



UNIVERSITÀ DI PARMA

il mondo che ti aspetta

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA



Evoluzione dei Sistemi Operativi

prof. Francesco Zanichelli

Stadi evolutivi dei sistemi di elaborazione

SisOp
2020/21



Sistemi isolati

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

Sistemi centralizzati - Mainframe

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

Sistemi decentrati - Minielaboratori

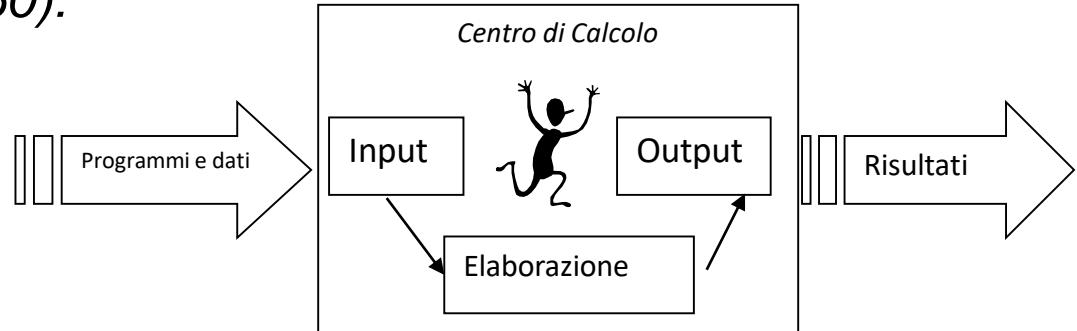
- Di nuovo decentralizzazione. Notevole potenza di calcolo ad un prezzo relativamente basso. "Rivolta dei mini" per superare la lentezza e la rigidità del servizio centralizzato.

Sistemi distribuiti

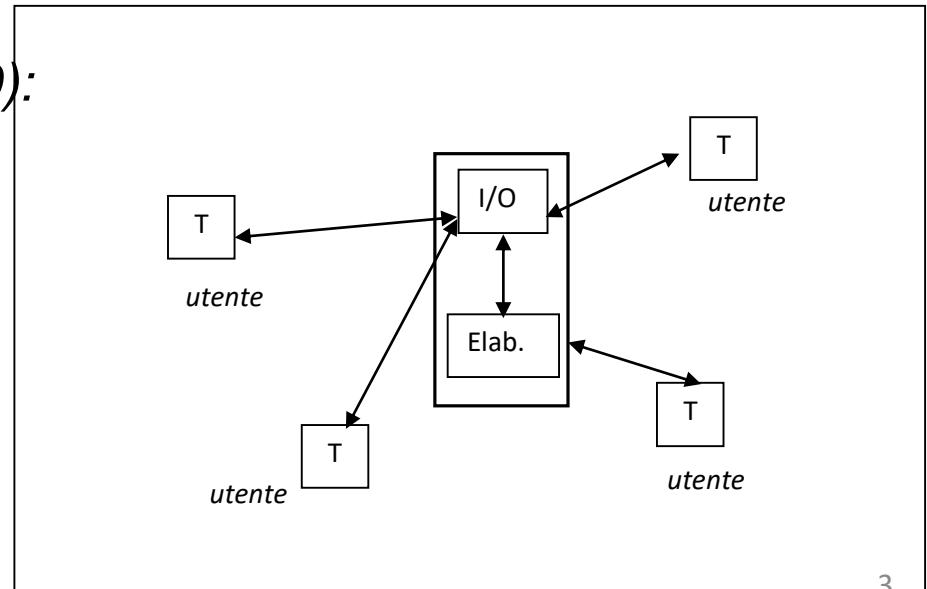
- Concepiti come un insieme di unità dotate di capacità operativa autonoma ed al tempo stesso in grado di scambiare mutuamente dati e risorse attraverso una rete di comunicazione.
- Intelligenza distribuita in un'ottica di cooperazione ed integrazione delle risorse.

Stadi evolutivi e modalità d'uso dei sistemi

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

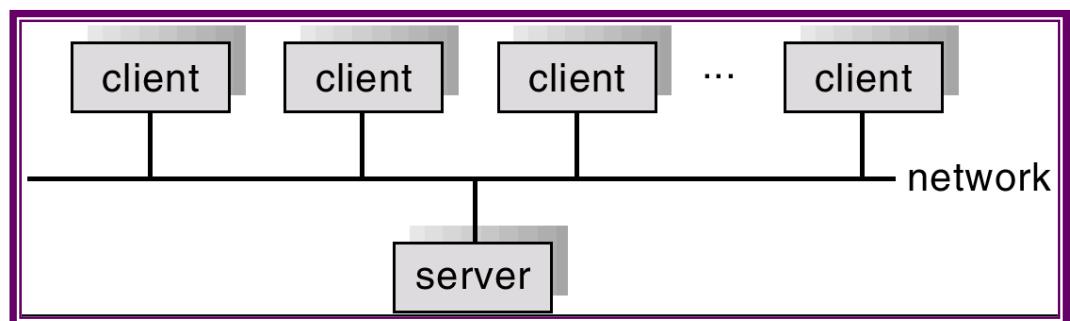
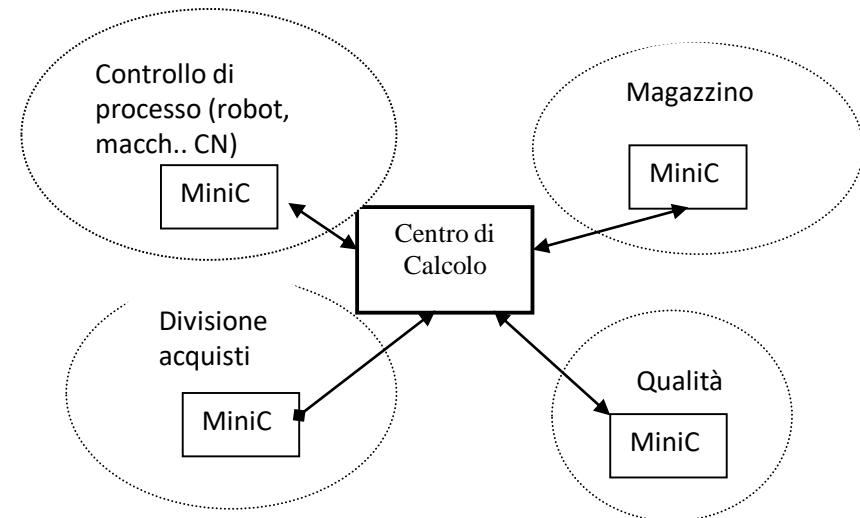


- il sistema *centralizzato* ('60-'70):
 - *remote job entry*
 - *teleprocessing*
 - *time-sharing*
 - *multiprocessing*
- facilitazione accessi
- distribuzione esperti



Stadi evolutivi e modalità d'uso dei sistemi

- i sistemi *decentralati* ('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*



Client-Server

SisOp
2020/21



- I dispositivi **client** sono utilizzati dagli utenti
 - dispositivi mobili (smartphone, tablet,...)
 - Thin Client (terminali utilizzati nei sistemi di virtualizzazione del desktop)
 - PC portatili/fissi
- Molti sistemi sono **server**, che rispondono alle richieste di servizio dei client, offrendo:
 - Servizi di calcolo o gestione dati (ad es. DBMS)
 - Servizi di gestione file (ad es. NFS)
 - Altri servizi

Stadi evolutivi e modalità d'uso dei sistemi

- Adesso?
 - Cloud, mobilità
 - Intelligenze negli apparati (smart everything)
 - Processing locale e in back-end (cloud, P2P, servizi)
 - Stratificazione software e riuso del software
- Risorsa cruciale, di cui c'è sempre carenza:
 - brainware

Stadi evolutivi e modalità d'uso dei sistemi

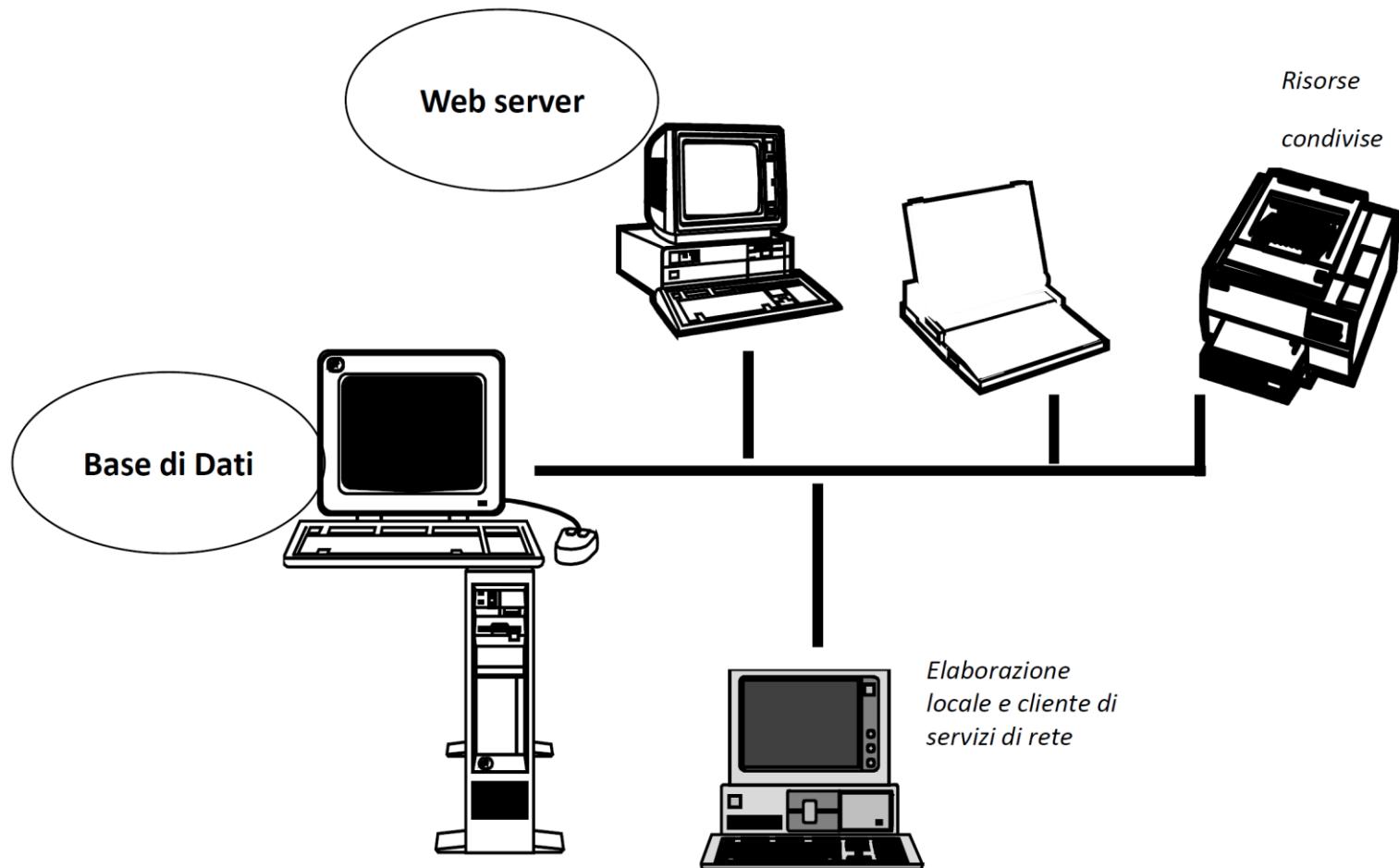
Sistemi distribuiti

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

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

Sistema di elaborazione distribuito (ufficio)

SisOp
2020/21



Sistema di elaborazione distribuito

SisOp
2020/21



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*.

Sistemi distribuiti per High Performance Computing: Cluster

SisOp
2020/21



CLUSTER:

classe di architetture di elaborazione parallele che si basa su un insieme di elaboratori indipendenti cooperante per mezzo di una rete di interconnessione e coordinato mediante un sistema operativo che lo rende in grado di operare su di un singolo workload.

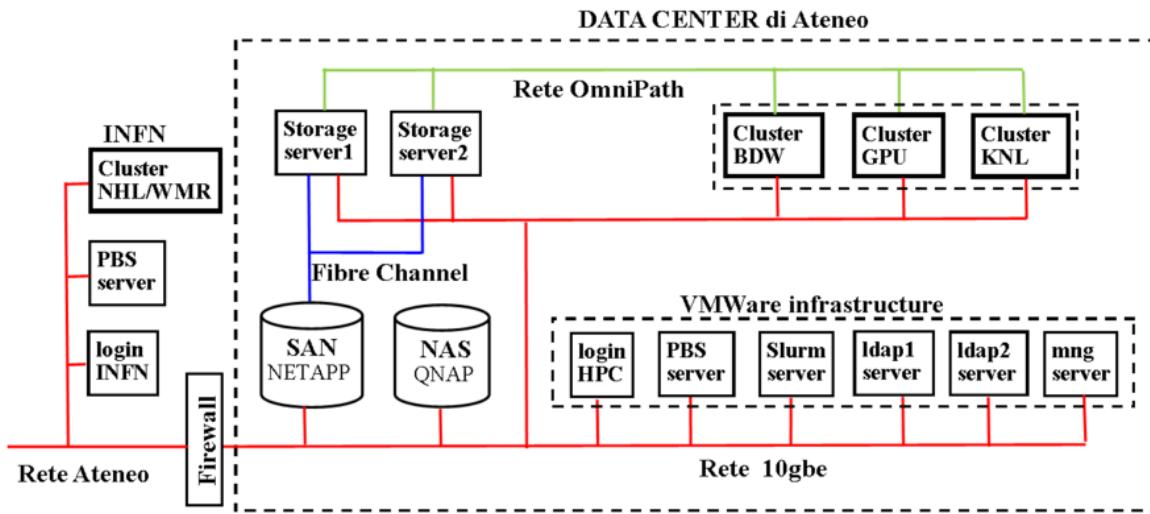


COMMODITY CLUSTER:

è costituito da nodi di elaborazione commerciali (COTS – Commercial off the shelf) in grado di operare in modo indipendente collegati tramite una rete di interconnessione (LAN o SAN) anch'essa di tipo COTS.



Cluster HPC@UNIPR



Nuovi nodi calcolo

- Cluster1 ([BDW](#))
 - 8 nodi con 2 Intel Xeon E5-2683v4 (2×16 cores, 2.1GHz, 40MB smartcache), 128 GB RAM.
 - 1 nodo con 2 Intel Xeon E5-2683v4 (2×16 cores, 2.1GHz, 40MB smartcache), 1024 GB RAM.
- Cluster2 ([GPU](#))
 - 2 nodi con 2 Intel Xeon E5-2683v4 (2×16 cores, 2.1GHz), 128 GB RAM, 5 GPU NVIDIA P100-PCIE-12GB (Pascal architecture).
- Cluster3 ([KNL](#))
 - 4 nodi con 1 Intel Xeon PHI 7250 (1×68 cores, 1.4GHz, 16GB MCDRAM), 192 GB RAM.

Dettaglio dei nodi:Prestazioni di picco (doppia precisione):

1 Nodo BDW -> 2x16 (cores) x 2.1 (GHz) x 16 (AVX2) = 1 TFlops, Max memory Bandwidth = 76.8 GB/s

1 GPU P100 -> 4.7 TFlops

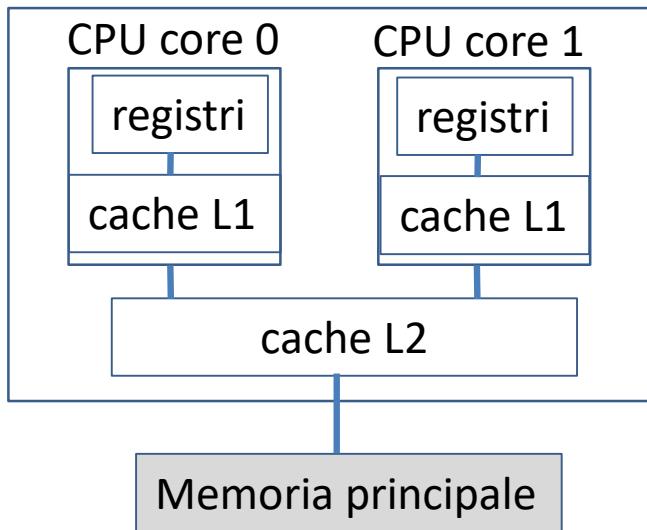
1 nodo KNL -> 68 (cores) x 1.4 (GHz) x 32 (AVX512) = 3 TFlops, Max memory bandwidth = 115.2 GB/s

Interconnessione con [Intel OmniPath](#) Prestazioni di picco: Bandwidth: 100 Gb/s, Latency: 100 ns.

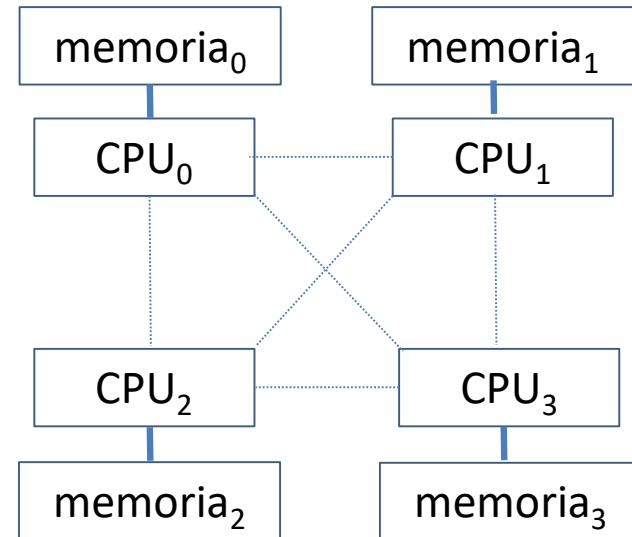
Sistemi multiprocessori

- I singoli nodi di un cluster/server, ma anche i normali PC, gli smartphone e molti sistemi embedded, hanno oggi di norma più di una CPU:
 - 2, 4, (6), 8 core e oltre
 - a volte uguali tra loro, a volte eterogenee (ad es. prestazioni e consumo differenti)

Dual core (MP simmetrico)



Quad core NUMA (Non Uniform Memory Access)



La (lunga) storia dei Sistemi Operativi

Sono nuovo qui,
avete un computer
con un vecchio sistema operativo?



Ti rendi conto che siamo stati assieme
per tre versioni di Windows?



Evoluzione dei S.O.

SisOp
2020/21



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)

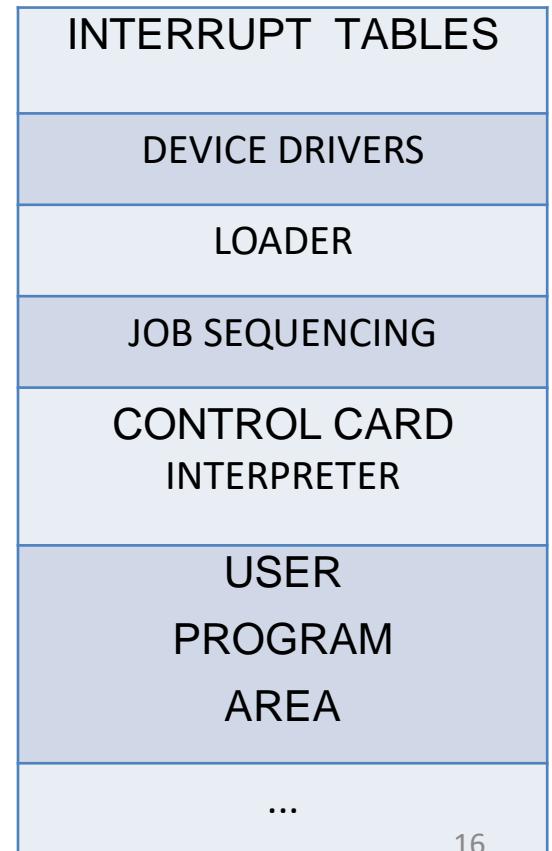
SisOp
2020/21



- *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*
- necessità di ridurre *idle time* => *automatic job sequencing*
- Monitor residente in memoria
- Il monitor richiede:
 - schede di controllo (JCL) per interpretare le richieste dell'utente (ad es. compilatore utilizzato)
 - 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-'85):**
 - sistemi a macchine virtuali
 - sistemi multiprocessore e distribuiti
 - interfacce amichevoli per l'utente
- **Quinta generazione ('85-'95)**
 - elaboratori personali
 - reti locali, avvio di internet, cultura del web
- **Sesta generazione ('95-'05)**
 - *elaborazione distribuita*, peer to peer, servizi online, commercio elettronico, ...

Tecniche di gestione di un sistema di calcolo

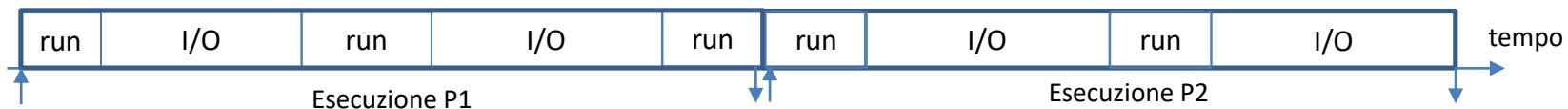
SisOp
2020/21



- 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:

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

T_p = tempo dedicato dalla CPU alla esecuzione del programma
T_t = tempo totale di permanenza nel sistema del programma

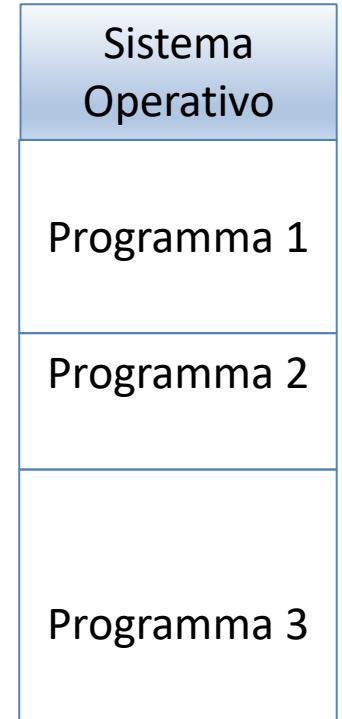
- throughput = numero di programmi eseguiti per unità di tempo

Sistema multiprogrammato

SisOp
2020/21



- Carica in memoria e 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.



