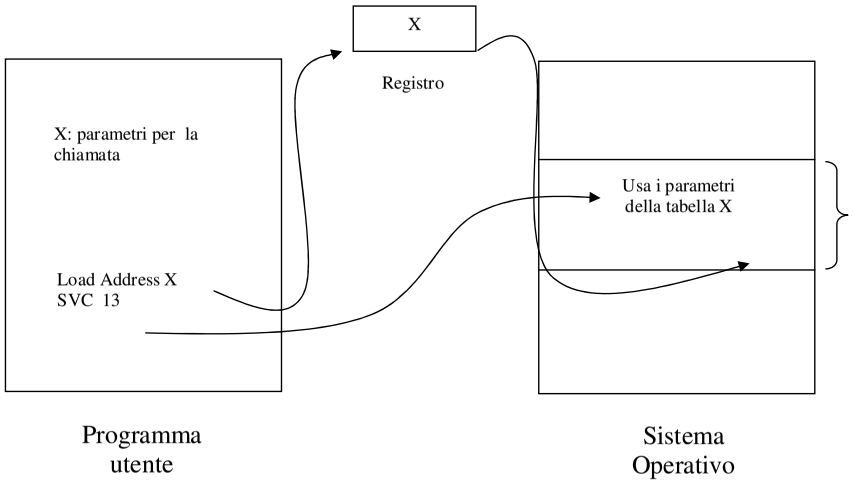
- Le istruzioni di I/O sono privilegiate. Il programma utente richiede *al S.O.*, tramite una *system call,* di eseguire l'operazione di I/O.



System call

- Tipicamente l'operando della istruzione di system call ne specifica il tipo (INT n), mentre il passaggio degli eventuali parametri avviene tramite registri o per indirizzo.

- Passaggio dei parametri mediante tabella:



System call

- Costituiscono *l'interfaccia* tra un programma in esecuzione ed il S.O.

- Istruzioni assembly, procedure chiamabili da linguaggi high-level. Nei linguaggi di alto livello sono tipicamente mascherate dal supporto a tempo di esecuzione.

- Categorie principali di system call:

a) controllo dei processi e dei job
b) manipolazione dei file e dei dispositivi
c) gestione delle informazioni
d) comunicazione

System call

a) controllo dei processi e dei job
- end, abort
- load, execute
- create process, terminate process
- get, set process attributes
- wait for time, wait for event, signal event
- allocate, free memory
- dump, trace

b) manipolazione dei file e dei dispositivi
- create, delete file
- open, close
- read, write, reposition file or device
- get, set file or device attributes
- request, release device

c) gestione delle informazioni
- get, set time or date
- get, set system data
- get, set attributes

d) comunicazione
- create, delete communication connection
- open, close communication
- send, receive message

Programmi di sistema

- Di varia natura:
  - manipolazione dei file (editor, cp, mv, rm, mkdir, ...)
  - Informazioni di stato (date, time, who, df, ...)
  - sviluppo software ed esecuzione (traduttori, linker e loaders, debuggers)
  - comunicazione (rlogin, ftp, mail)
  - applicativi (spreadsheet, latex, ...)
  - *interprete dei comandi*: esegue i comandi realizzati come programmi di sistema speciali (unix)

- I programmi di sistema sono determinanti per la *visione d'utente* del S.O., mentre le system call ne riflettono la struttura interna.

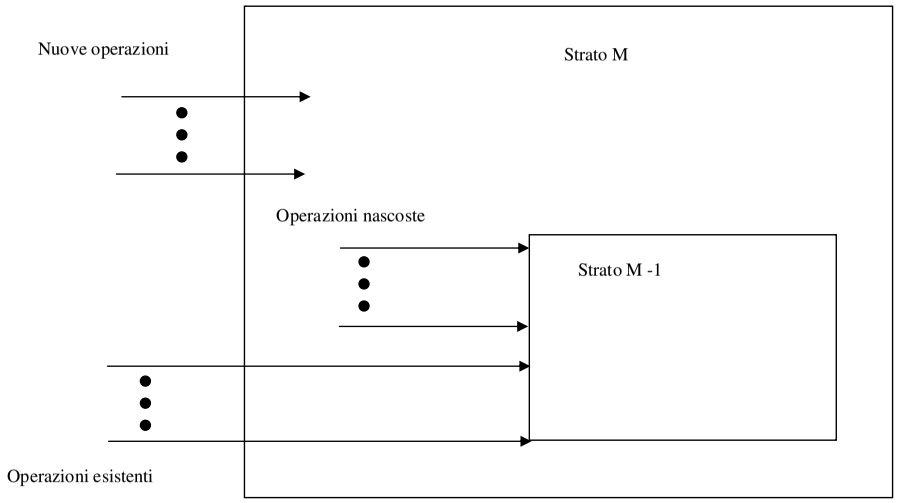
- La progettazione della interfaccia con l'utente e` indipendente dalla struttura interna del S.O.