Organizzazione della memoria principale

Gestione Batch

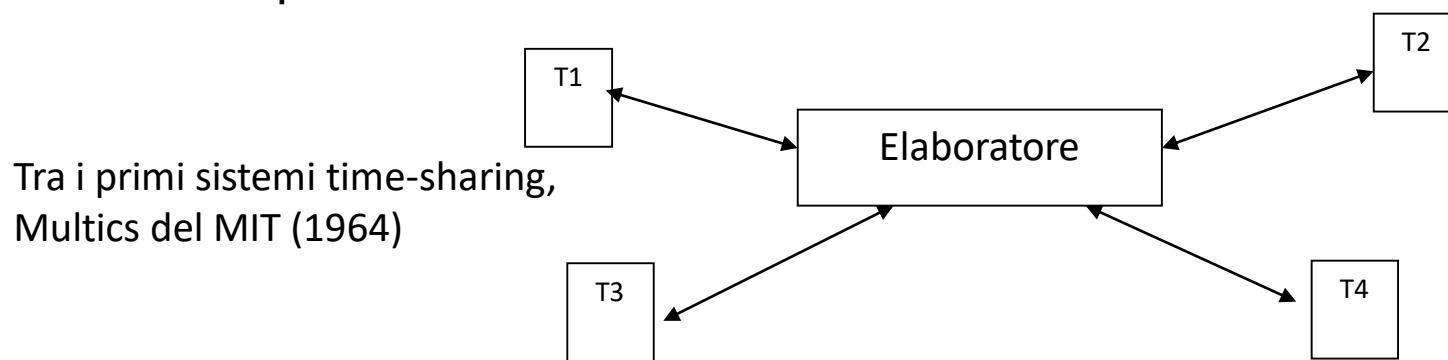
SisOp
2020/21



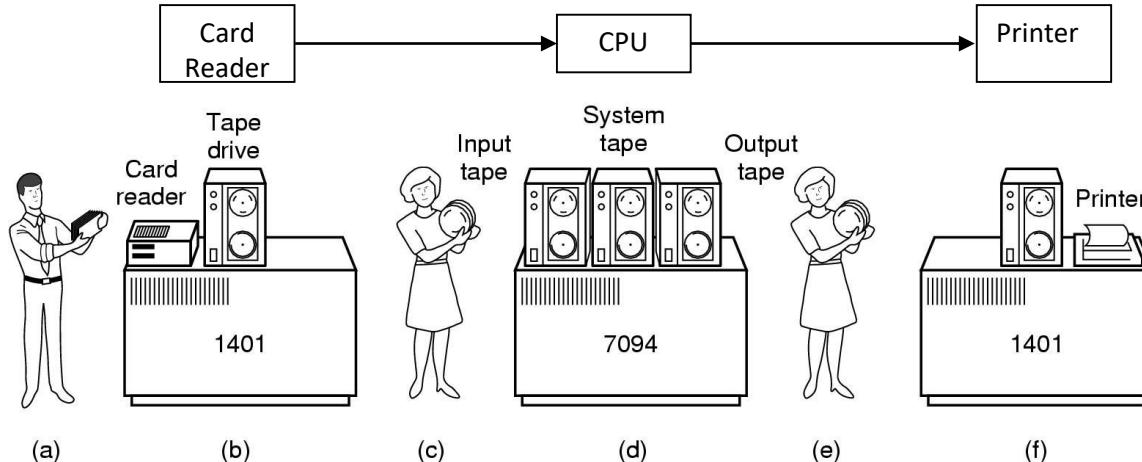
- Significa raggruppare i lavori o i programmi in *lotti* per conseguire una *maggior utilizzazione delle risorse*, cioè un throughput più elevato
- Un concetto che si è evoluto nel tempo:
 - l'operatore raggruppa i programmi in lotti e li immette in tale forma nel sistema per un più razionale utilizzo 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 (time slice)*
- Migliora i *tempi di risposta* (turn-around time) ma peggiora l'utilizzazione delle risorse
- La multiprogrammazione permette ad ogni utente di vedere il comportamento dell'elaboratore come se gli fosse del tutto dedicato
- Normalmente una modalità di gestione time-sharing è adottata nei sistemi *conversazionali*, in cui più utenti contemporaneamente "colloquiano" con il sistema



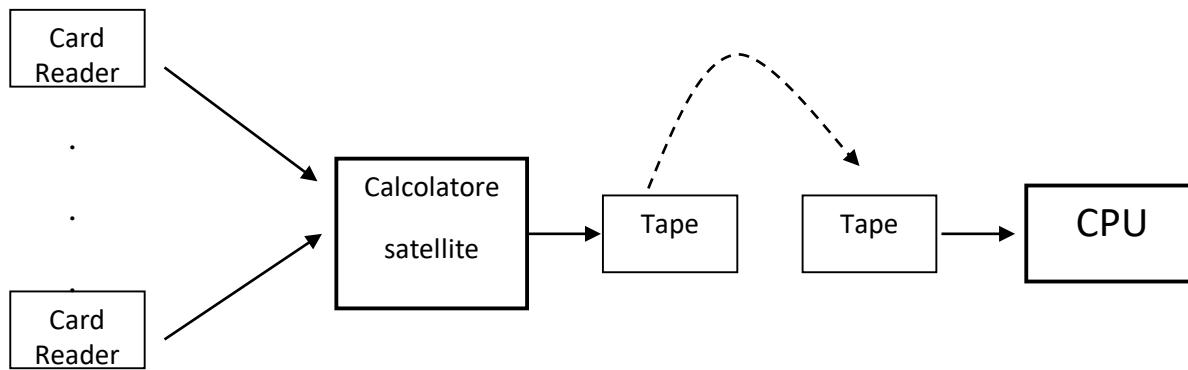
Evoluzione dei sistemi batch



- **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

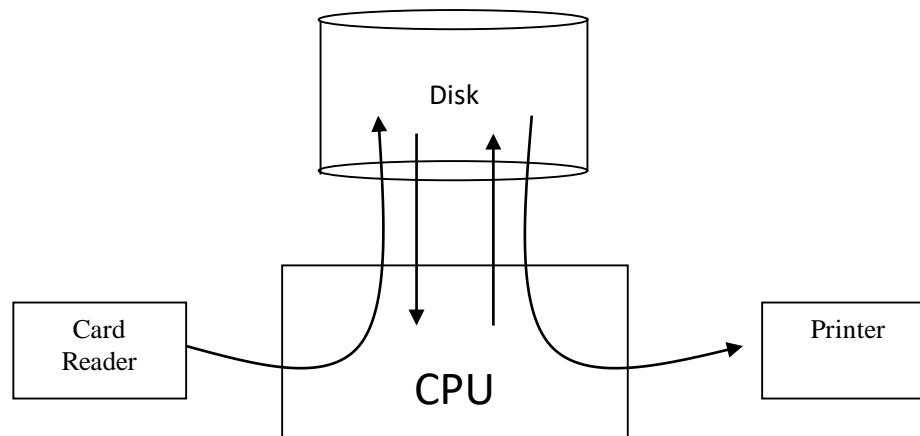
- Vantaggio ulteriore delle operazioni fuori-linea è la possibilità di utilizzare più lettori di scheda collegati con una stessa unità nastro in ingresso e più stampanti collegate con una stessa unità nastro in uscita



- 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 che permettono solo un accesso sequenziale).



Richiami e notazione

- Un sistema di elaborazione è costituito da una o più CPU, memoria principale, memoria secondaria, dispositivi di I/O
- Memoria: un vettore $M[0:2^n-1]$ (es. $n=32 \rightarrow$ circa 4 miliardi di indirizzi) 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]

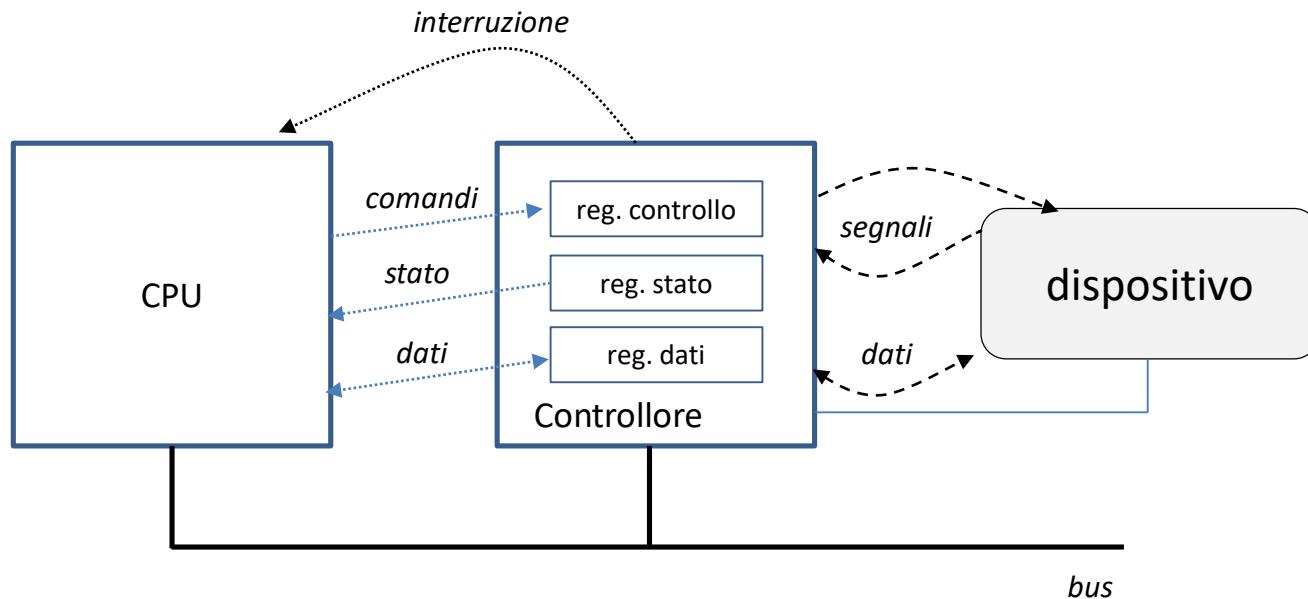
 PC := PC + 1

 execute(istruz. in IR) // execute di un jump determina la modifica del PC, etc.)

until CPU halt

Controllore di un dispositivo

SisOp
2020/21



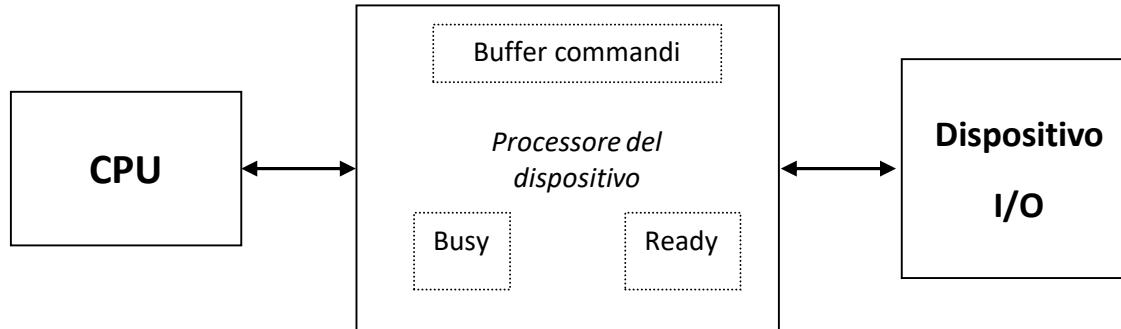
Un modello semplificato

SisOp
2020/21



- Nota: Ci riferiamo inizialmente al caso (*semplificato*) di assenza nella CPU del meccanismo delle **interruzioni**
- In questo caso la gestione dell'I/O da parte della CPU è necessariamente a **polling** (detta anche a controllo di programma), e come vedremo risulta inefficiente (a causa dei lunghi tempi morti della CPU in attesa del completamento delle operazioni I/O – in generale assai lente)
- Il compito del S.O. è quindi più semplice ma l'efficienza è bassa dato che non è possibile sovrapporre l'attività di CPU con quella di I/O

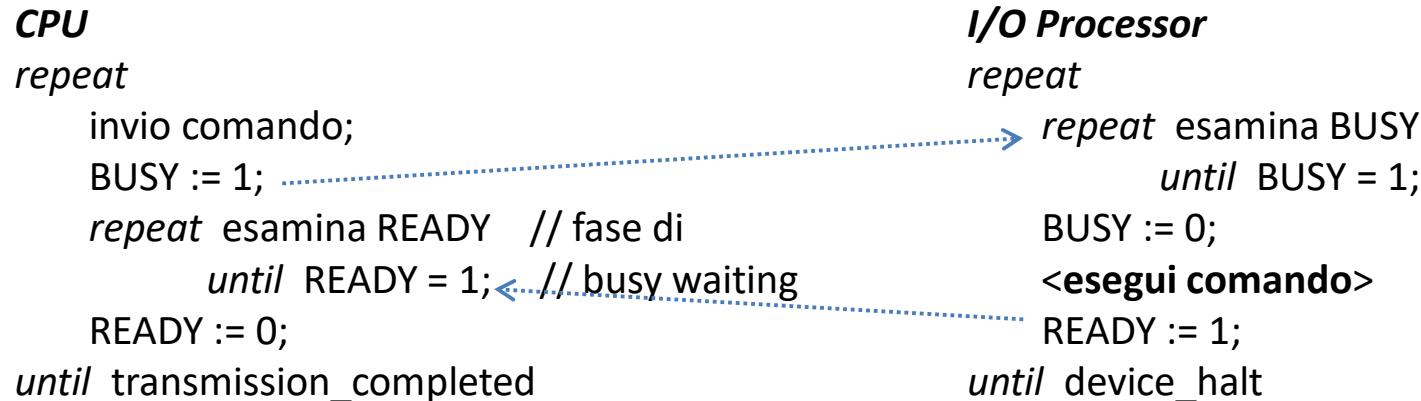
Un modello di interazione tra CPU e dispositivi di I/O



1. La CPU inserisce un comando nel buffer ed attiva il processore del dispositivo mettendo ad 1 il flag BUSY
2. Il processore 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 può essere espresso come:



- Generalmente il tempo dedicato all'I/O costituisce una parte fondamentale del *tempo complessivo di esecuzione* di un programma (cioè 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 attività di I/O.

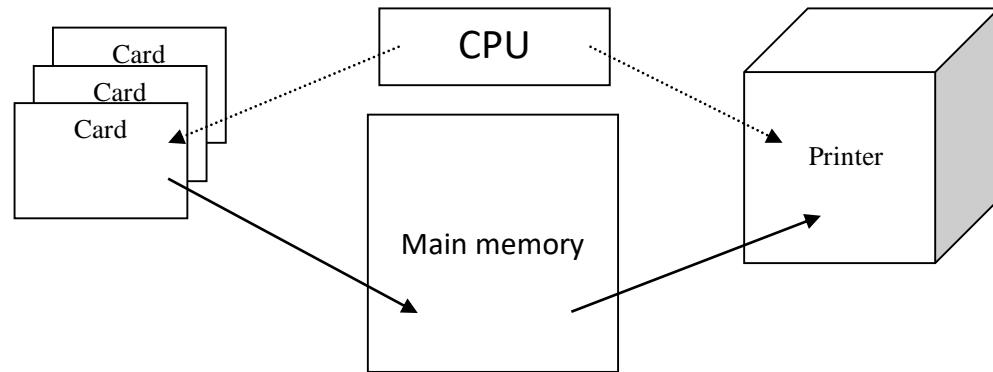
Un semplice S.O.

- Sequenza di operazioni:

repeat

leggi il pacco di schede
compila
carica
esegui
stampa i risultati

until il calcolatore si ferma



- *Lettore di schede (CR) e stampante (LP)* sono le *metafore storiche* dei dispositivi di I/O.
- Compito del S.O. è 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.

*Nota Bene : si ricorda che da questa slide fino alla n. 33 si descrive un modello **teorico** di SO per un **ipotetica CPU senza interruzioni** di interesse unicamente didattico, appunto per mostrare come tale sistema sarebbe assolutamente inefficiente e con minimi margini di miglioramento*

Gestione I/O

SisOp
2020/21



- 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	Occupato

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	Occupato

Programmi di controllo I/O

SisOp
2020/21



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 velocità della CPU** che esegue il programma di controllo **nei confronti dei dispositivi**.
- Si e` introdotta una *forma di sincronizzazione* tra due dispositivi di per sé asincroni.

Prestazioni del S.O.

- La notevole differenza di velocità tra CPU e dispositivi periferici fa sì che i programmi di controllo trascorrono 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: frequenza clock CPU > 1 GHz)

- Il tempo di esecuzione medio di una istruzione macchina, assumendo un tempo di ciclo di 1 usec, è circa 2,5 - 3 usec
- Il controllo software della stampa di una linea richiede circa **45 usec**
- Velocità di stampa: 20 linee / secondo \Rightarrow Tempo di stampa di una linea: $1/20 \text{ sec} = \mathbf{50000 \text{ usec}}$

Prestazioni

- Il tempo necessario per eseguire n programmi è:

$$t = I + CLE + O$$

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

- Il throughput è:

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

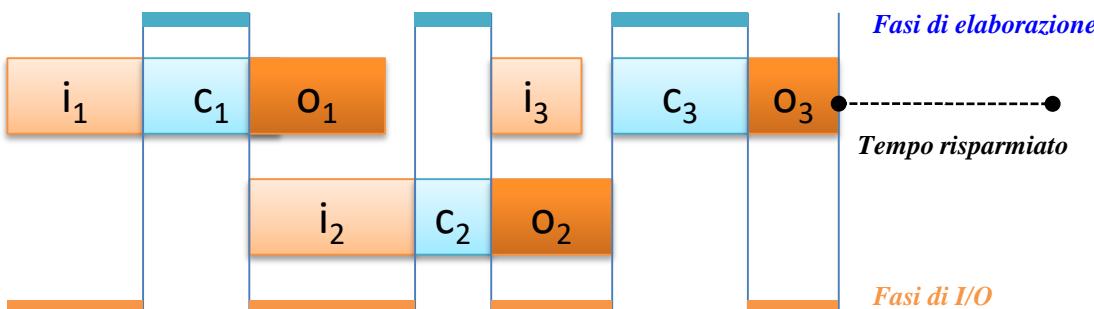
- Per migliorare il throughput occorre sovrapporre *le tre fasi I, CLE, O nel tempo*
- Tale sovrapposizione richiede che i dispositivi periferici e la CPU operino in modo *il più possibile indipendente*.

Sovrapposizione tra ingresso e uscita



- throughput di un sistema con esecuzione seriale:

$$\frac{n}{t} = \frac{n}{CLE + I + O} = \frac{n}{CLE + \sum_{k=1}^n (i_k + o_k)}$$



E' un miglioramento molto parziale: il vero miglioramento si ha sovrapponendo il più possibile i tempi di CLE (CPU) con quelli di I/O

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

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

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

Programma di controllo I/O

SisOp
2020/21



- 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/O

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

SisOp
2020/21



- 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 perché 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 ottenuta inserendo nei programmi, ad intervalli regolari, la richiesta di esecuzione dei programmi di controllo. Questa soluzione è *inaccettabile* perché fa ricadere sul programmatore un compito del S.O.
- La soluzione si ha attraverso il concetto di **interruzione**, realizzato tramite hardware.



UNIVERSITÀ DI PARMA

il mondo che ti aspetta

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA



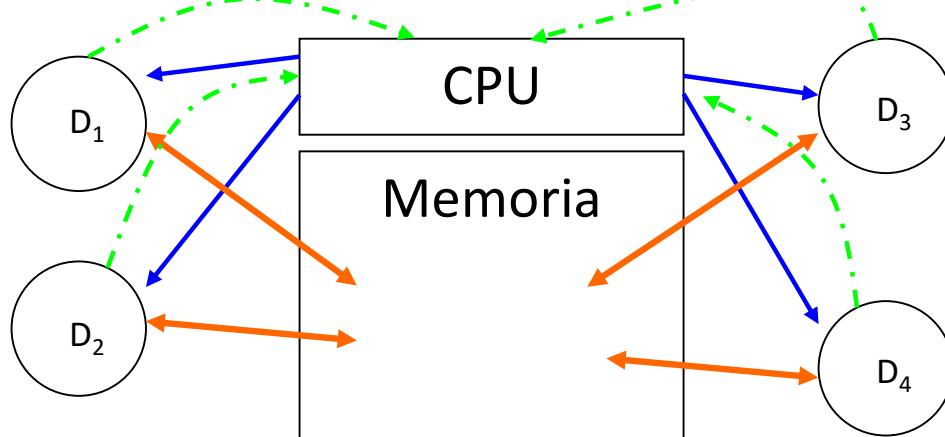
Interruzioni e multiprogrammazione

prof. Francesco Zanichelli

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

```

La CPU provvede all'esecuzione del programma corrente senza alcuna possibilità di essere interrotta per eseguire istruzioni specifiche per la gestione di un dispositivo che necessita di un intervento

Modello di comportamento dell'interruzione da dispositivo

- Nella CPU è presente un *vettore di bit* (*registro di richieste di interruzione - IRR*) *in cui ogni bit memorizza l'interruzione di un particolare dispositivo o di più dispositivi.*
- Il ciclo di Fetch-Execute diventa:

```
repeat
    if IRR = 0 then
        IR := M[PC]           // in assenza di interruzioni continua l'esecuzione del programma
        PC := PC + 1
    else
        IR := M[0] fi        // in presenza di interruzioni passa ad eseguire l'interrupt routine
        execute(istruz. in IR) // in questo caso (semplificato) con un entry point fisso
    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; /* azzerà 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)

SisOp
2020/21



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).

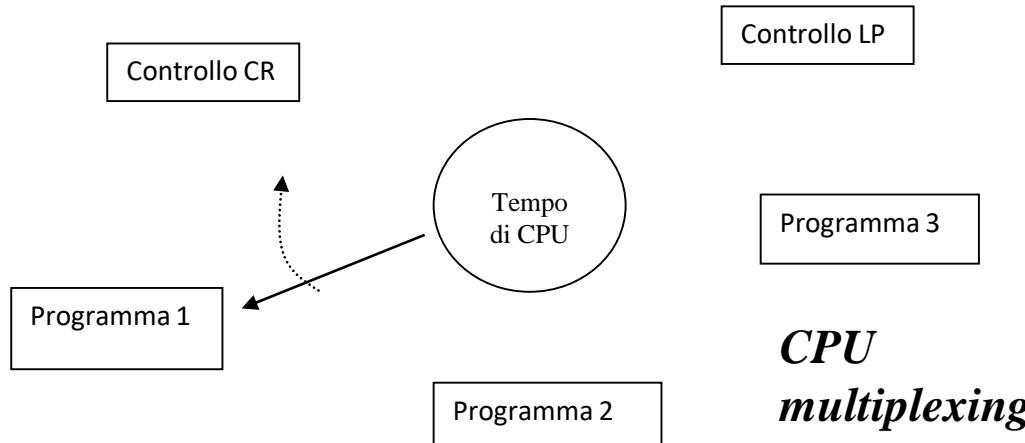


Interruzioni e S.O.

- E' compito del S.O. gestire le interruzioni tramite le *interrupt routine*.
- Nel semplice modello di S.O. visto in precedenza, *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:



- Piena utilizzazione delle risorse: l'insieme dei programmi caricati in memoria principale può essere scelto in modo da ottenere la massima occupazione della CPU (e della memoria).
- Programmi *I/O bound* e *Compute bound*

Multiprogrammazione e multitasking

- Multiprogrammazione : quando un processo (job batch) in esecuzione deve attendere un evento (ad es. I/O), il SO passa il controllo della CPU ad un altro processo in memoria, e così via.
- Multitasking : un'estensione logica della multiprogrammazione in cui la CPU commuta tra diversi processi (task o thread) più frequentemente (time sharing) e fornisce all'utente un veloce tempo di risposta.

Multiprogrammazione

Tranquilla, mamma,
riesco a fare tutto...



Mi porti i documenti, poi ci incontreremo.
Adesso devo lavorare per un altro cliente...



Interruzioni e multiprogrammazione

SisOp
2020/21



- Il S.O. raccoglie sia le interruzioni generate da eventi esterni (asincrone) sia quelle determinate dal programma in esecuzione (sincrone).
- Al termine della *routine di interruzione*, invece di ritornare al programma interrotto, il S.O. può scegliere, in base ad un *determinato algoritmo*, a quale programma *presente in memoria e pronto per l'esecuzione* affidare il controllo della CPU.
→ I programmi possono essere attivati (ovvero posti in esecuzione) 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 elevati di quelli I/O bound, è necessario introdurre anche un interrupt periodico (timer) che comunque determina l'invocazione dello scheduler.