Struttura del S.O.

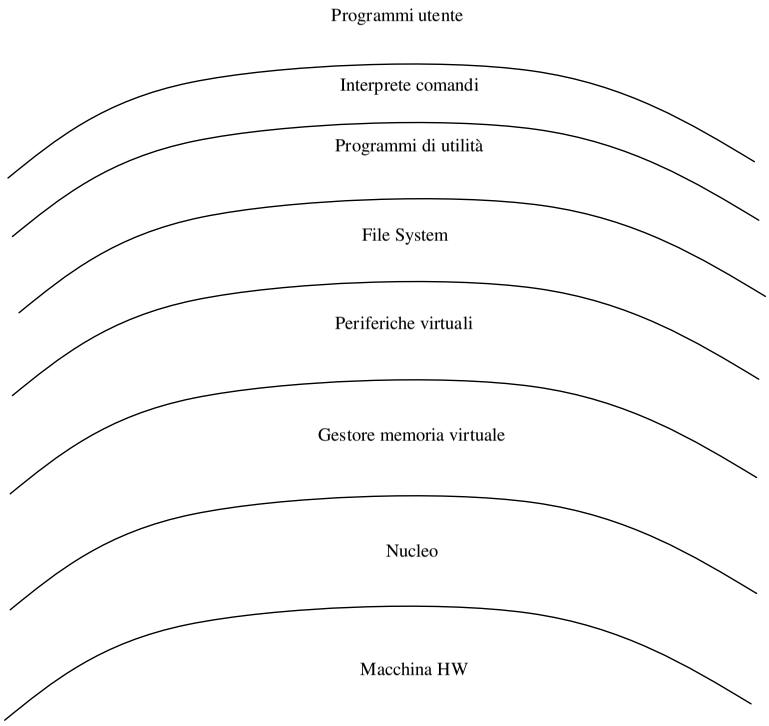
- Necessità di modularizzazione date le dimensioni

- Sistema a *livelli*: il livello piu` interno e` l'hw, quello piu` esterno l'interfaccia di utente

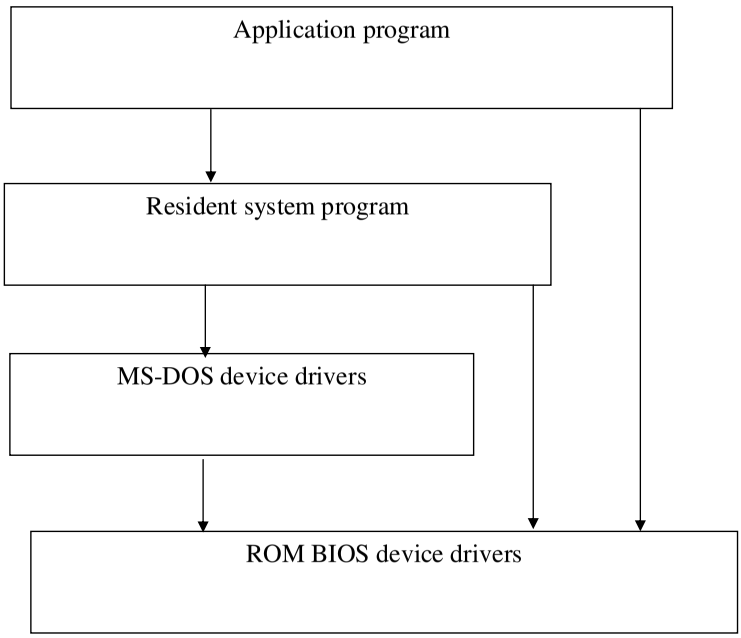
- Affinchè il livello Li possa richiedere i servizi al livello Li-1 deve conoscerne una *specifica* precisa, tuttavia l'*implementazione* di tali servizi deve risultare totalmente nascosta.