Interruzioni e multiprogrammazione

SisOp
2020/21



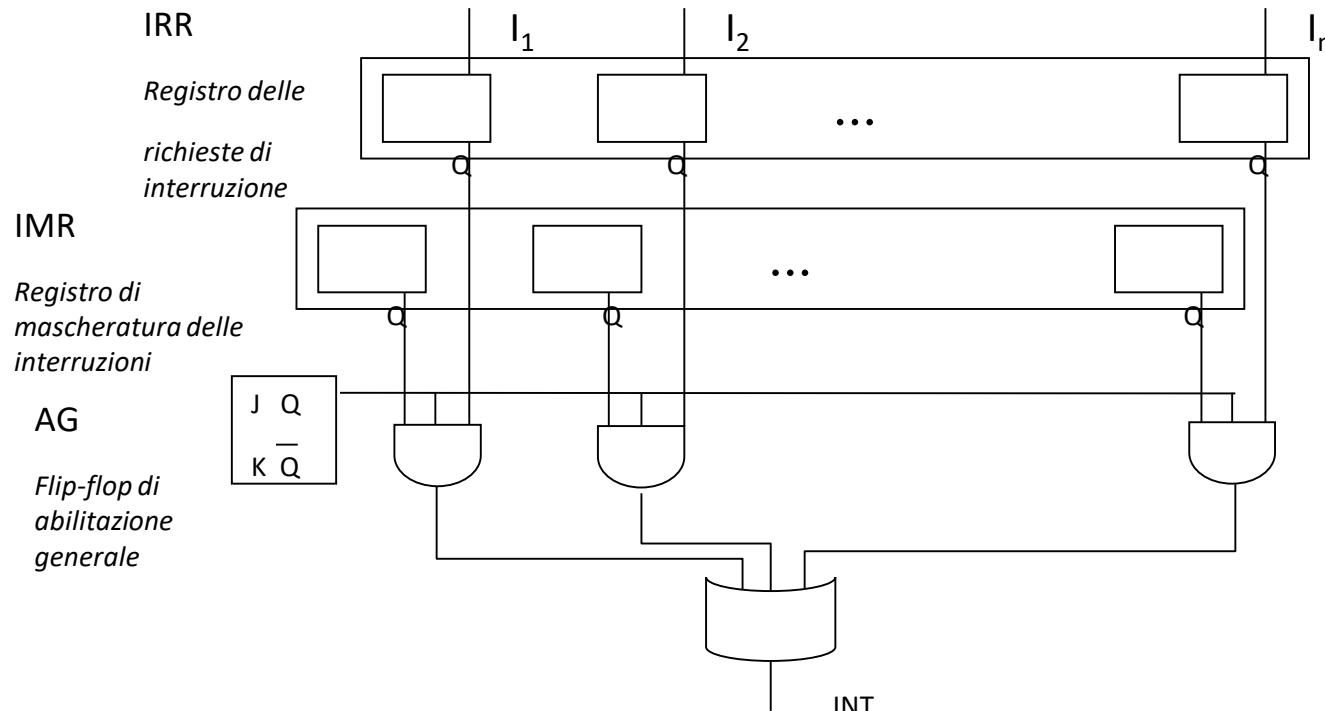
- Un programma può richiedere l'accesso ad un dispositivo già impegnato (eventualmente nel servizio di una richiesta precedente dello stesso programma!).
- Il S.O. deve pertanto:
 - mantenere in una tabella lo *stato* dei dispositivi,
 - sospendere un programma che intende accedere ad una risorsa già 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

1. 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
2. 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 sé un'interruzione ma solo una *richiesta di interruzione*. Affinché 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

SisOp
2020/21



- Ciascun evento "li" causa di interruzione è memorizzato in un flip-flop IRR_i ed abilitato da IMR_i 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)
 - $IRR_i = 1$ (si è presentata la causa di interruzione "li")
 - $IMR_i = 1$ (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

SisOp
2020/21

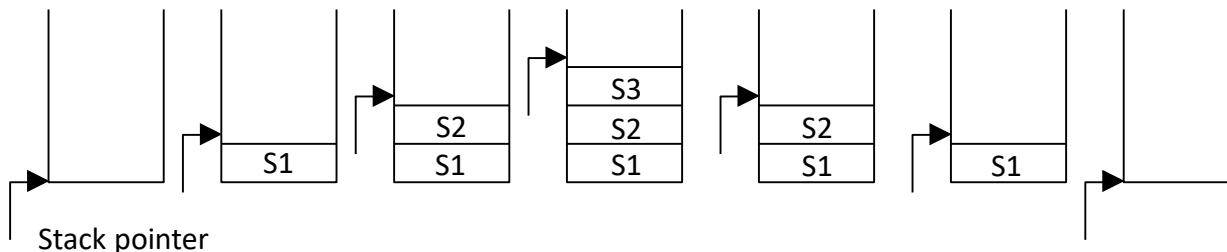


- Al verificarsi di una interruzione occorre:
 - a. salvare tutte le informazioni necessarie per la ripresa del programma interrotto
 - b. individuare la causa dell'interruzione
 - c. eseguire le azioni richieste (servizio dell'interruzione)
 - d. ripristinare lo stato del programma interrotto e riavviarlo
 - 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 fonti di interruzione possono suggerire una gestione con diversi livelli di priorità.
- Durante il servizio di un interrupt di livello L le interruzioni di livello $K \leq L$ restano disabilitate, mentre le altre possono essere abilitate. Questa gestione può essere realizzata o agevolata dall'hw (*PIC - 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 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

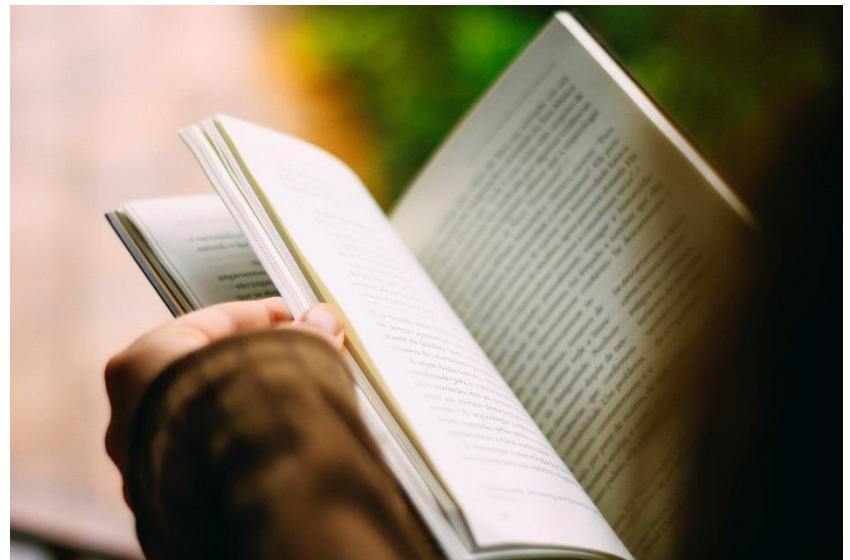
Busy waiting vs Interruzioni

L'attesa per la consegna del nuovissimo gadget...

Ma quando arriva ?!?



Drin ! Drin ! E' il corriere!





UNIVERSITÀ DI PARMA

il mondo che ti aspetta

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA



I processi

prof. Francesco Zanichelli



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 programma all'altro (anziché "ricevere" programmi o job 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*.

Algoritmo, Programma, Processo

SisOp
2020/21



- **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
 - Un programma eseguibile, ad es. un file EXE generato da un compilatore è un oggetto passivo, un insieme di byte che rappresentano istruzioni e dati
- **Processo (sequenziale):** La sequenza di eventi cui dà luogo un elaboratore quando opera sotto il controllo di un particolare programma (evento = esecuzione di una operazione)



Processi

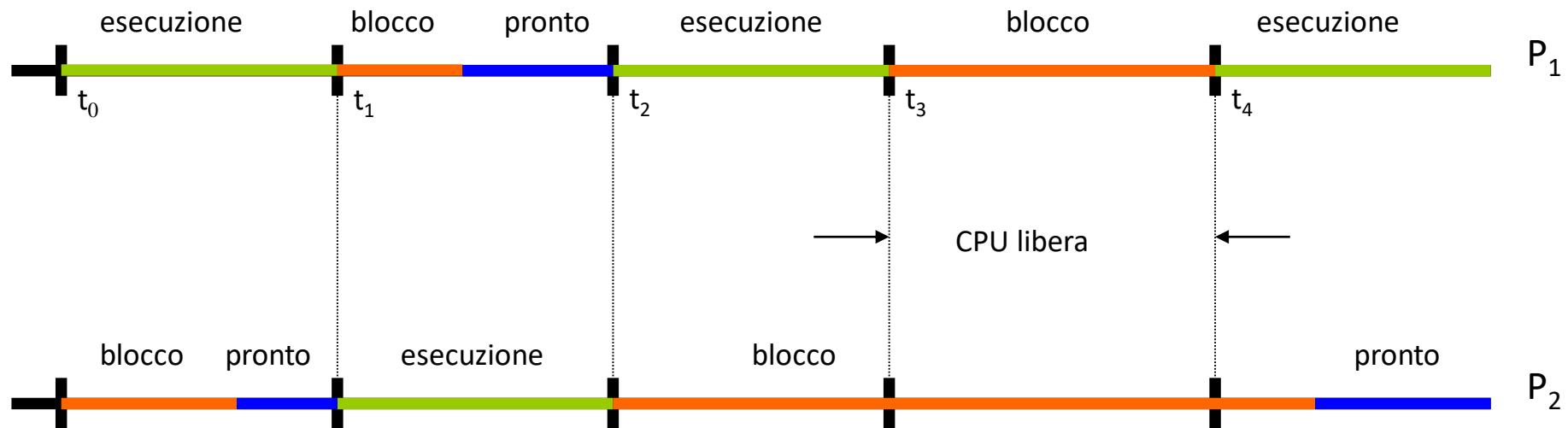
- **Un processo (o task) è l'unità funzionale in un S.O.**
- Un processo è *controllato da un programma* e ha bisogno di un *processore per la esecuzione*.
- Alcuni processi dispongono di un processore privato e pertanto sono permanentemente in esecuzione (ad esempio i controllori delle periferiche).
- 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 termine **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** (se la CPU è una sola, in generale uno per CPU del sistema).
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

SisOp
2020/21



Processo:

"Attività svolta dalla CPU per l'esecuzione di un determinato programma"

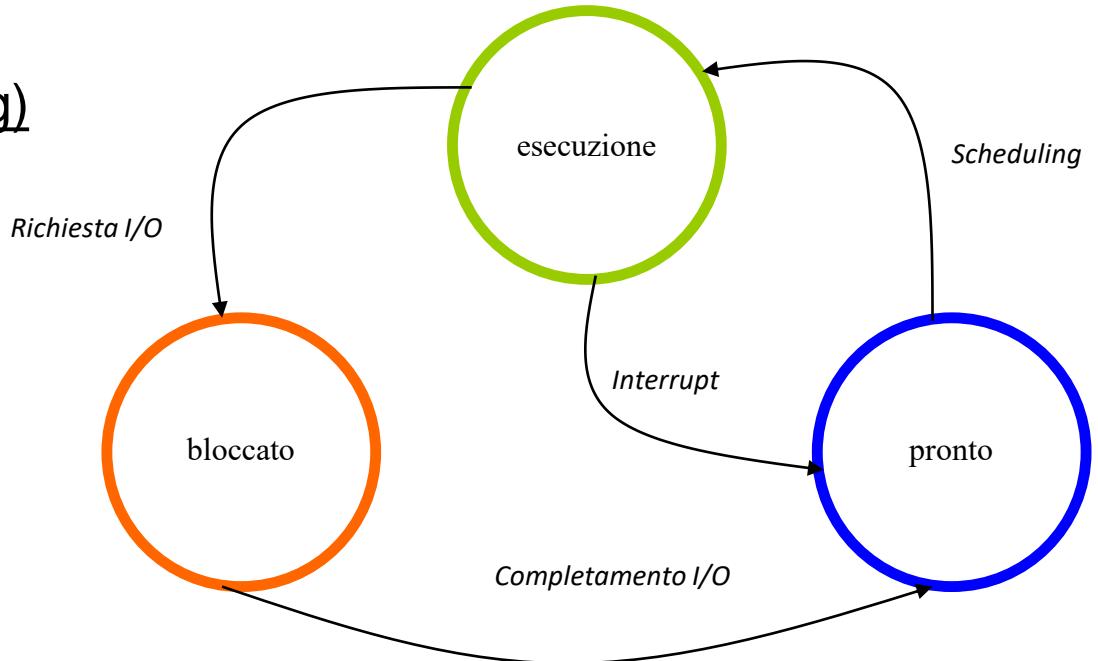
Stato dei processi

- Per tenere traccia e gestire correttamente i programmi in memoria principale, il S.O. deve associare esplicitamente ad essi un concetto di **stato del processo**:
 - In esecuzione se attivo sul processore
 - Pronto per l'esecuzione se dispone di tutte le risorse e le condizioni logiche per eseguire ma non dispone del processore
 - In attesa (o **bloccato**) se non dispone delle risorse e delle condizioni logiche per essere eseguito
- L'evoluzione dello stato dei processi è guidata dagli eventi e dalle decisioni del S.O. lungo traiettorie definite da un *diagramma degli stati*

Stato di un processo

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

(Altri strati transitori per creazione e terminazione)



- Se numero di CPU = numero di processi:

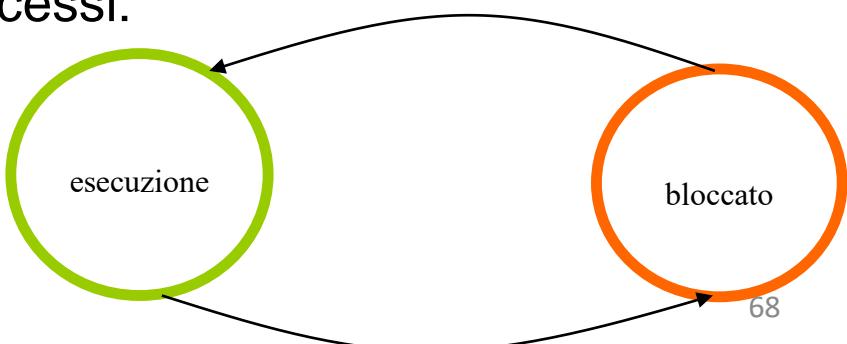
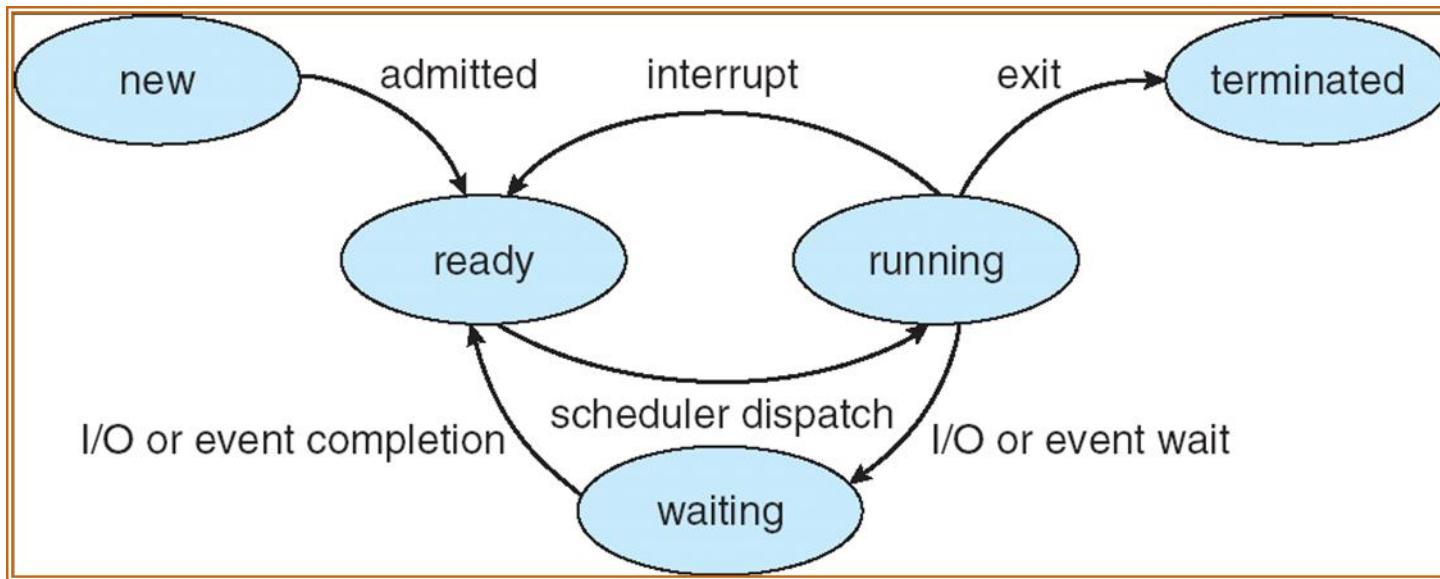


Diagramma di stato dei processi

SisOp
2020/21



Stati dei processi UNIX

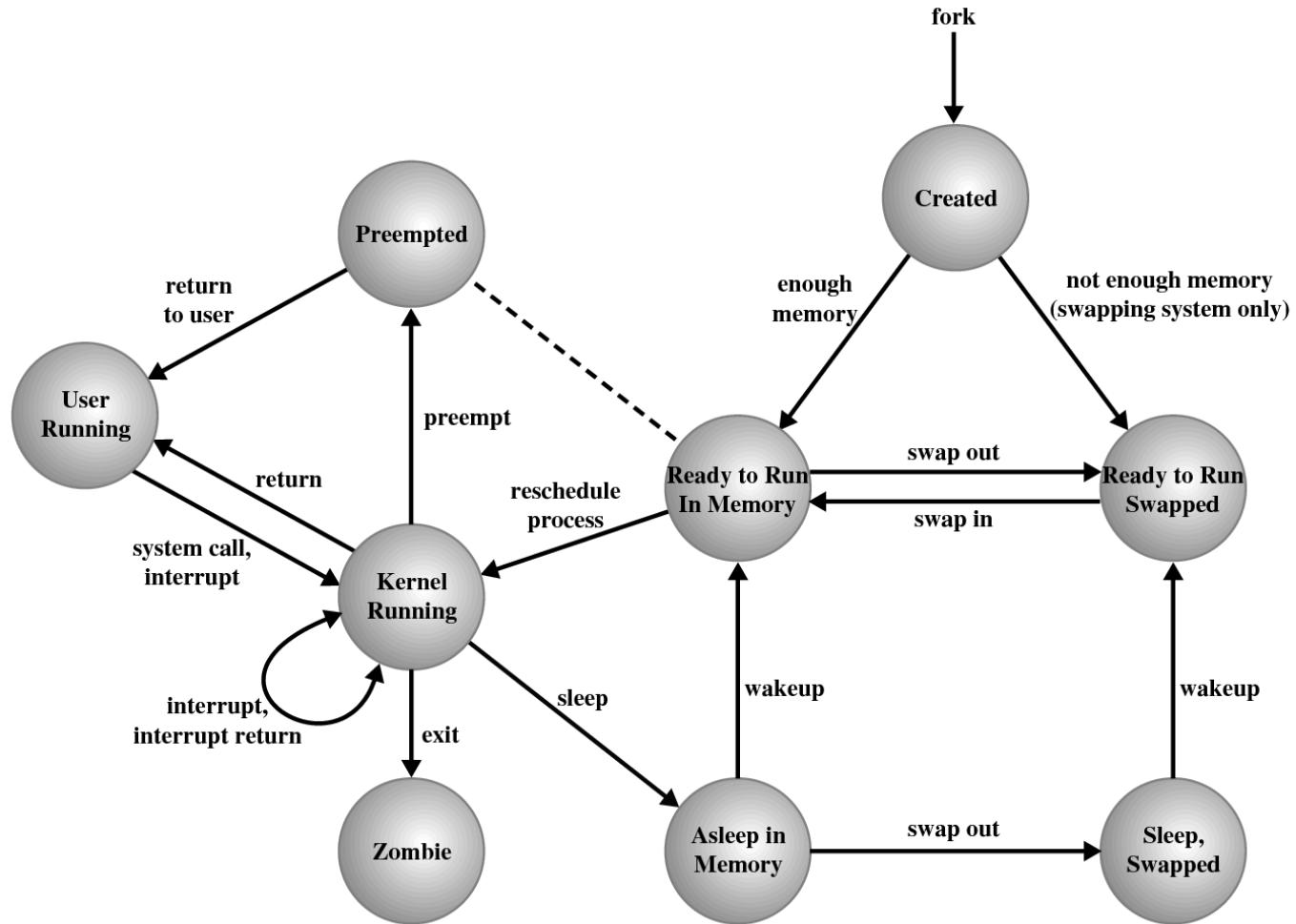


Figure 3.15 UNIX Process State Transition Diagram



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

SisOp
2020/21



- 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 area protetta del S.O., associata al processo e contenente tutte le informazioni proprie del processo

Process Control Block (PCB)

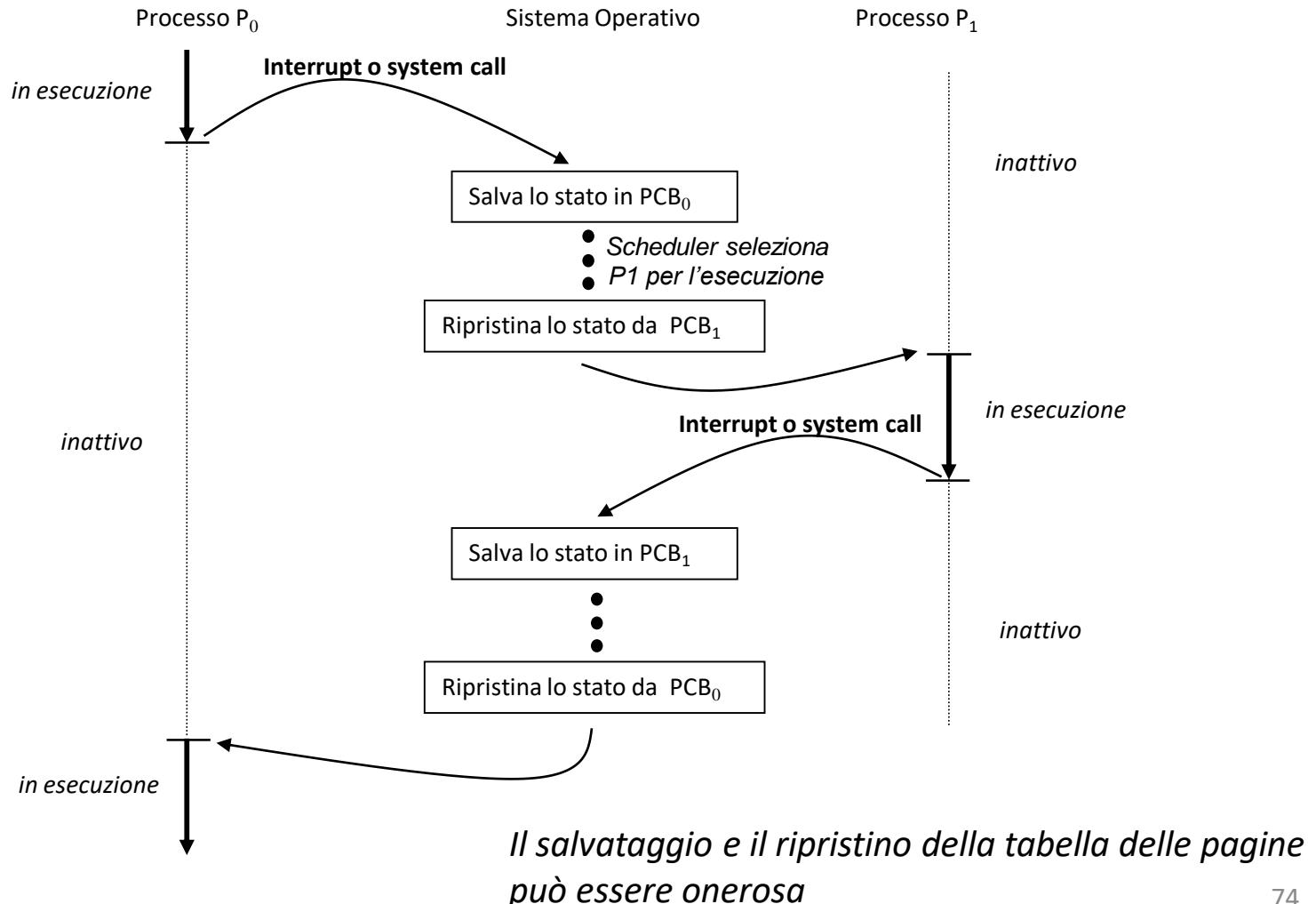
- Stato del processo: pronto, bloccato, in esecuzione
- Program Counter (PC)
- Registri della CPU: accumulatori, registri indice, registri general-purpose
- Informazioni per la gestione della memoria: registri base e limite, tabella delle pagine (per il demand paging, vedi più avanti)
- Informazioni 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

pointer	Process state
Process ID	
Program Counter	
registers	
Memory limits	
List of open files	

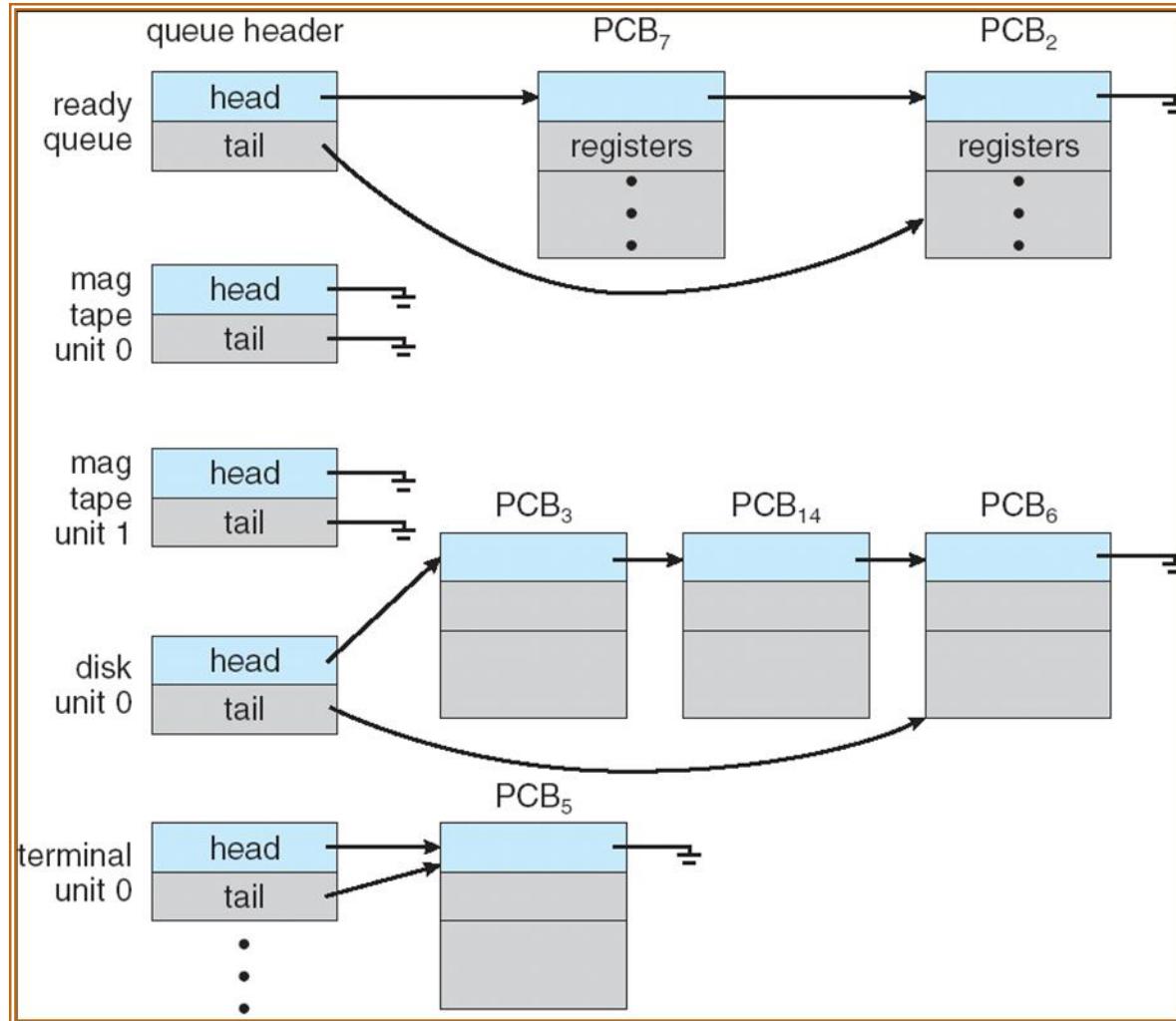
*Nei sistemi **UNIX** del passato parte di queste informazioni risiedevano nella **user area**: una porzione di memoria nell'immagine del processo **accessibile solo in modo kernel***

In **Linux** il PCB è costituito dalla struttura C **task_struct** definita in <linux/sched.h>
Il kernel mantiene una lista doppiamente concatenata di task_struct con un puntatore **current** al processo attualmente in esecuzione

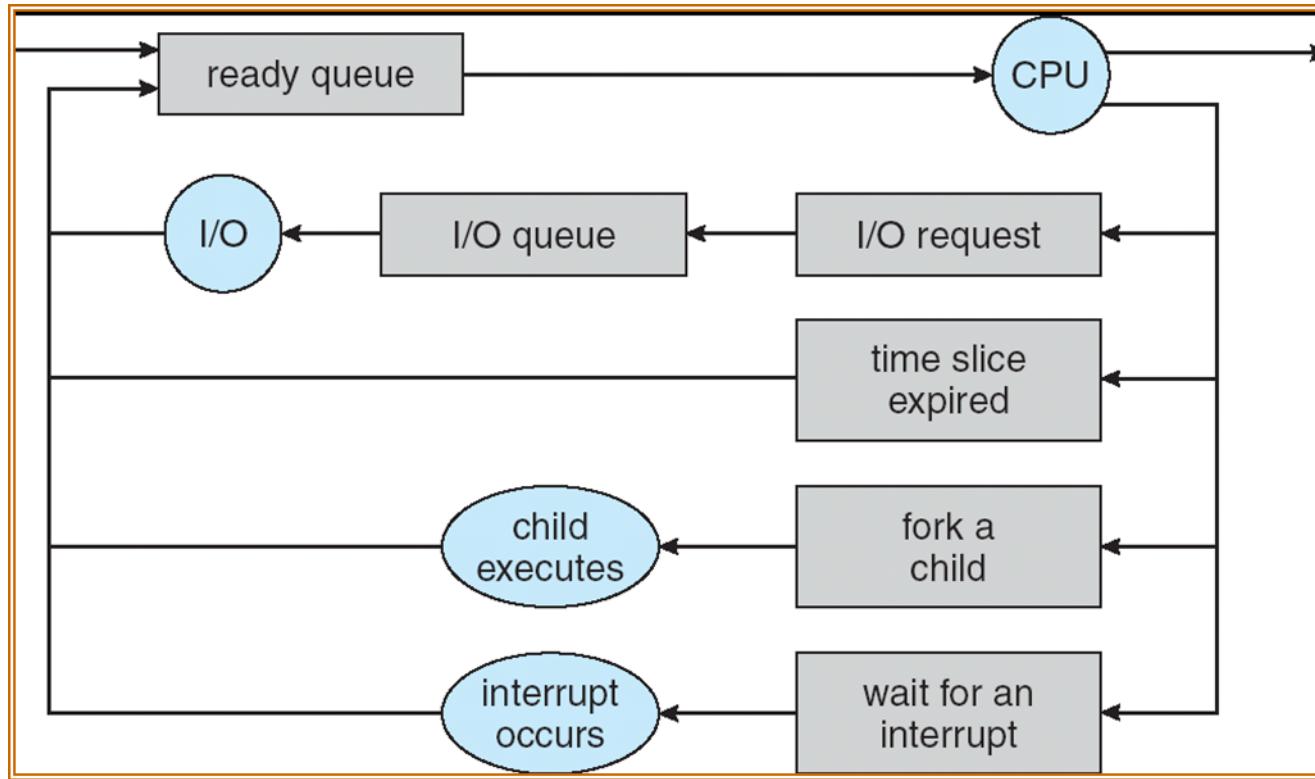
Commutazione della CPU tra processi



Gestione delle transizioni di stato



Sistema operativo come gestore di code





UNIVERSITÀ DI PARMA

il mondo che ti aspetta

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA



Organizzazione e funzioni del SO

prof. Francesco Zanichelli

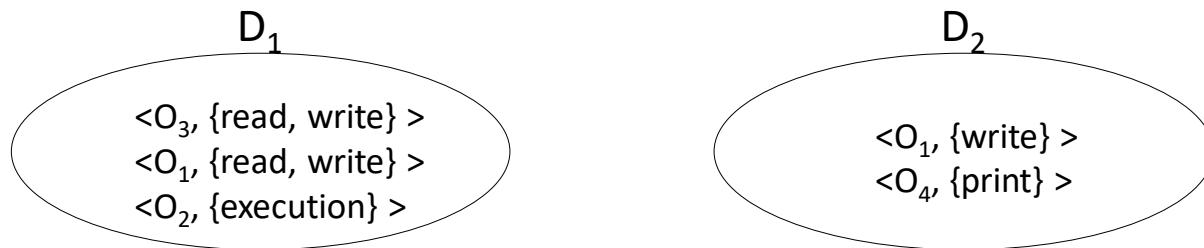


Sistema di protezione

- Un processo potrebbe tentare di modificare il programma o i dati di un altro processo o di parte del SO 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 ai processi l'accesso diretto ai dispositivi, etc.
- All'hardware è 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.
- Gli errori vengono segnalati e affidati alla gestione del SO tramite il meccanismo delle *trap*.
- In presenza di errore o di violazione della protezione il SO provvede a terminare il processo, segnala la terminazione anomala ed effettua eventualmente un *dump* della memoria.

Sistema di protezione

- In generale, 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 all'esecuzione di un'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.).

Esempio di capability

- Una *capability* (*a volte anche denominata key* – chiave) è un token di autorità immodificabile dai processi ma che può essere comunicato tra essi. Si tratta di un valore che è un riferimento a un oggetto e a un insieme associato (lista) di diritti di accesso. Un programma utente in un SO basato su *capability* deve necessariamente utilizzare una *capability* per accedere ad un oggetto

La stringa nella memoria di un processo:

`"/etc/passwd"` identifica univocamente un oggetto (file) nel sistema ma non specifica i diritti di accesso e quindi non è una capability

I valori:

`"/etc/passwd" O_RDWR` identificano un oggetto e un insieme di diritti di accesso (lettura e scrittura in UNIX). Non si tratta tuttavia di una capability dato che il possesso da parte del processo di questi valori non indica se quell'accesso sia autorizzato dal SO

- Supponiamo che processo esegua la system call `open` con successo (cfr. Icidi UNIX) :

```
int fd = open("/etc/passwd", O_RDWR);
```

- La variabile **fd** ora contiene l'indice di un file descriptor nella tabella dei file aperti del processo. Questo file descriptor è una *capability* : infatti la sua esistenza nella tabella dei file aperti del processo è sufficiente per sapere che il processo ha veramente un accesso legittimo all'oggetto.
- Un aspetto fondamentale è che la tabella dei file aperti è mantenuta in memoria di kernel (associata al processo) e non può essere direttamente manipolata dal processo utente, come si vedrà nella parte UNIX del corso.

Protezione

- Il sistema di protezione richiede l'esistenza di *più modi di funzionamento della CPU*:
 - *supervisor mode* (detto anche 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 SO prima della cessione del controllo ad un processo di utente.
- Possono esservi anche più modi : *protection rings* (ad es. 0 – kernel, 1,2 - programmi che accedono ad I/O, 3 – programmi utente)
 - ring -1 per hypervisor (vedi più avanti nella sezione sulla virtualizzazione)
- 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 nello scheduling round-robin)

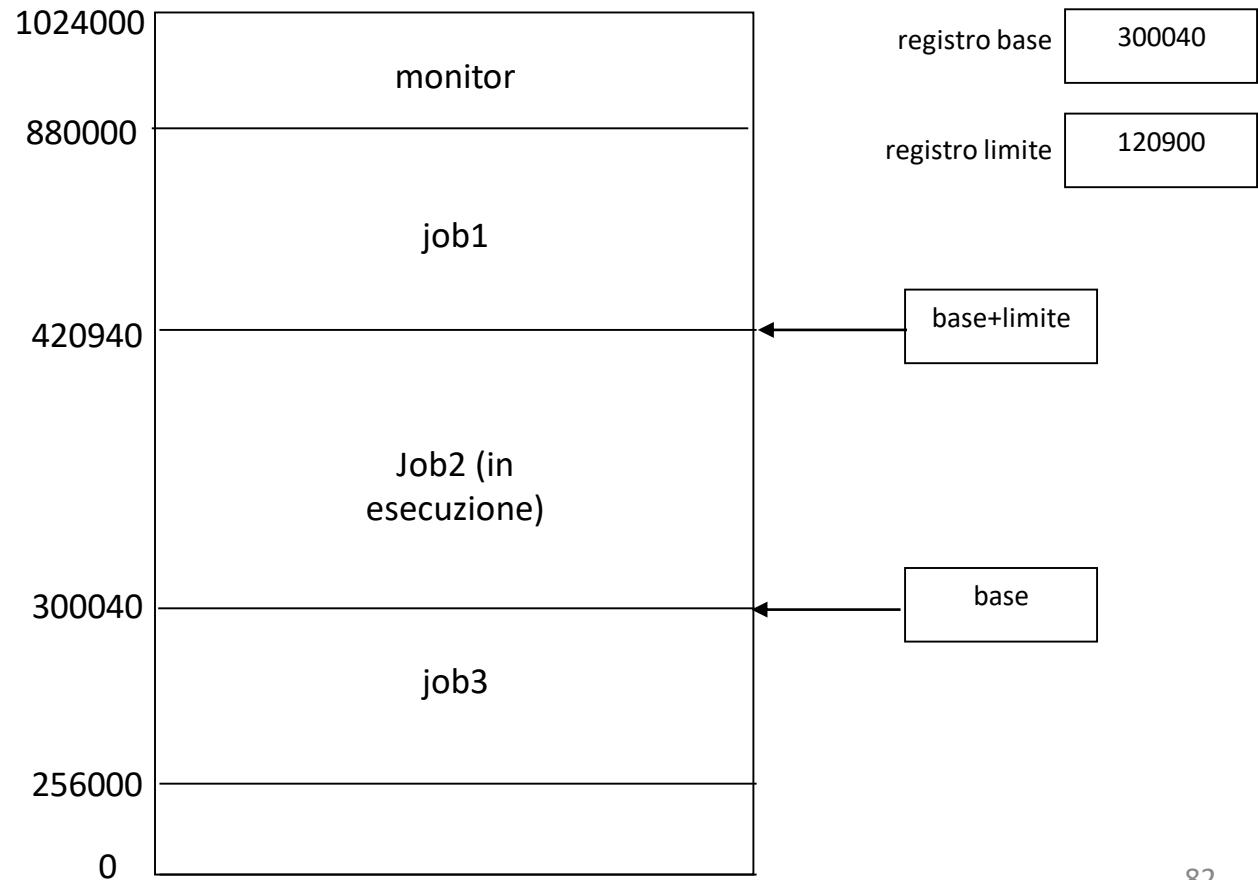
Protezione della memoria con registri limite

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

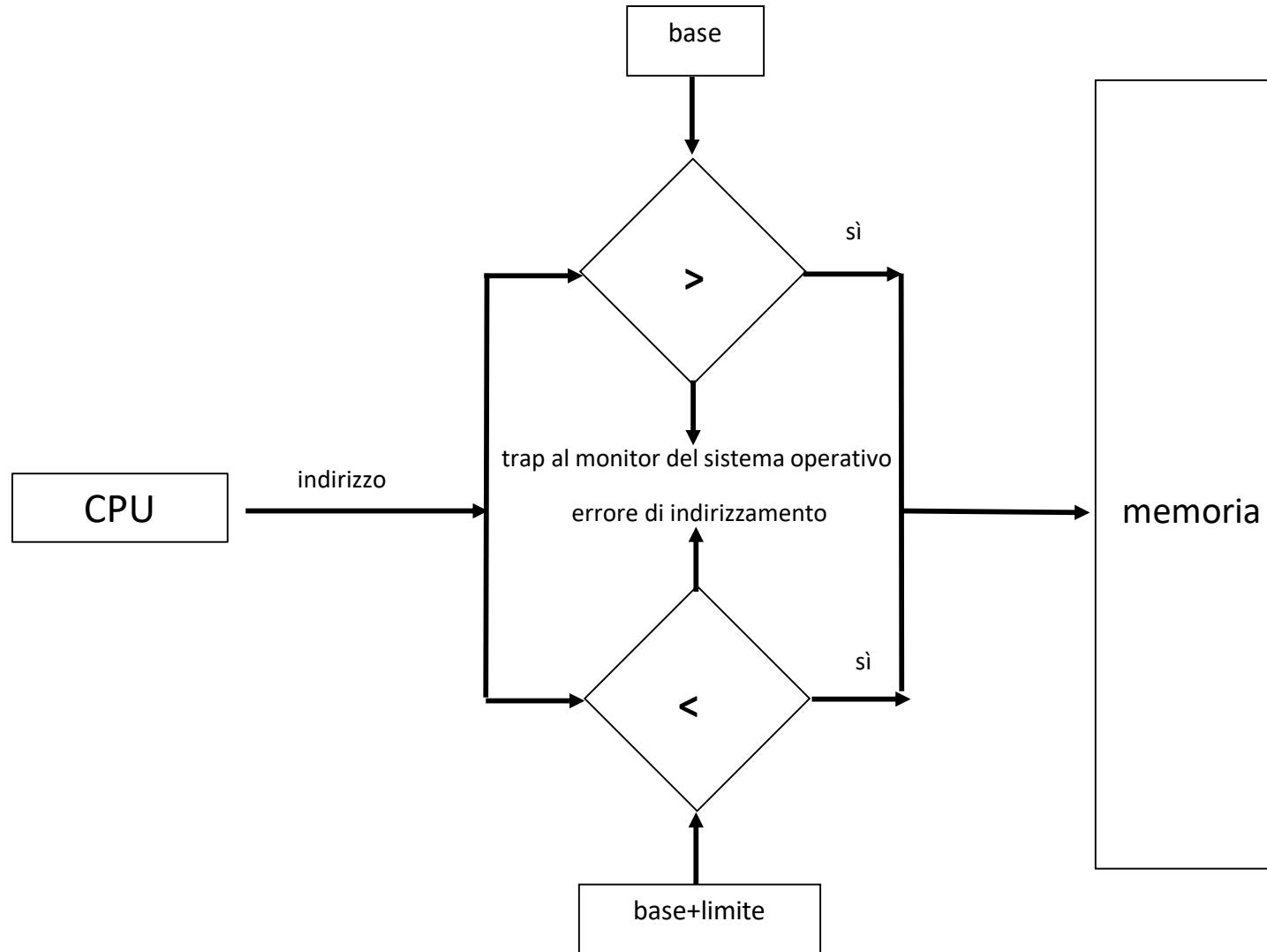
N.B.

*E' un esempio elementare
di protezione della
memoria*

*Le CPU moderne
incorporano una MMU
(memory management unit)
che viene gestita dal S.O.
per ottenere una protezione
della memoria più fine
(pagine da 4k) e flessibile
all'interno della gestione
della memoria virtuale*

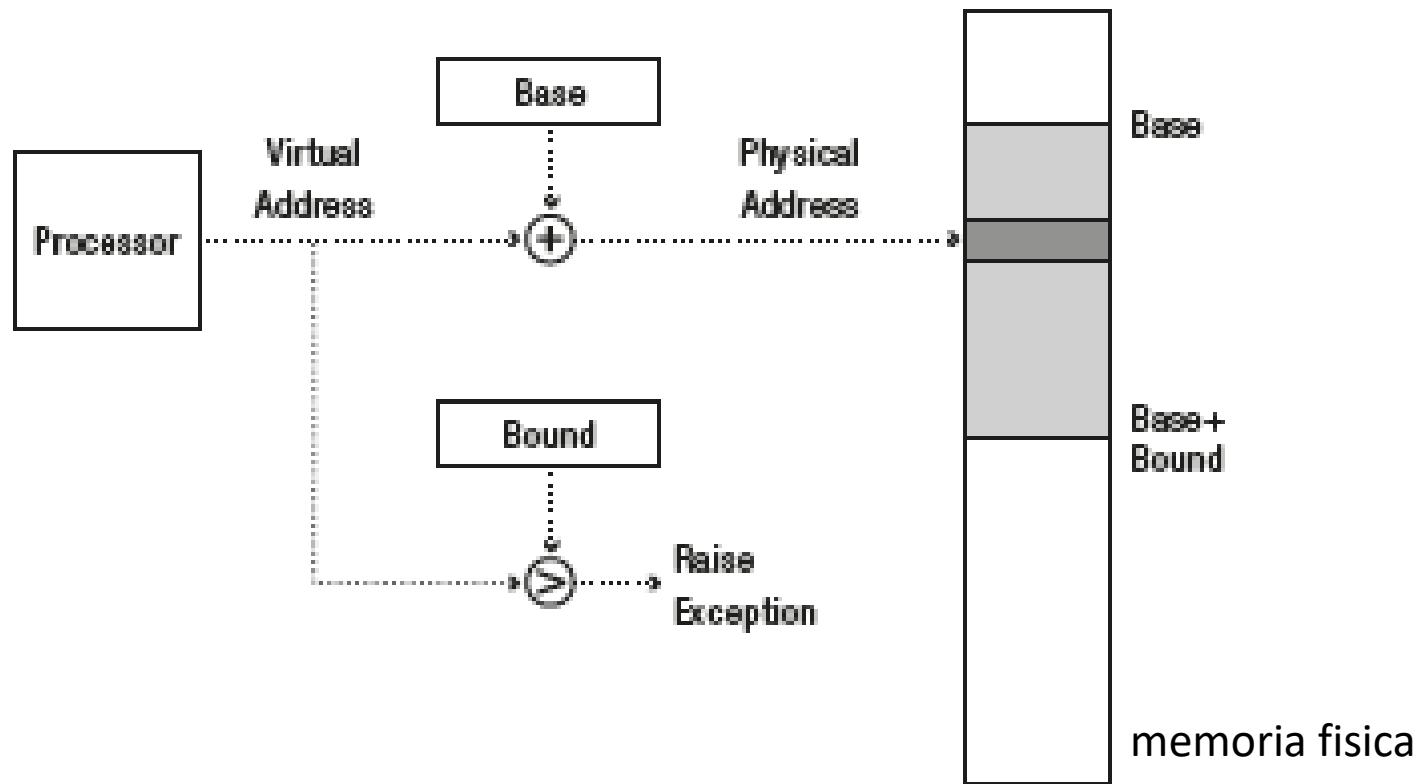


Protezione della memoria con registri limite



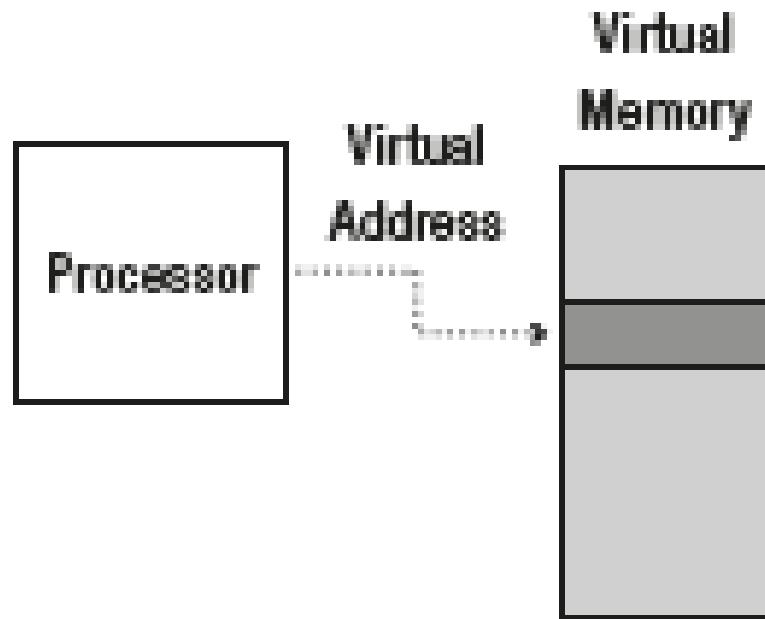
Spazio di indirizzamento virtuale e spazio di indirizzamento fisico