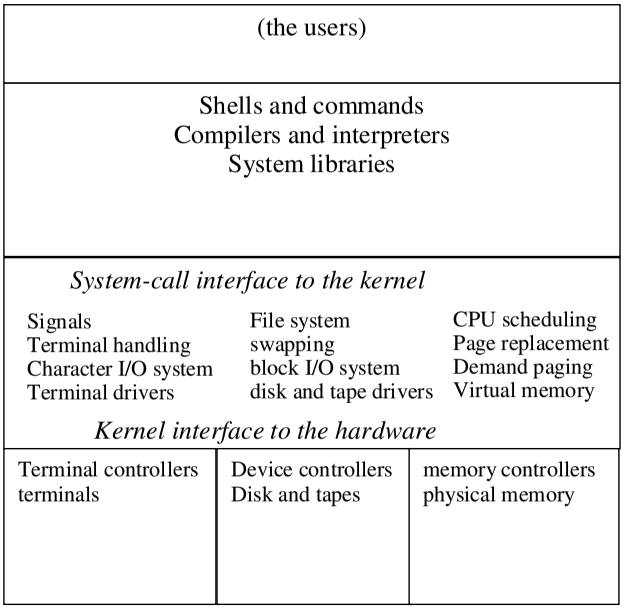
La struttura "a cipolla"



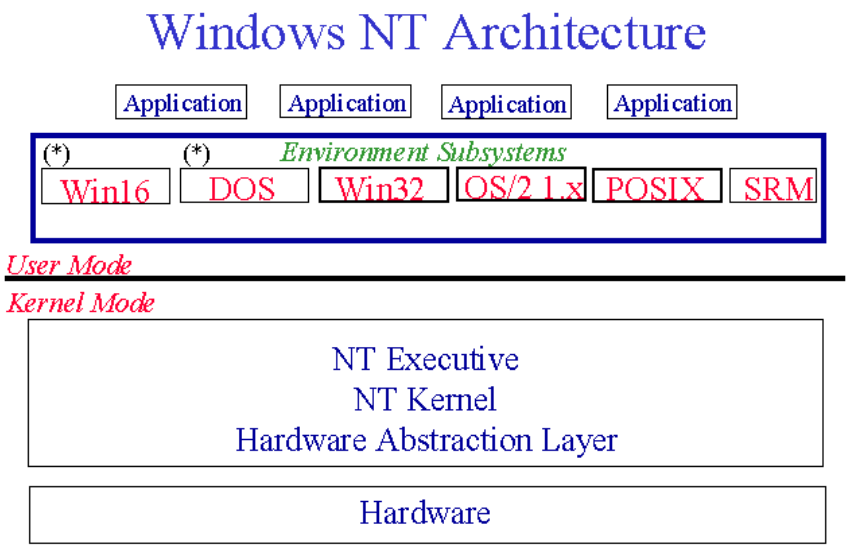
La struttura a livelli di MS-DOS



La struttura a livelli di UNIX



La struttura a livelli di Windows NT



(*) not an environment subsystem

Struttura del S.O.

- La stratificazione più opportuna può risultare non evidente; è dipendente dall'evoluzione tecnologica dell'hw.

- Sistema a *macchine virtuali* (VM IBM): usando lo scheduling della CPU e la tecnica della memoria virtuale, si possono creare macchine virtuali, una per ogni processo. Si consegue il massimo livello di protezione, a scapito dell'efficienza.

- Realizzazione in linguaggi ad alto livello (UNIX BSD4.3: 3% assembly, il resto in C)

- *Nucleo* o *kernel*: mette a disposizione le system call ai programmi di sistema ed applicativi.

Nucleo di un S.O.

- Fornisce un meccanismo per la creazione e la distruzione dei processi

- Provvede allo scheduling della CPU, alla gestione della memoria e dei dispositivi di I/O

- Fornisce strumenti per la sincronizzazione dei processi

- Fornisce strumenti per la comunicazione tra processi

Struttura gerarchica del Sistema Operativo

L0: bare machine
L1: processor management (lower module) / scheduler
L2: memory management
L3: processor management (upper module) [messaggi, creazione/distruzione processi]
L4: device management
L5: information management

\