- Il processore genera un indirizzo (virtuale) che non coincide con l'indirizzo fisico di accesso alla memoria



Astrazione memoria virtuale

- Il processore accede per conto del processo ad uno spazio di indirizzamento *logico*



Memoria virtuale: motivazioni

SisOp
2020/21



- Il codice deve essere in memoria per essere eseguito, ma raramente l'intero programma è utilizzato
 - Codice di gestione errori, routine non comuni, grandi strutture dati
- L'intero codice del programma non è necessario allo stesso tempo
- Si consideri la capacità di eseguire programmi parzialmente caricati
 - Il programma non è più limitato dai limiti della memoria fisica
 - Ogni programma richiede meno memoria mentre esegue -> più programmi eseguono allo stessi tempo
 - Migliora l'utilizzazione della CPU e il throughput senza che aumenti il response o il turnaround time
 - Meno I/O è necessario per caricare o avvicendare (swap) i programmi in memoria -> ogni programma utente esegue più rapidamente

Memoria virtuale

SisOp
2020/21

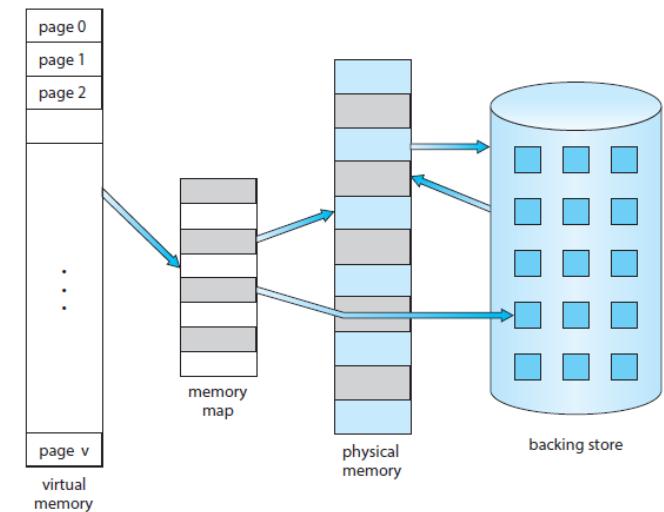


- **Memoria virtuale** – separazione della memoria logica dei programmi dalla memoria fisica
 - Solo una parte del programma è necessaria in memoria per l'esecuzione
 - Lo spazio di indirizzamento logico può quindi essere molto più grande dello spazio di indirizzamento fisico
 - Consente agli spazi di indirizzamento di essere condivisi da più processi
 - Consente un più efficiente creazione dei processi
 - Più programmi eseguono concorrentemente
 - Meno I/O è necessario per caricare o avvicendare (swap) processi

Memoria virtuale



- **Spazio di indirizzamento virtuale** – vista logica di come il processo è memorizzato in memoria
 - Di solito inizia all'indirizzo 0, indirizzi contigui fino alla fine dello spazio
 - La memoria fisica è invece organizzata in (page) frame
 - La MMU (Memory Management Unit – un tempo separata, ora integrata con la CPU) deve mappare gli indirizzi logici su quelli fisici (utilizzando una mappa della memoria)
- La memoria virtuale può essere implementata con:
 - **Demand segmentation** (segmentazione su richiesta – ormai obsoleta)
 - **Demand paging** (paginazione su richiesta)



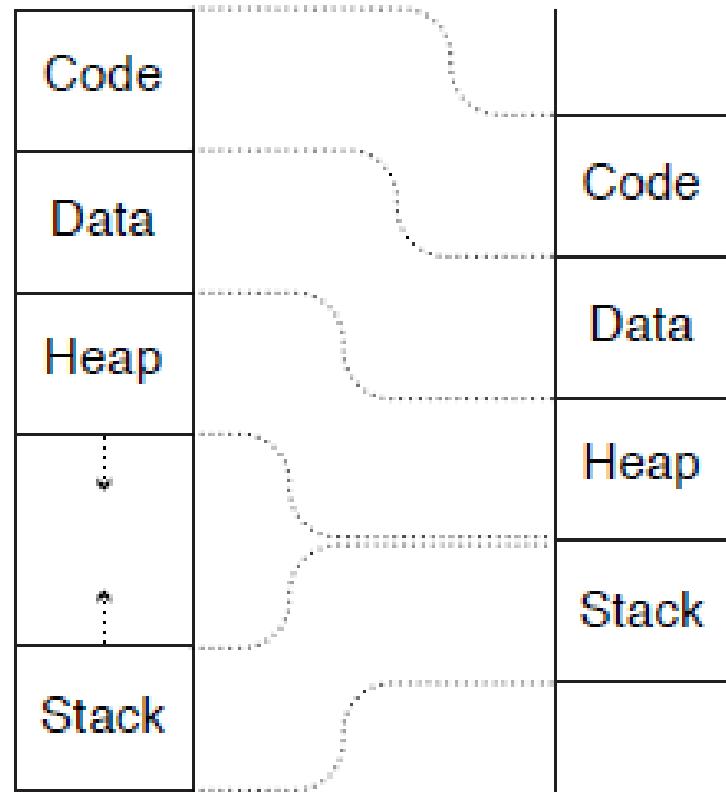
La memoria virtuale può essere più grande di quella fisica

Indirizzi virtuali

- La protezione mediante indirizzi base e limite presenta molti inconvenienti
- → Il processore utilizza indirizzi virtuali e il processo ritiene di disporre dell'intero spazio di indirizzamento
- La traduzione da indirizzi virtuali a indirizzi fisici viene effettuata in hardware (MMU)
- La tabella di traduzione viene predisposta dal SO e modificata al cambio di contesto
- Il processore vede solo indirizzi virtuali
- Quali implicazioni con multiprogrammazione?

Virtual Addresses
(Process Layout)

Physical
Memory



Esempio di spazio di indirizzamento di un processo Linux

Indirizzi virtuali

```

00400000-004f0000 r-xp 00000000 08:01 2373800      /bin/bash
006ef000-006f0000 r--p 000ef000 08:01 2373800      /bin/bash
006f0000-006f9000 rw-p 000f0000 08:01 2373800      /bin/bash
006f9000-006ff000 rw-p 00000000 00:00 0
00882000-00b96000 rw-p 00000000 00:00 0
[heap]
7f54815ce000-7f54815d8000 r-xp 00000000 08:01 797270  /lib/x86_64-linux-gnu/libnss_files-2.19.so
7f54815d8000-7f54817d7000 ---p 0000a000 08:01 797270  /lib/x86_64-linux-gnu/libnss_files-2.19.so
7f54817d7000-7f54817d8000 r--p 00009000 08:01 797270  /lib/x86_64-linux-gnu/libnss_files-2.19.so
7f54817d8000-7f54817d9000 rw-p 0000a000 08:01 797270  /lib/x86_64-linux-gnu/libnss_files-2.19.so
7f54817d9000-7f54817e4000 r-xp 00000000 08:01 797275  /lib/x86_64-linux-gnu/libnss_nis-2.19.so
7f54817e4000-7f54819e3000 ---p 0000b000 08:01 797275  /lib/x86_64-linux-gnu/libnss_nis-2.19.so
7f54819e3000-7f54819e4000 r--p 0000a000 08:01 797275  /lib/x86_64-linux-gnu/libnss_nis-2.19.so
7f54819e4000-7f54819e5000 rw-p 0000b000 08:01 797275  /lib/x86_64-linux-gnu/libnss_nis-2.19.so
7f54819e5000-7f54819fc000 r-xp 00000000 08:01 797288  /lib/x86_64-linux-gnu/libnsl-2.19.so
7f54819fc000-7f5481bfb000 ---p 00017000 08:01 797288  /lib/x86_64-linux-gnu/libnsl-2.19.so
7f5481bfb000-7f5481bfc000 r--p 00016000 08:01 797288  /lib/x86_64-linux-gnu/libnsl-2.19.so
7f5481bfc000-7f5481bfd000 rw-p 00017000 08:01 797288  /lib/x86_64-linux-gnu/libnsl-2.19.so
7f5481bfd000-7f5481bff000 rw-p 00000000 00:00 0
[heap]
7f5481bff000-7f5481c08000 r-xp 00000000 08:01 797277  /lib/x86_64-linux-gnu/libnss_compat-2.19.so
7f5481c08000-7f5481e07000 ---p 00009000 08:01 797277  /lib/x86_64-linux-gnu/libnss_compat-2.19.so
7f5481e07000-7f5481e08000 r--p 00008000 08:01 797277  /lib/x86_64-linux-gnu/libnss_compat-2.19.so
7f5481e08000-7f5481e09000 rw-p 00009000 08:01 797277  /lib/x86_64-linux-gnu/libnss_compat-2.19.so
7f5481e09000-7f5482385000 r--p 00000000 08:01 1835287  /usr/lib/locale/locale-archive
7f5482385000-7f5482543000 r-xp 00000000 08:01 797293  /lib/x86_64-linux-gnu/libc-2.19.so
7f5482543000-7f5482743000 ---p 001be000 08:01 797293  /lib/x86_64-linux-gnu/libc-2.19.so
7f5482743000-7f5482747000 r--p 001be000 08:01 797293  /lib/x86_64-linux-gnu/libc-2.19.so
7f5482747000-7f5482749000 rw-p 001c2000 08:01 797293  /lib/x86_64-linux-gnu/libc-2.19.so
7f5482749000-7f548274e000 rw-p 00000000 00:00 0
[heap]
7f548274e000-7f5482751000 r-xp 00000000 08:01 797289  /lib/x86_64-linux-gnu/libdl-2.19.so
7f5482751000-7f5482950000 ---p 00003000 08:01 797289  /lib/x86_64-linux-gnu/libdl-2.19.so
7f5482950000-7f5482951000 r--p 00002000 08:01 797289  /lib/x86_64-linux-gnu/libdl-2.19.so
7f5482951000-7f5482952000 rw-p 00003000 08:01 797289  /lib/x86_64-linux-gnu/libdl-2.19.so
7f5482952000-7f5482977000 r-xp 00000000 08:01 786507  /lib/x86_64-linux-gnu/libtinfo.so.5.9
7f5482977000-7f5482b76000 ---p 00025000 08:01 786507  /lib/x86_64-linux-gnu/libtinfo.so.5.9
7f5482b76000-7f5482b7a000 r--p 00024000 08:01 786507  /lib/x86_64-linux-gnu/libtinfo.so.5.9
7f5482b7a000-7f5482b7b000 rw-p 00028000 08:01 786507  /lib/x86_64-linux-gnu/libtinfo.so.5.9
7f5482b7b000-7f5482b9e000 r-xp 00000000 08:01 797282  /lib/x86_64-linux-gnu/ld-2.19.so
7f5482d52000-7f5482d7c000 r--p 00000000 08:01 1855420  /usr/share/locale-langpack/it/LC_MESSAGES/bash.mo
7f5482d7c000-7f5482d80000 rw-p 00000000 00:00 0
[heap]
7f5482d96000-7f5482d9d000 r-s 00000000 08:01 1973187  /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
7f5482d9d000-7f5482d9e000 r--p 00022000 08:01 797282  /lib/x86_64-linux-gnu/ld-2.19.so
7f5482d9e000-7f5482d9f000 rw-p 00023000 08:01 797282  /lib/x86_64-linux-gnu/ld-2.19.so

```

`cat /proc/PID/maps`

7f5482d9f000-7f5482da0000 rw-p 00000000 00:00 0

[stack]

7ffffd2b08000-7ffffd2b29000 rw-p 00000000 00:00 0

[vdso]

7ffffd2ba2000-7ffffd2ba4000 r-xp 00000000 00:00 0

[vsyscall]

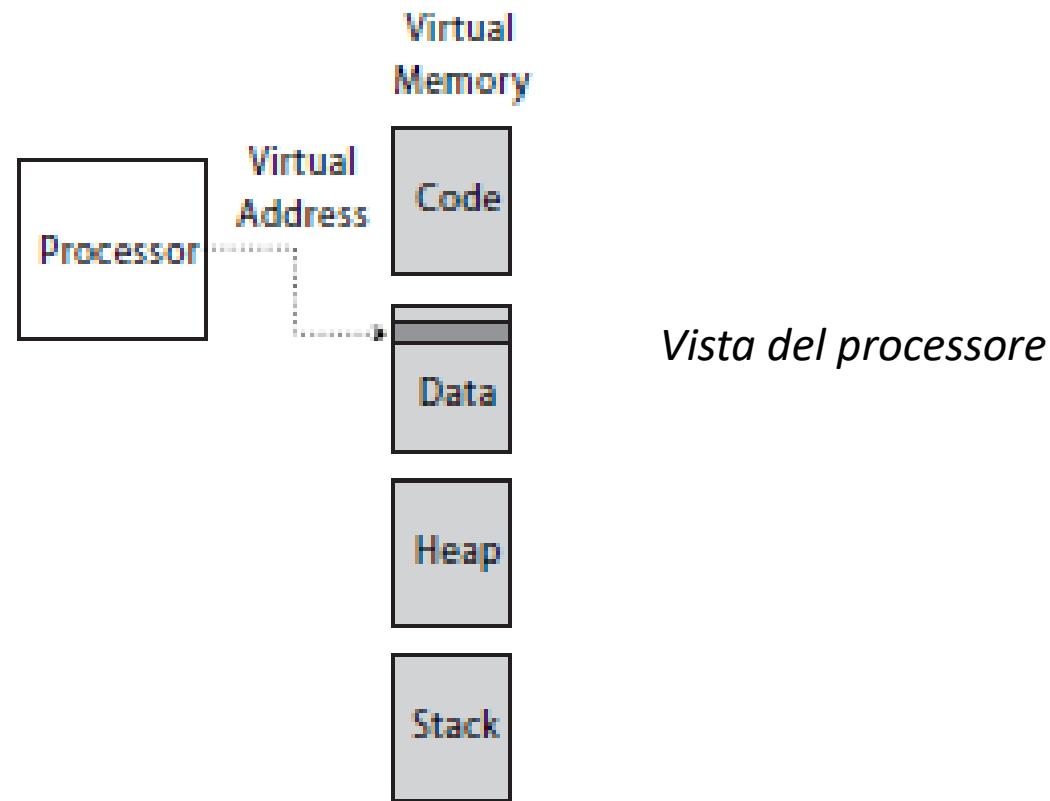
fffffffff600000-fffffffff601000 r-xp 00000000 00:00 0

Segmentazione

SisOp
2020/21

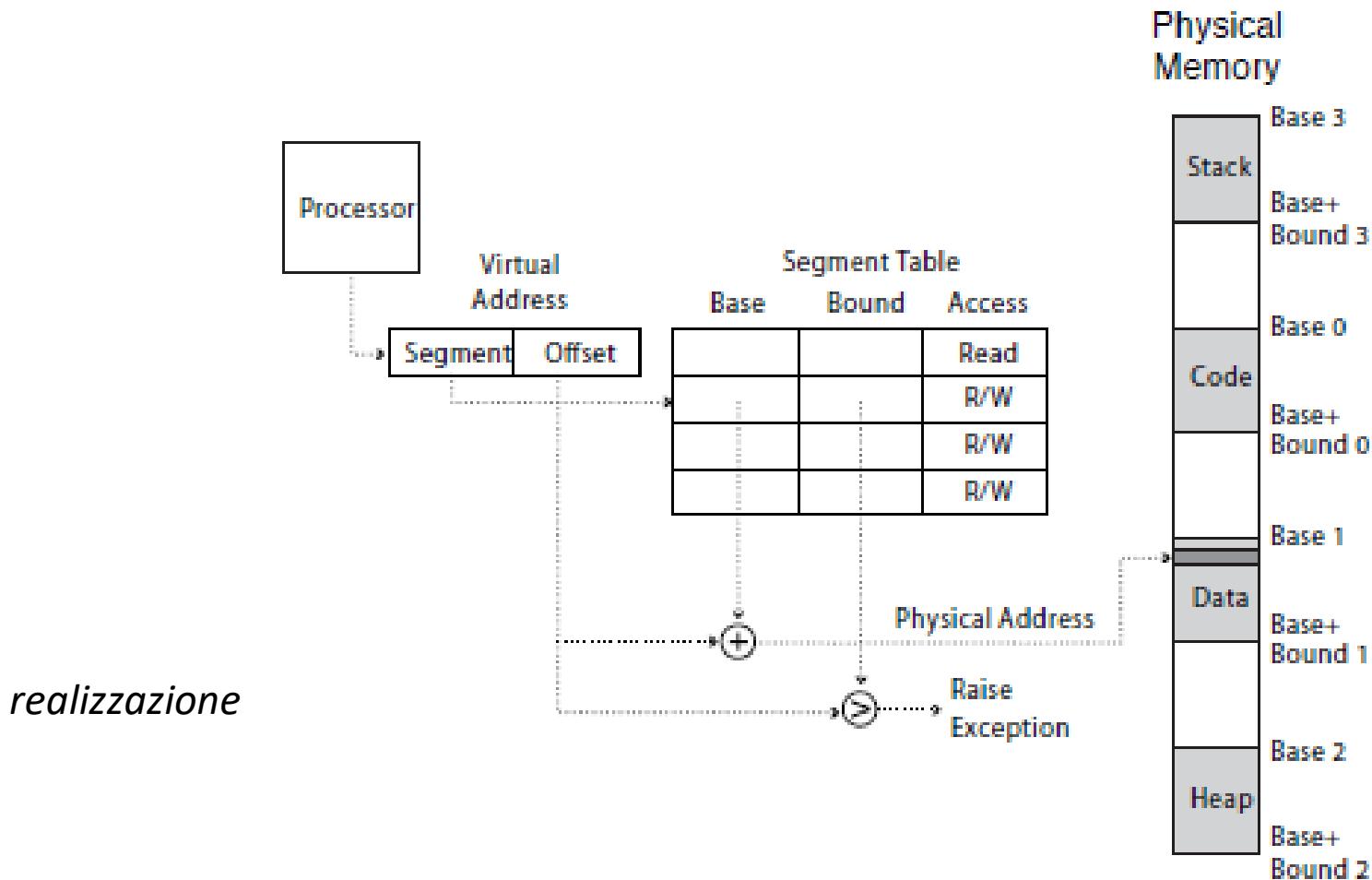


- Traduzione indirizzi con tabella dei segmenti



Segmentazione

- Traduzione indirizzi con tabella dei segmenti



Paginazione

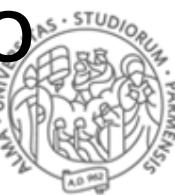
SisOp
2020/21



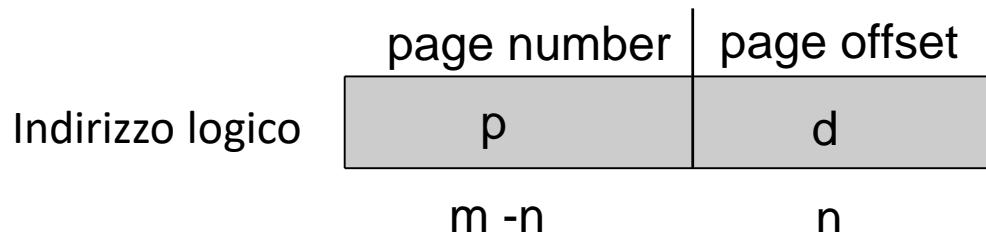
- Lo spazio di indirizzamento fisico di un processo può essere non contiguo; al processo viene allocata memoria fisica appena ognivolta questa è disponibile
 - Si evita la frammentazione esterna (un problema della allocazione contigua della memoria, caricando e rimuovendo processi, si frammenta lo spazio libero, cosicchè non c'è alcun area contigua abbastanza grande per la richiesta)
 - Evita di avere blocchi di memoria di dimensione diversa
- Divide la memoria fisica in blocchi di dimensione fissa detti (page) **frame**
 - La dimensione è una potenza di 2, tra 512 byte e 16 Mbyte, ad es. 4KB
- Divide la memoria logica in blocchi detti **pagine**
- Il SO deve tenere traccia di tutti i frame liberi
- Per eseguire un programma di dimensione **N** pagine, è necessario trovare **N** frame liberi e caricare il programma
- Va predisposta una **page table** per tradurre gli indirizzi logici in quelli fisici per ogni processo
- Anche la memoria ausiliaria (backing store) è divisa in pagine
- C'è ancora la frammentazione interna: l'ultima pagina/frame è tipicamente utilizzata/o solo parzialmente

Schema di traduzione dell'indirizzo

SisOp
2020/21



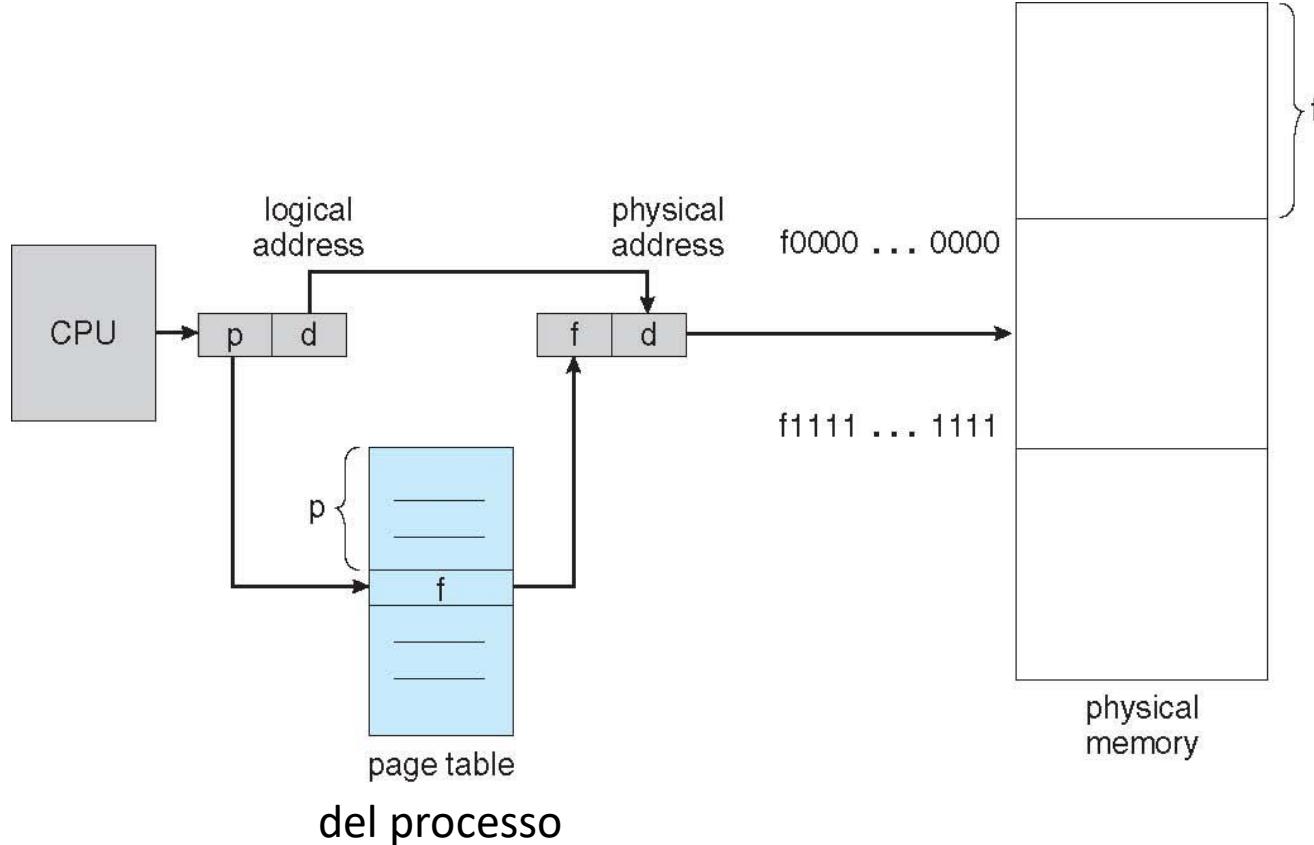
- L'indirizzo generato dalla CPU è diviso in:
 - **Page number (p)** – usato come un indice nella **page table** che contiene l'indirizzo di base di ogni page frame nella memoria fisica
 - **Page offset (d)** – combinato con l'indirizzo di base per definire l'indirizzo di memoria fisica da inviare alla MMU



Per uno spazio di indirizzamento di dimensione 2^m con dimensione delle pagine di 2^n

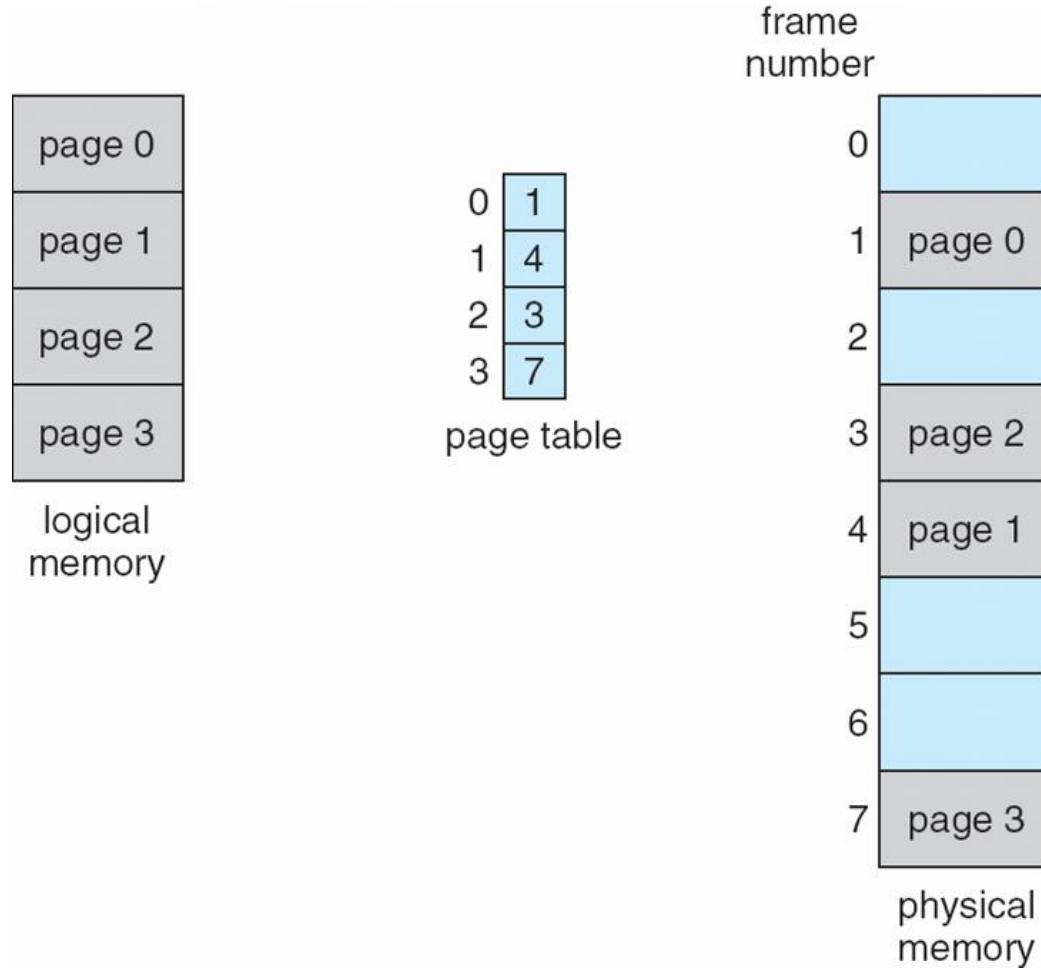
Hardware di paginazione

SisOp
2020/21



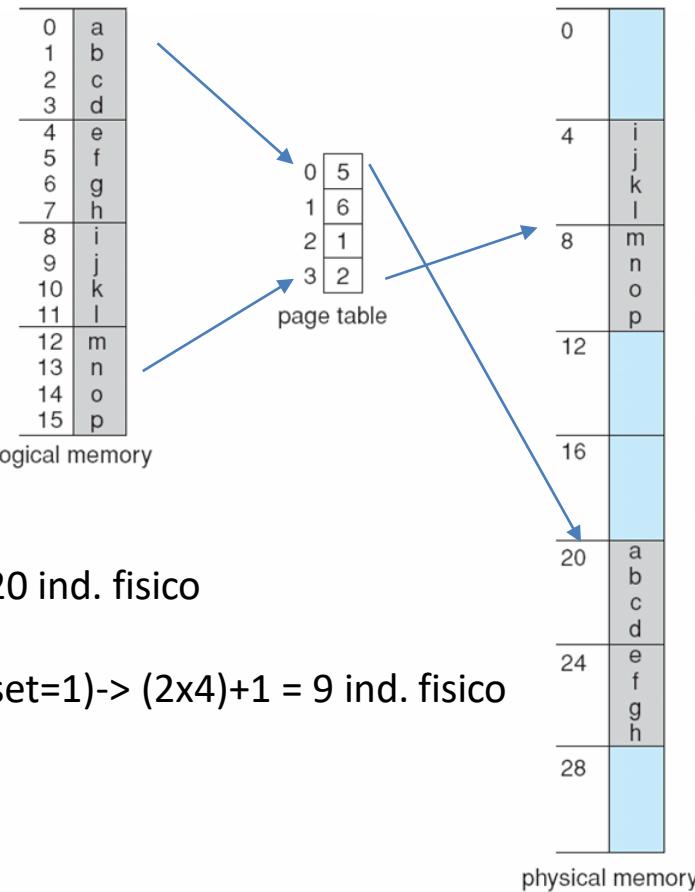
Modello di paginazione delle memoria logica e fisica

SisOp
2020/21



Esempio di paginazione

SisOp
2020/21



Indirizzo logico 0 -> $(5 \times 4) + 0 = 20$ ind. fisico

Indirizzo logico 13 (page=3,offset=1)-> $(2 \times 4) + 1 = 9$ ind. fisico

$n=2$ e $m=4$ 32-byte di memoria che contiene 8 pagine di 4-byte

Indirizzo logico

page number	page offset
p	d

m - n n

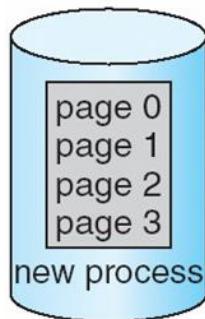
Frame liberi

SisOp
2020/21



free-frame list

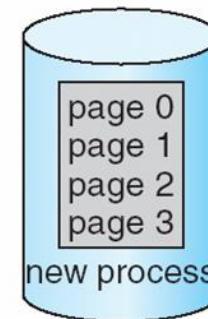
14
13
18
20
15



(a)

free-frame list

15



0	14
1	13
2	18
3	20

new-process page table



(b)

Prima della allocazione

Dopo l'allocazione

Possibile implementazione della Page Table

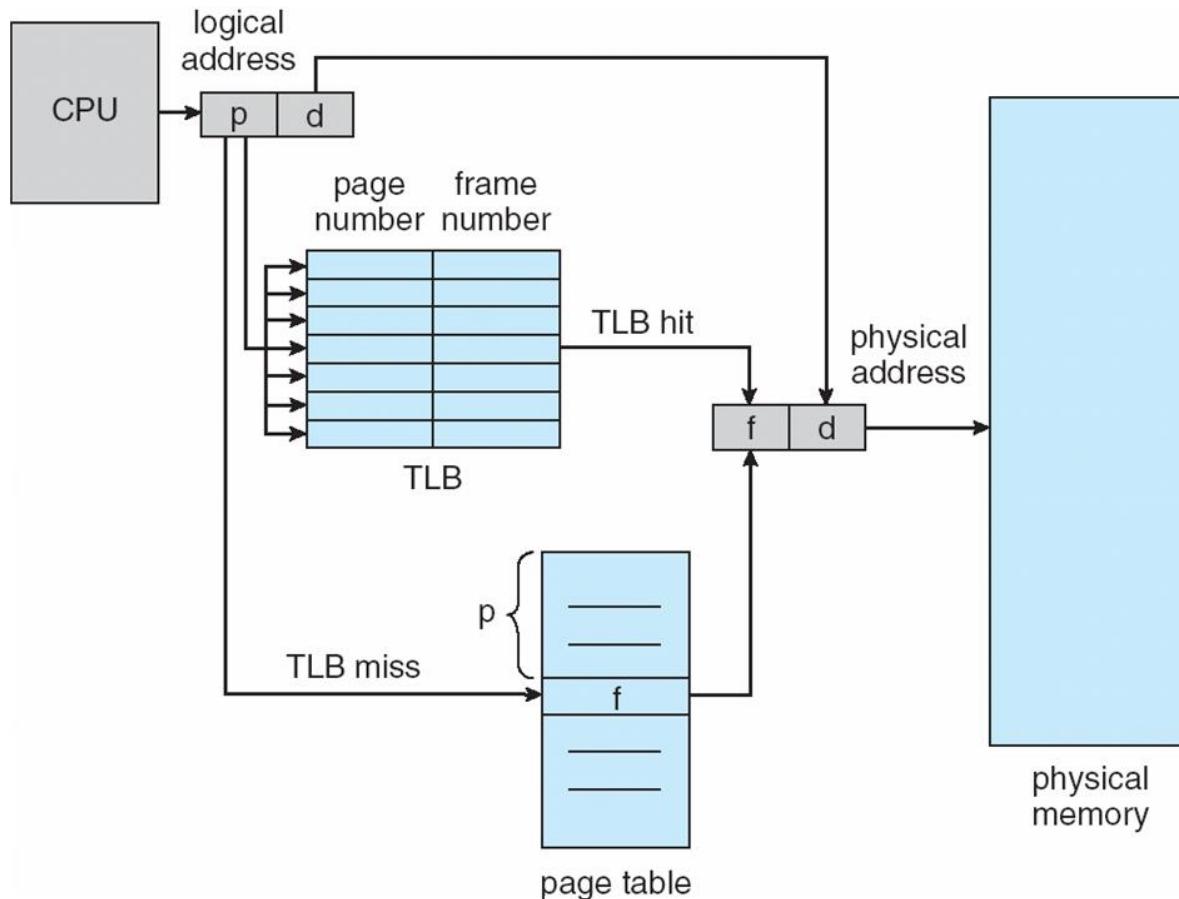
SisOp
2020/21



- La tabella delle pagine è mantenuta in memoria
- Il **Page-table base register (PTBR)** punta alla tabella delle pagine
- Il **Page-table length register (PTLR)** indica la dimensione della page table
- In questo schema ogni accesso a dati/instruzioni richiede due accessi alla memoria
 - Uno per la page table e uno per dati/instruzioni
- Può essere risolto con l'uso di una special cache HW di ricerca chiamata **memoria associativa** o **translation look-aside buffers (TLBs)**

HW di paginazione con TLB

SisOp
2020/21



Protezione della memoria nella paginazione

SisOp
2020/21

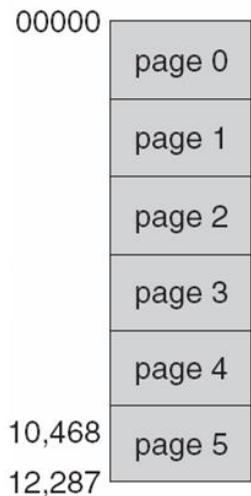


La protezione della memoria è implementata associando un bit di protezione ad ogni frame per indicare se è consentito un accesso read-only o read-write

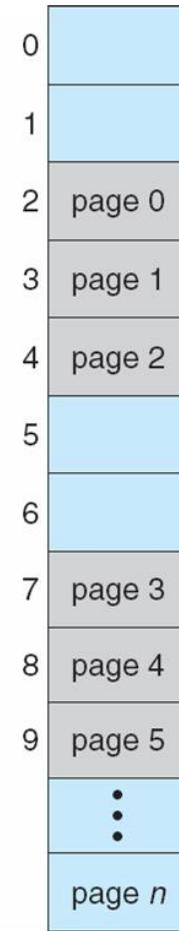
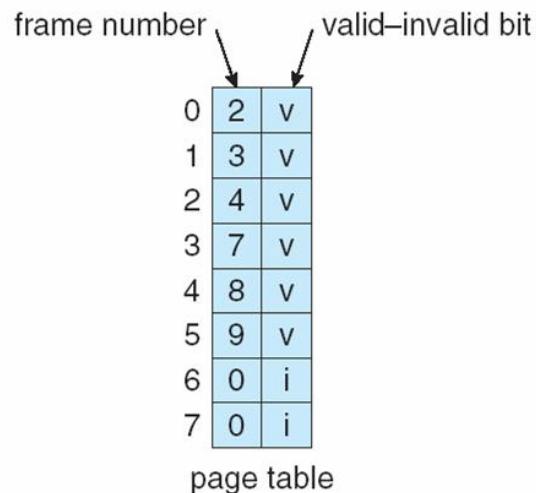
- Possono esserci altri bit per indicare che la pagina è execute-only, etc
- Un bit **Valid-Invalid** è attaccato ad ogni voce della page table:
 - “valid” indica che la pagina associata è nello spazio di indirizzamento del processo e presente in memoria
 - “invalid” indica che la pagina non è nello spazio di indirizzamento del processo oppure è valida ma attualmente in memoria secondaria (un eventuale accesso provoca un'eccezione di **page fault**)
 - oppure si utilizza il **page-table length register (PTLR)** per verificare che l'indirizzo logico si trovi nell'intervallo valido (numero di pagine) del processo
- Ogni violazione causa un'eccezione (trap) al kernel

Valid (v)/Invalid (i) bit nella tabella delle pagine

SisOp
2020/21

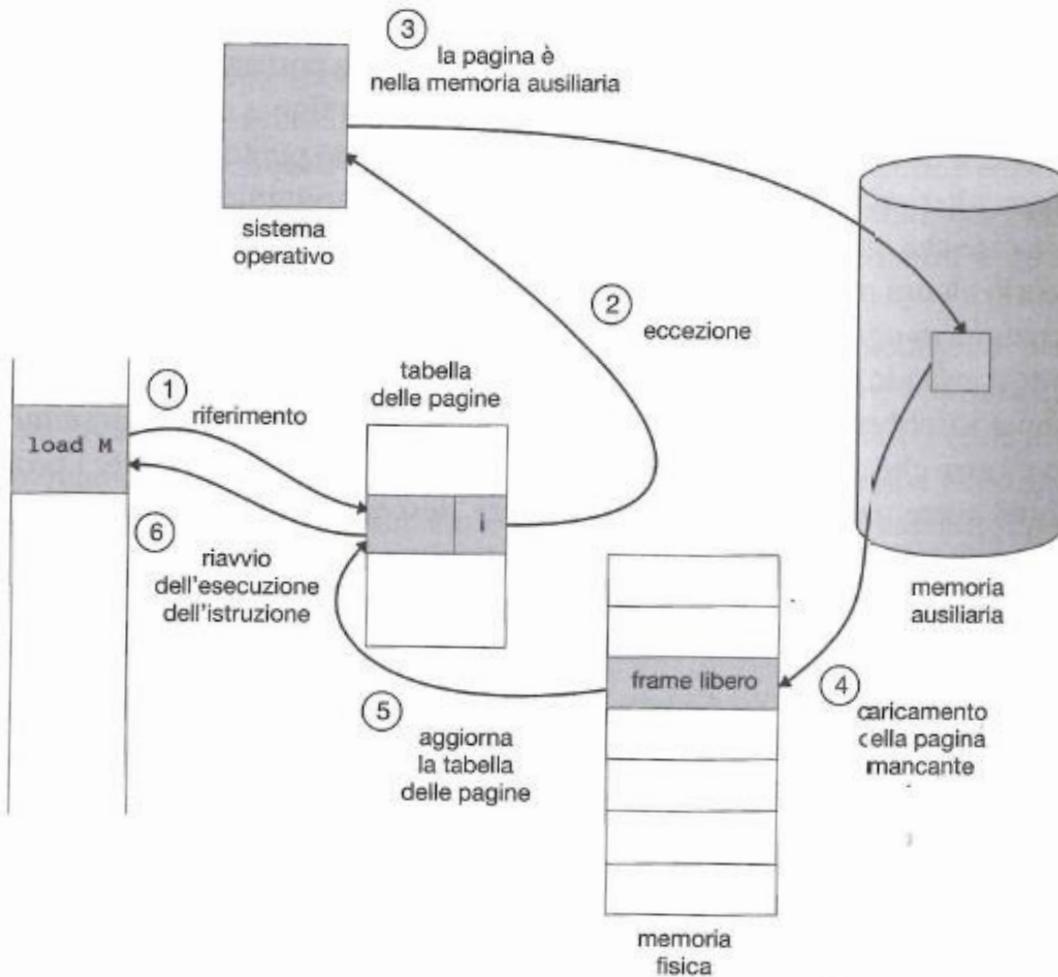


Memoria logica



Memoria fisica

Gestione page fault



Pagine condivise

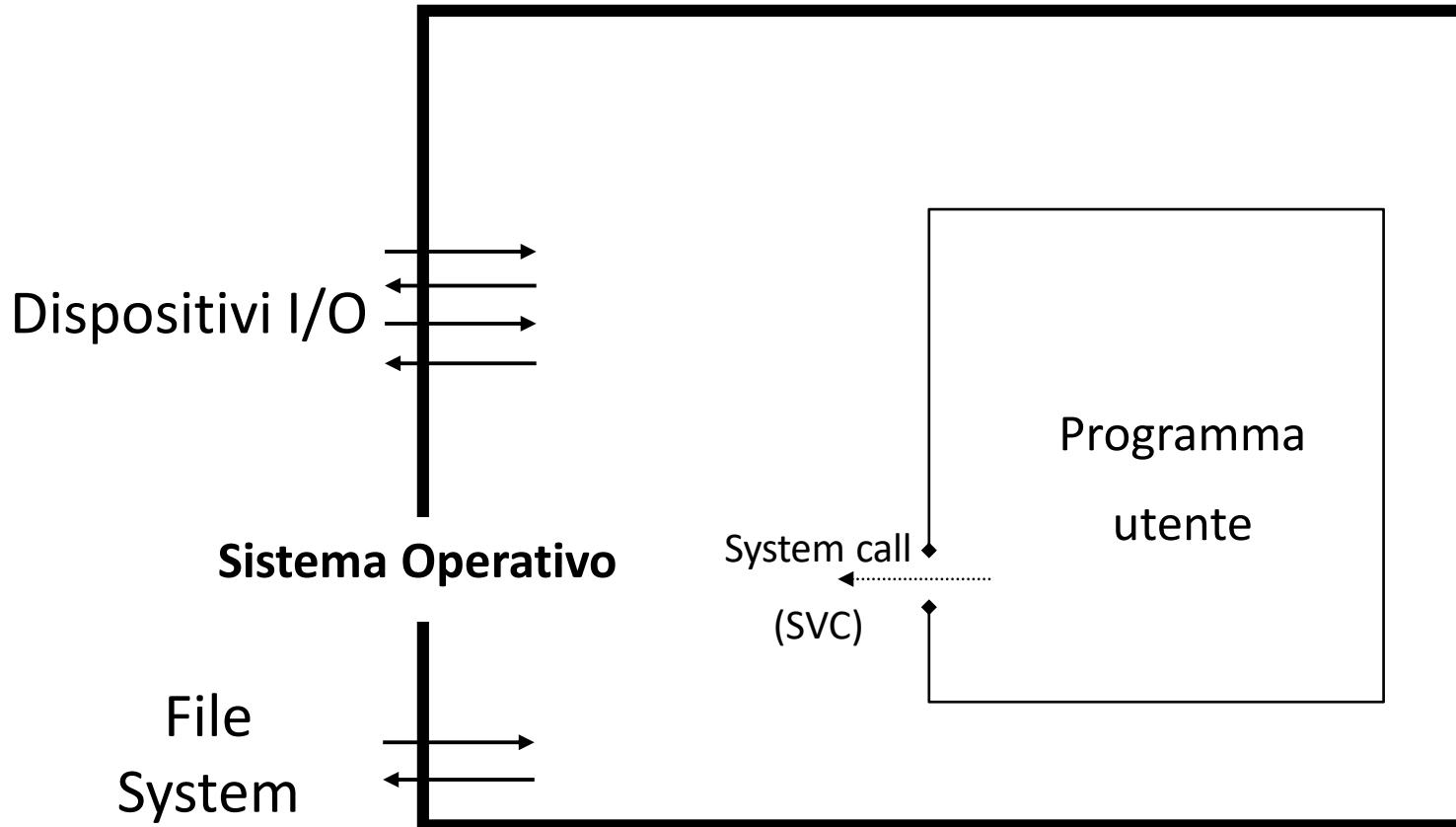
SisOp
2020/21



- **Codice condiviso**
 - Una sola copia di codice read-only (**rientrante**) condiviso tra processi (ovvero text editor, compilatori, sistemi a finestra)
 - Simile a thread multipli che condividono lo stesso spazio di indirizzamento del processo (vedi più avanti)
 - Anche utile per la comunicazione tra processi se è consentita la condivisione di pagine read-write (possibile anche in UNIX/Linux, ma nel corso vedremo solo la comunicazione alternativa basata sullo scambio di messaggi)
- **Codice e dati privati**
 - Ogni processo mantiene una copia separata del codice e dei dati
 - Le pagine per il codice e i dati privati possono apparire ovunque nello spazio di indirizzamento logico del processo

Confinamento del programma utente

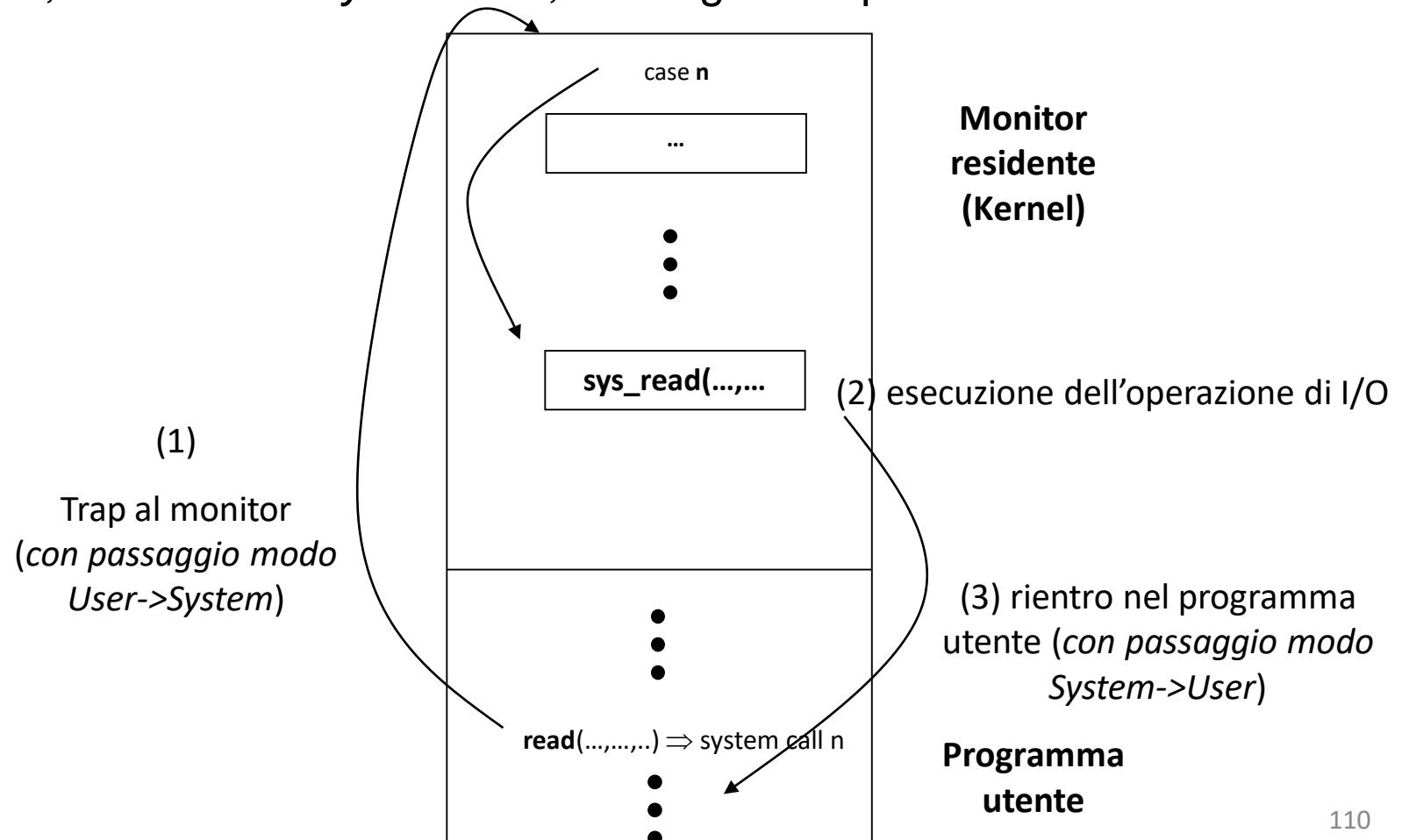
SisOp
2020/21



Il programma utente non può accedere direttamente alle risorse del sistema;
accede solo attraverso l'intermediazione del SO

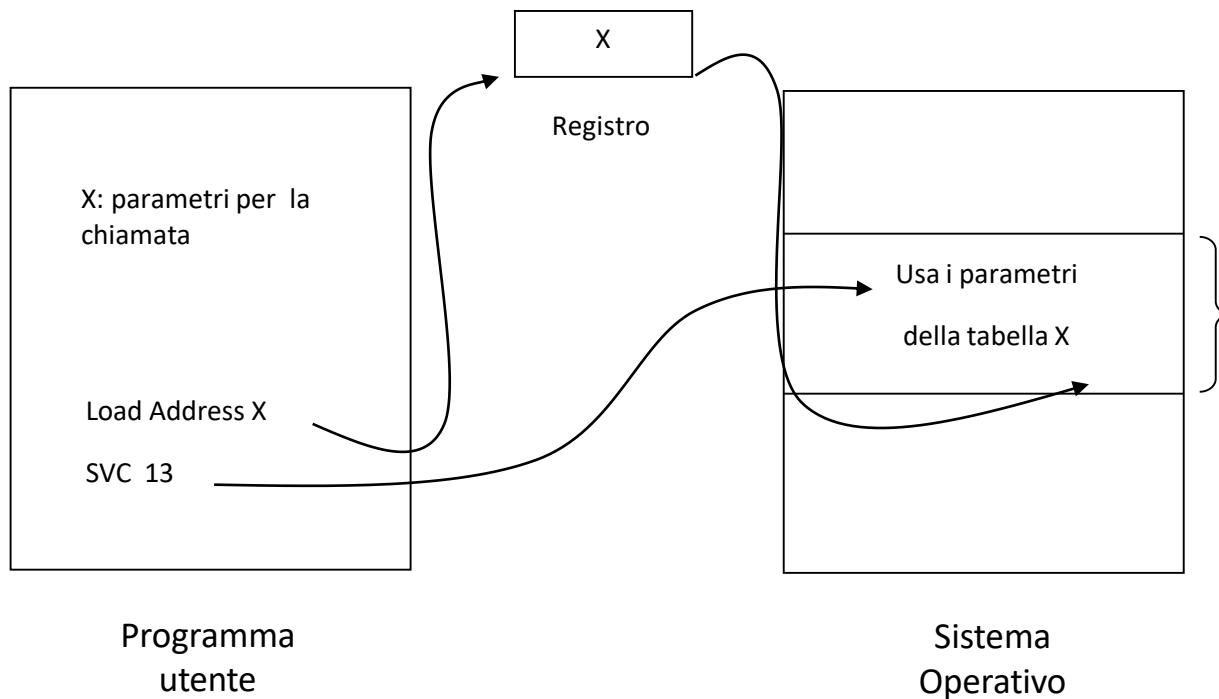
Esecuzione delle operazioni di I/O

- 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 (ad es. per x86 istruzione di interrupt software INT in passato – ad. es int 80h per Linux a 32bit, ora SYSENTER/SYSEXIT-Intel e SYSCALL/SYSRET-Amd – cfr. <https://wiki.osdev.org/SYSENTER>) inserite in funzioni richiamabili da linguaggi di alto livello.
Nei linguaggi di alto livello sono tipicamente mascherate dal supporto a tempo di esecuzione all'interno di librerie (ad es. la libc di UNIX/Linux contiene tutte le system call del S.O.)
- 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
(IPC – interprocess communication)

Una tabella ricercabile delle System Call Linux con i link ai corrispondenti sorgenti del kernel
<https://filippo.io/linux-syscall-table/>

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

NB: Le system call (API) di UNIX/Linux saranno oggetto della seconda parte del corso

Windows API

SisOp
2020/21



- **Windows API** è il nome dato da Microsoft al core set di application programming interfaces disponibile nei Microsoft Windows operating systems.
- E' progettato per l'uso da programmi C/C++ ed è il modo più diretto di interagire con il sistema Windows per le applicazioni.
- L'accesso a più basso livello ad un sistema Windows, soprattutto richiesto per i device drivers, è fornito dal Windows Driver Model nelle versioni correnti di Windows.
- Un software development kit (SDK) è disponibile per Windows, che fornisce documentazione e strumenti per abilitare gli sviluppatori a creare software usando la Windows API e le tecnologie Windows associate.

Il materiale è tratto da da https://microsoft.fandom.com/wiki/Windows_API
e da https://en.wikipedia.org/wiki/Windows_API

Panoramica API Windows

Le funzionalità fornite dalla Windows API possono essere raggruppate in 7 categorie:

1. Base Services

Forniscono accesso alle risorse fondamentali disponibili in un sistema Windows. Sono incluse cose come [file systems](#), dispositivi, [processes](#) e [threads](#), accesso al [Windows registry](#), e [error handling](#). Queste funzioni risiedono in kernel32.dll e advapi32.dll su 32-bit Windows.

2. Graphics Device Interface

Fornisce la funzionalità per emettere content grafici a [monitors](#), [printers](#) e altri [output devices](#). Risiede in gdi32.dll su 32-bit Windows.

3. User Interface

Fornisce la funzionalità per gestire [windows](#) e principali controlli di base, come [buttons](#) e [scrollbars](#), ricevere input da mouse e tastiera, e altre funzioni associate con la parte [GUI](#) di Windows. Questa unità funzionale risiede in [user32.dll](#) su 32-bit Windows.

A partire da [Windows XP](#), i controlli di base risiedono in [comctl32.dll](#), assieme ai controlli comuni.

Overview Windows API (cont.)

SisOp
2020/21



4. Common Dialog Box Library

Provides applications the standard [dialog boxes](#) for opening and saving files, choosing color and font, etc. The library resides in a file called commdlg.dll on 16-bit Windows, and comdlg32.dll on 32-bit Windows. It is grouped under the *User Interface* category of the API.

5. Common Control Library

Gives applications access to some advanced controls provided by the operating system. These include things like [status bars](#), progress bars, [toolbars](#) and [tabs](#). The library resides in a [DLL](#) file called commctrl.dll on 16-bit Windows, and comctl32.dll on 32-bit Windows. It is grouped under the *User Interface* category of the API.

6. Windows Shell

Component of the Windows API allows applications to access the functionality provided by the [operating system shell](#), as well as change and enhance it. The component resides in shell.dll on 16-bit Windows, and shell32.dll and shlwapi.dll on 32-bit Windows. It is grouped under the *User Interface* category of the API.

7. Network Services

Give access to the various [networking](#) capabilities of the operating system. Its sub-components include NetBIOS, Winsock, NetDDE, RPC and many others.



Versioni Windows API

Quasi ogni nuova versione di Microsoft Windows ha introdotto le proprie aggiunte e modifiche alle API di Windows. Il nome dell'API, tuttavia, è rimasto coerente tra le diverse versioni di Windows, e le modifiche del nome sono state limitate alle principali modifiche architetturali e di piattaforma per Windows. Alla fine Microsoft ha cambiato il nome dell'allora attuale famiglia di API Win32 in Windows API e lo ha trasformato in un termine che si adatta perfettamente sia alle versioni API passate che a quelle future.

- **Win16** is the API for the first, [16-bit](#) versions of [Microsoft Windows](#). These were initially referred to as simply the *Windows API*, but were later renamed to Win16 in an effort to distinguish them from the newer, 32-bit version of the Windows API. The functions of Win16 API reside in mainly the core files of the OS: *kernel.exe* (or *knl/286.exe* or *knl/386.exe*), *user.exe* and *gdi.exe*. Despite the [file extension](#) of *exe*, these actually are [dynamic-link libraries](#).
- **Win32** is the [32-bit application programming interface](#) (API) for versions of Windows from 95 onwards. The API consists of functions implemented, as with Win16, in system DLLs. The core DLLs of Win32 are [kernel32.dll](#), [user32.dll](#), and [gdi32.dll](#). Win32 was introduced with [Windows NT](#). The version of Win32 shipped with [Windows 95](#) was initially referred to as Win32c, with *c* meaning *compatibility*. This term was later abandoned by Microsoft in favor of Win32.

Versioni Windows API

- **Win64** è la variante dell' API implementata su piattaforme a 64-bit della Windows architecture (x86-64). Versioni sia a 32-bit che a 64-bit di un'applicazione possono essere compilate da un'unico codebase, sebbene alcune API più vecchie sono state sconsigliate, e alcune dell'APIs che erano già state sconsigliate in Win32 sono state rimosse. Tutti i pointers di memoria sono a 64-bit di default (il modello LLP64), così il codice sorgente deve essere controllato per la compatibilità con l'aritmetica dei puntatori a 64-bit e riscritto dove necessario.

64-bit data models							Sample operating systems
Data model	short (integer)	int	long (integer)	long long	pointers, size_t		
LLP64	16	32	32	64	64		Microsoft Windows (x86-64 and IA-64) using Visual C++; and MinGW
LP64	16	32	64	64	64		Most Unix and Unix-like systems, e.g., Solaris, Linux, BSD, macOS. Windows when using Cygwin; z/OS

- **WinCE** è l'implementazione della Windows API per il sistema operativo Windows CE, utilizzato nei sistemi embedded.

Interazione del programma

SisOp
2020/21



- **La Windows API è stata progettata soprattutto per l'interazione tra il Sistema operativo e un'applicazione.**
- Per la comunicazione tra differenti applicazioni Windows, Microsoft ha sviluppato una serie di tecnologie a fianco della principale Windows API. Ciò è iniziato con [Dynamic Data Exchange](#) (DDE), che è stata rimpiazzata da [Object Linking and Embedding](#) (OLE) e più tardi dal [Component Object Model](#) (COM), [Automation Objects](#), controlli [ActiveX](#),
e soprattutto il [.NET Framework](#). ←
- Non vi è sempre una chiara distinzione tra queste tecnologie, e vi è molta sovrapposizione.
- La varietà di termini è in pratica il risultato di raggruppare meccanismi software che sono collegati a un dato aspetto dello sviluppo software.
.NET è una metodologia generale autosufficiente e una tecnologia per sviluppare applicazioni desktop e web scritte in una a varietà di linguaggi [just-in-time \(JIT\) compiled](#).

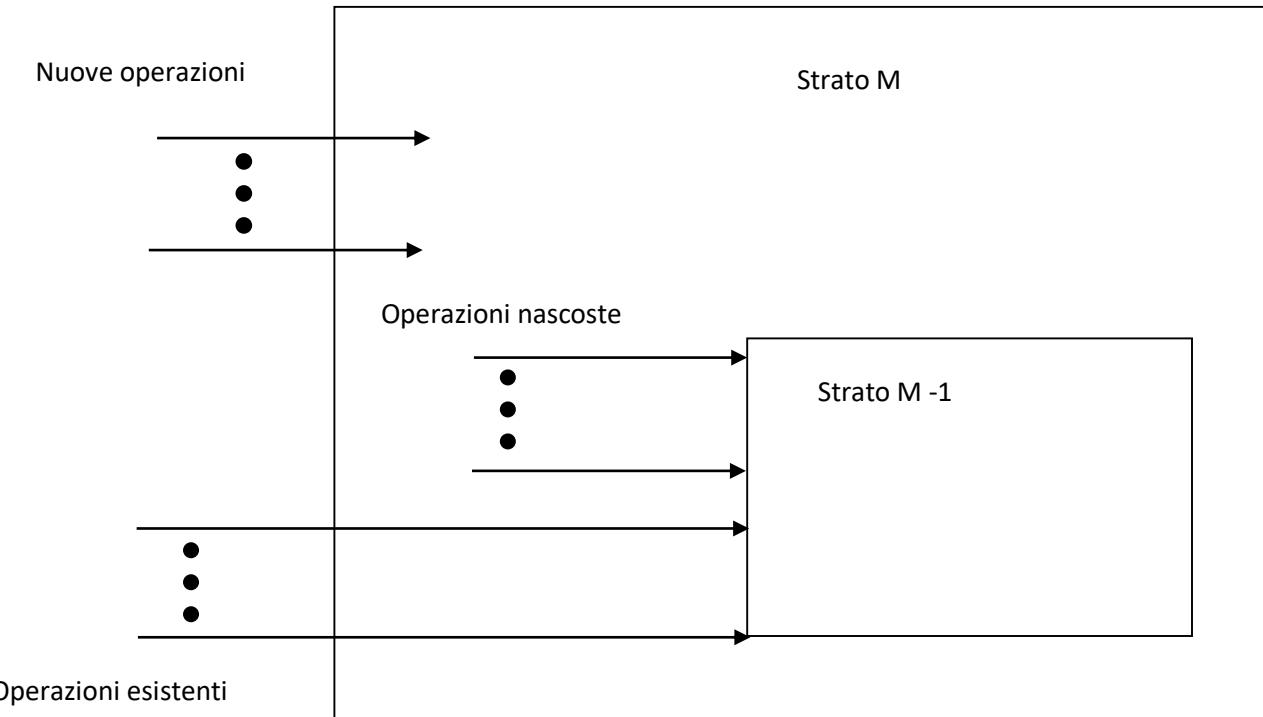


Programmi di sistema

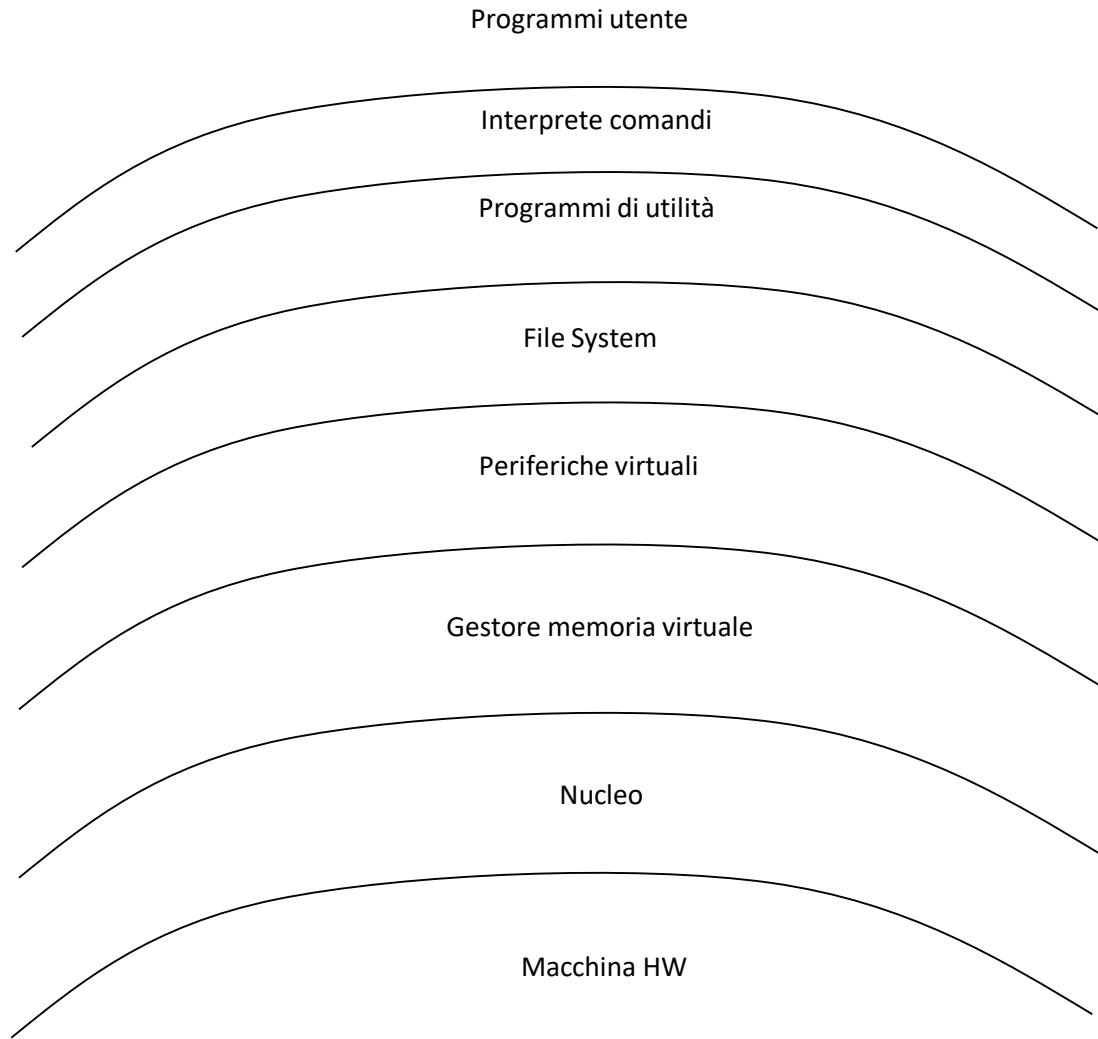
- Di varia natura, ad esempio in UNIX/Linux :
 - manipolazione dei file (editor, cp, mv, rm, mkdir, ...)
 - Informazioni di stato (date, time, who, df, ...)
 - sviluppo software ed esecuzione (traduttori, linker e loader, debugger)
 - comunicazione (ssh, ftp, mail)
 - applicativi (spreadsheet, latex, ...)
 - *interprete dei comandi*: esegue i comandi realizzati come programmi di sistema speciali
- 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 è indipendente dalla struttura interna del S.O.

Struttura del S.O.

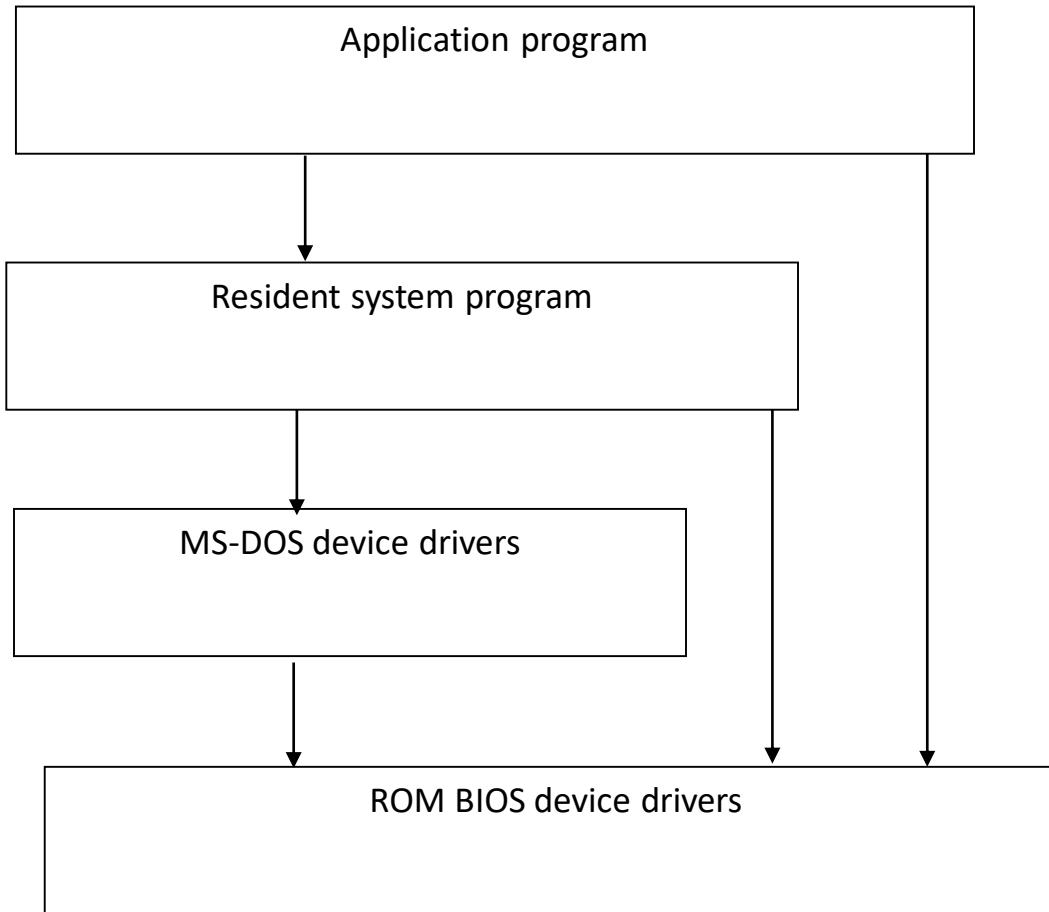
- Necessità di modularizzazione date le dimensioni e la complessità
- Sistema a *livelli*: *il livello più interno è l'hw, quello più esterno l'interfaccia di utente*
- Affinché il livello L_i possa richiedere i servizi al livello L_{i-1} deve conoscerne una *specifica precisa*, tuttavia *l'implementazione di tali servizi* deve risultare totalmente nascosta.



La struttura "a cipolla"



La struttura a livelli di MS-DOS

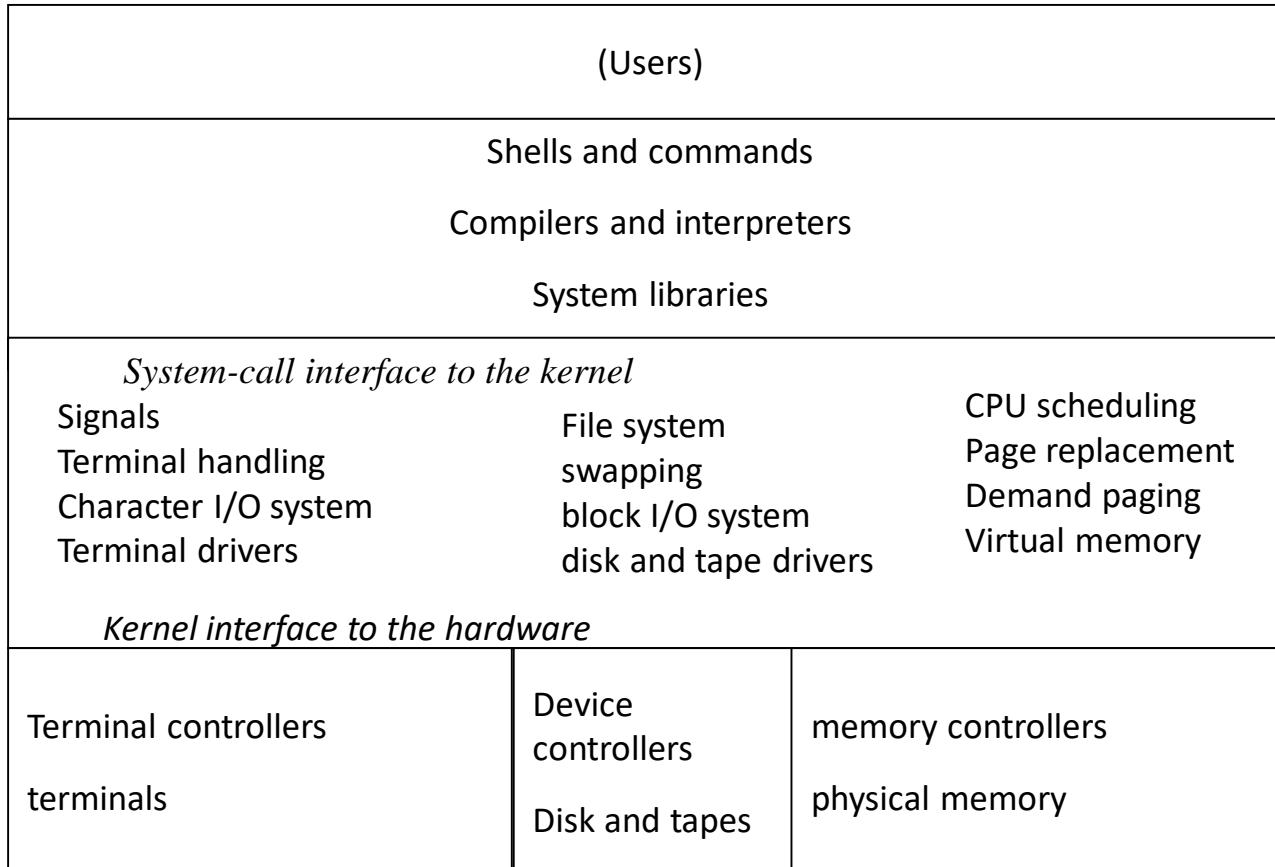


La struttura a "livelli" di UNIX

SisOp
2020/21



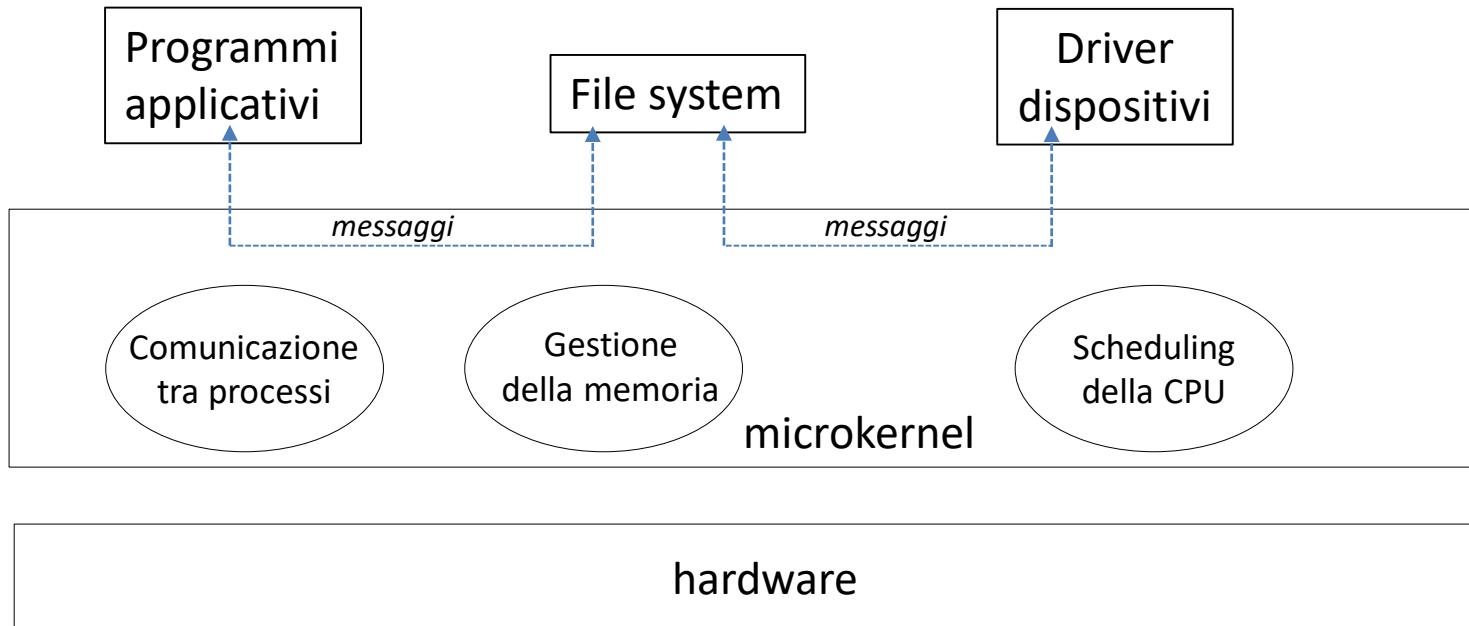
Struttura
monolitica



Con i moduli del kernel caricabili dinamicamente, supportati dai moderni SO incluso Linux, si può avere un kernel è costituito da un insieme di componenti di base, integrati da funzionalità aggiunte dinamicamente all'avvio o in esecuzione per mezzo di moduli.

Architettura a microkernel

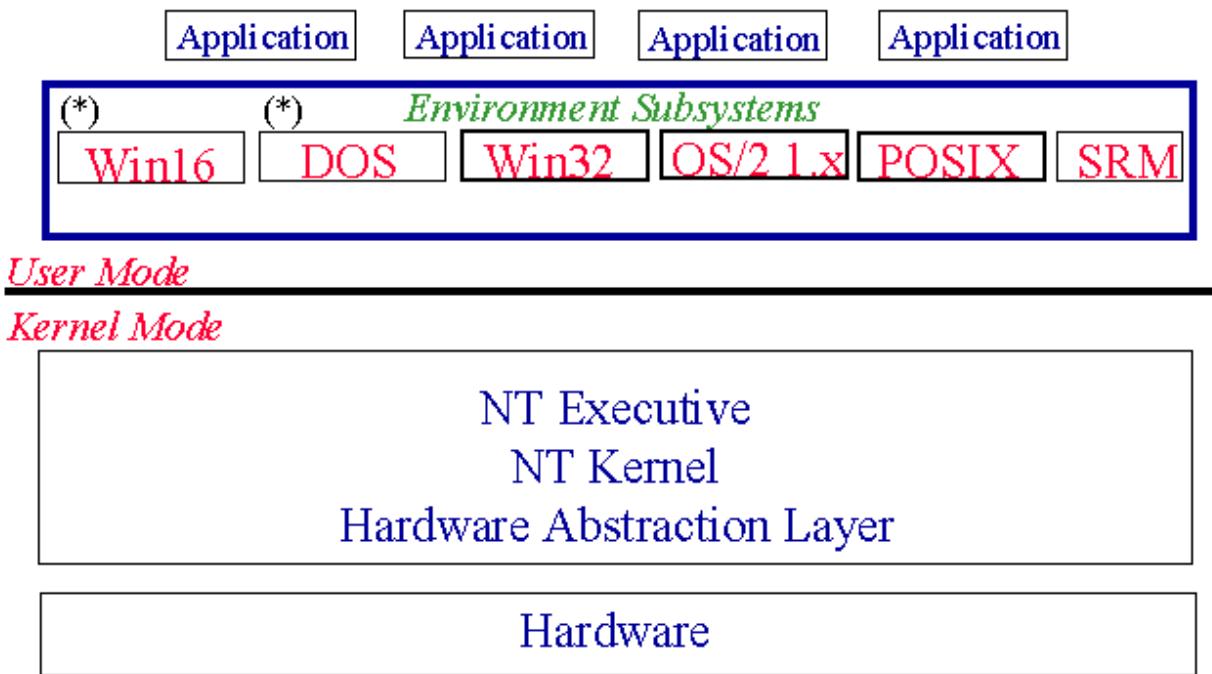
SisOp
2020/21



Esempi di microkernel: Darwin (componente kernel dei SO dei sistemi operativi macOS e iOS) contiene anche il microkernel Mach, QNX Neutrino alla base del SO real-time QNX per sistemi embedded

La struttura a livelli di Windows NT

Windows NT Architecture



(*) 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

SisOp
2020/21



- 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



UNIVERSITÀ DI PARMA

il mondo che ti aspetta

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA



La virtualizzazione

prof. Francesco Zanichelli

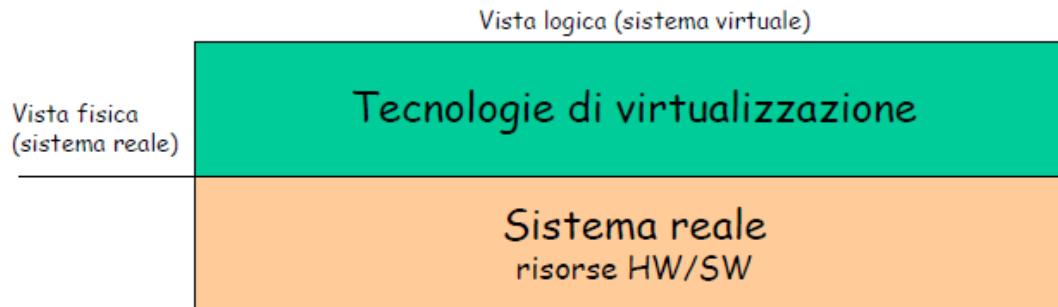


Macchine virtuali

- Macchine virtuali (VMWare, VirtualBox, Xen, Java?, .NET?) sono la logica evoluzione dell'approccio a livelli.
 - Virtualizzano sia hardware che kernel del SO
- Creano l'illusione di processi multipli, ciascuno in esecuzione sul suo processore privato e con la propria memoria virtuale privata, messa a disposizione dal proprio kernel del SO, che può essere diverso per processi diversi

Virtualizzazione

- Dato un sistema caratterizzato da un insieme di risorse (hardware e software), virtualizzare il sistema significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.
- Ciò si ottiene introducendo un livello di indirezione tra la vista logica e quella fisica delle risorse.



- **Obiettivo:** disaccoppiare il comportamento delle risorse hardware e software di un sistema di elaborazione, così come viste dall'utente, dalla loro realizzazione fisica.

Esempi di virtualizzazione

Astrazione: in generale un oggetto astratto (risorsa virtuale) è la rappresentazione semplificata di un oggetto (risorsa fisica):

- esibendo le proprietà significative per l'utilizzatore
- nascondendo i dettagli realizzativi non necessari.
- Esempio: tipi di dato vs. rappresentazione binaria nella cella di memoria
- Il disaccoppiamento è realizzato dalle operazioni (interfaccia) con le quali è possibile utilizzare l'oggetto.

Linguaggi di Programmazione. La capacità di portare lo stesso programma (scritto in un linguaggio di alto livello) su architetture diverse è possibile grazie alla definizione di una macchina virtuale in grado di interpretare ed eseguire ogni istruzione del linguaggio, indipendentemente dall'architettura del sistema (S.O. e HW):

- Interpreti (esempio Java Virtual Machine)
- Compilatori

Virtualizzazione a livello di processo. I sistemi multitasking permettono la contemporanea esecuzione di più processi, ognuno dei quali dispone di una macchina virtuale (CPU, memoria, dispositivi) dedicata. La virtualizzazione è realizzata dal kernel del sistema operativo.

Sistemi Operativi per la Virtualizzazione

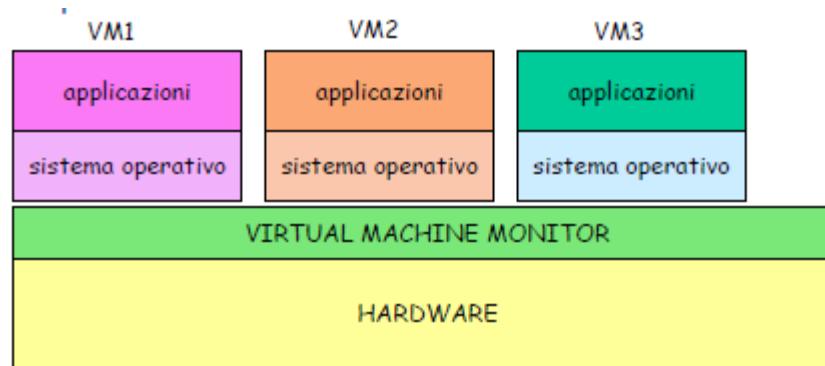
SisOp
2020/21



- La macchina fisica viene trasformata in n interfacce (macchine virtuali), ognuna delle quali è una replica della macchina fisica:
 - dotata di tutte le istruzioni del processore (sia privilegiate che non privilegiate)
 - dotata delle risorse del sistema (memoria, dispositivi di I/O).
- Su ogni macchina virtuale è possibile installare ed eseguire un sistema operativo (eventualmente diverso da macchina a macchina): **Virtual Machine Monitor**

Virtualizzazione di Sistema

- Una singola piattaforma hardware viene condivisa da più sistemi operativi, ognuno dei quali è installato su una diversa macchina virtuale.



- Il disaccoppiamento è realizzato da un componente chiamato Virtual Machine Monitor (**VMM**, o **hypervisor**) il cui compito è consentire la condivisione da parte di più macchine virtuali di una singola piattaforma hardware. Ogni macchina virtuale è costituita oltre che dall'applicazione che in essa viene eseguita, anche dal sistema operativo utilizzato.
- Il VMM è il mediatore unico nelle interazioni tra le macchine virtuali e l'hardware sottostante, che garantisce:
 - **isolamento** tra le VM
 - **stabilità** del sistema

VMM di sistema vs. VMM ospitato

SisOp
2020/21

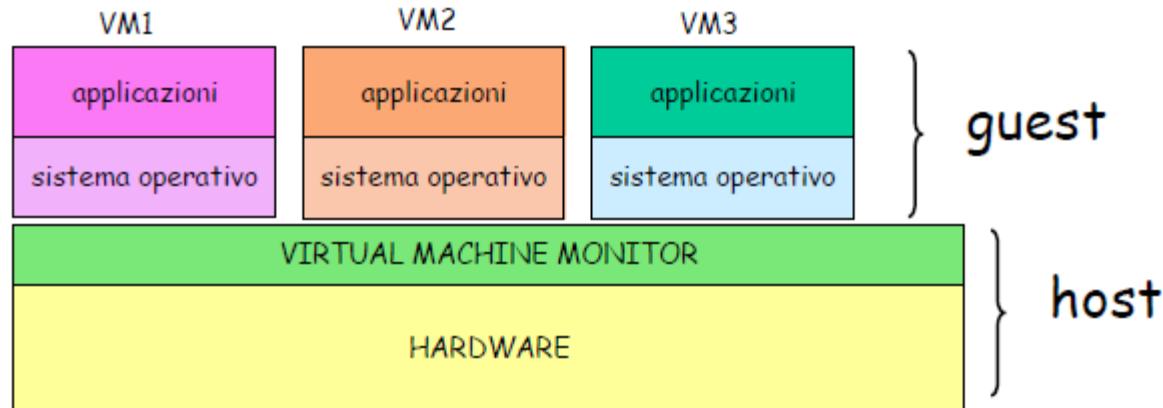


VMM di Sistema

- le funzionalità di virtualizzazione vengono integrate in un sistema operativo leggero, costituendo un unico sistema posto direttamente sopra l'hardware dell'elaboratore.
- E' necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche.
- Esempi di VMM di sistema: VMware ESX, Xen, VirtualIron.

VMM di Sistema

- **Host:** piattaforma di base sulla quale si realizzano macchine virtuali. Comprende la macchina fisica, l'eventuale sistema operativo ed il VMM.
- **Guest:** la macchina virtuale. Comprende applicazioni e sistema operativo



VMM di Sistema

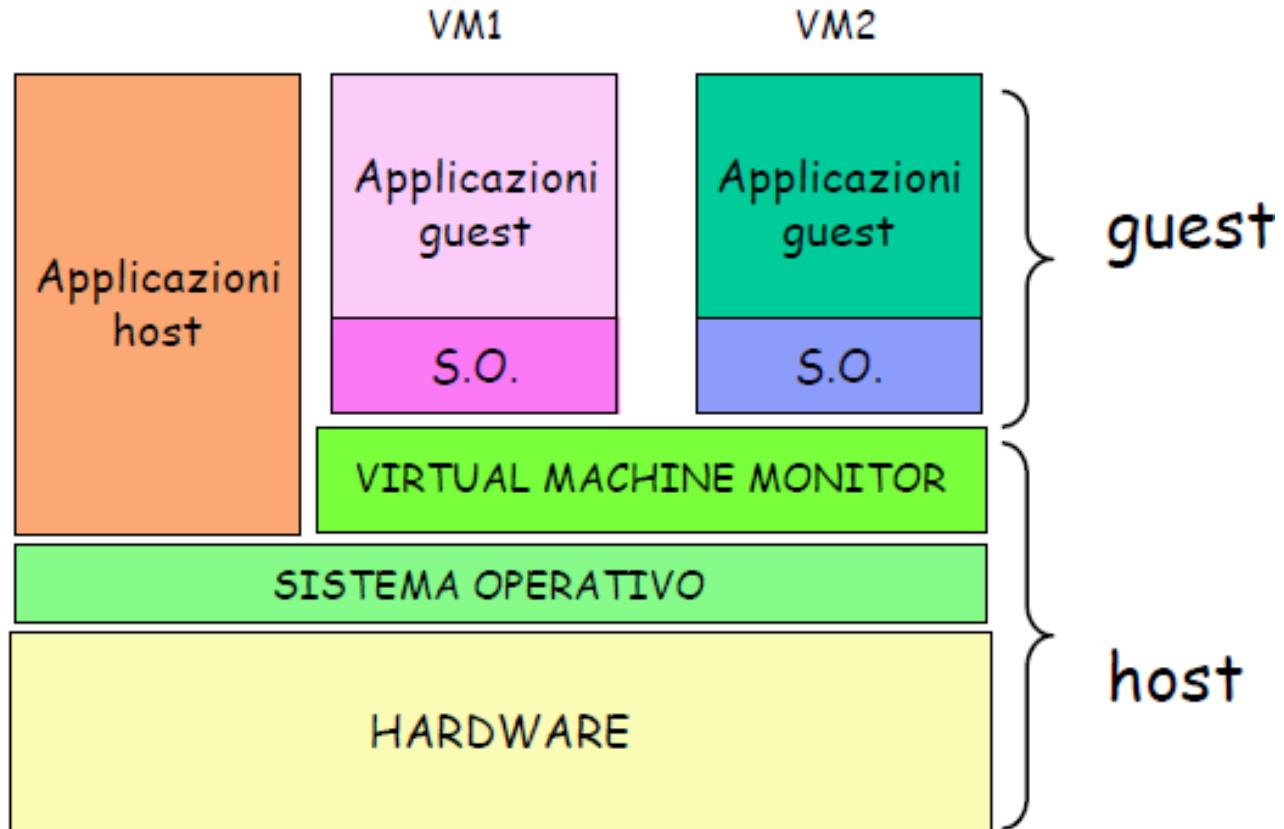


VMM ospitato

Il VMM viene installato come un'applicazione sopra un sistema operativo esistente, che opera nello spazio utente e accede all'hardware tramite le system call del S.O. su cui viene installato.

- Più semplice l'installazione (come un'applicazione).
- Può fare riferimento al S.O. sottostante per la gestione delle periferiche e può utilizzare altri servizi del S.O.(es. scheduling, gestione delle risorse.).
- Peggiore la performance.
- Prodotti: User Mode Linux, VMware Server/Player, Microsoft Virtual Server, Parallels

VMM ospitato



Emulazione vs Virtualizzazione

SisOp
2020/21



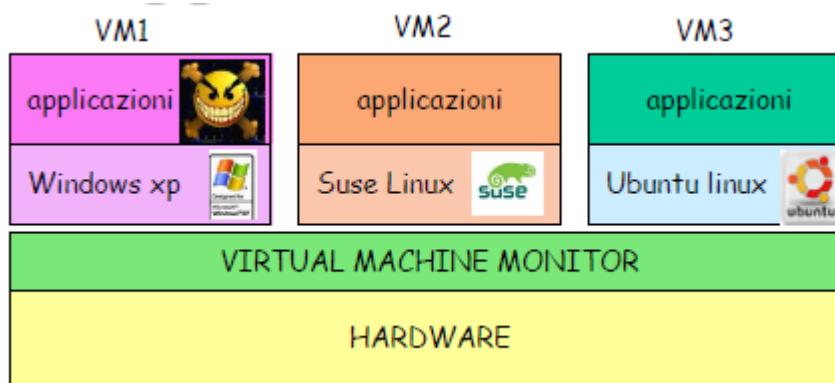
Emulazione

- eseguire applicazioni (o SO) compilate per un'architettura su di un'altra.
- uno strato software emula le funzionalità dell'architettura; il s.o. esegue sopra tale strato (a livello user). Le istruzioni macchina privilegiate e non privilegiate vengono emulate via SW (Bochs, Qemu).

Virtualizzazione

- definizione di contesti di esecuzione multipli (macchine virtuali) su di un singolo processore, partizionando le risorse.

Vantaggi della virtualizzazione



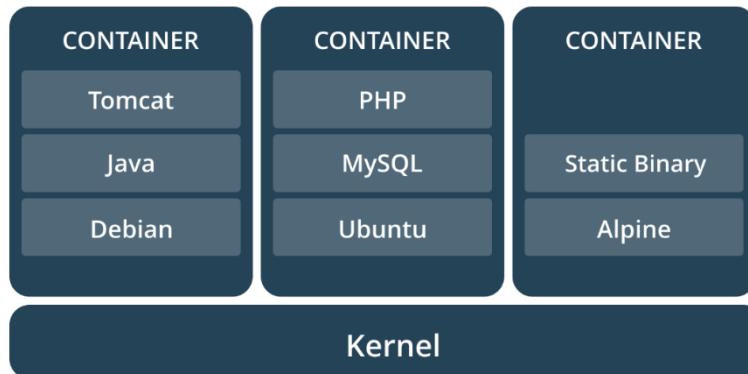
- **Uso di più S.O. sulla stessa macchina fisica:** più ambienti di esecuzione (eterogenei) per lo stesso utente:
 - Legacy systems
 - Possibilità di esecuzione di applicazioni concepite per un particolare S.O.
- **Isolamento degli ambienti di esecuzione:** ogni macchina virtuale definisce un ambiente di esecuzione separato (sandbox) da quelli delle altre:
 - possibilità di effettuare testing di applicazioni preservando l'integrità degli altri ambienti e del VMM.
 - Sicurezza: eventuali attacchi da parte di malware o spyware sono confinati alla singola macchina virtuale

Vantaggi della virtualizzazione

- **Consolidamento HW:** possibilità di concentrare più macchine (ad es. server) su un'unica architettura HW per un utilizzo efficiente dell'hardware (es. server farm):
 - Abbattimento costi HW
 - Abbattimento costi amministrazione
- **Gestione facilitata delle macchine:** è possibile effettuare in modo semplice:
 - la creazione di macchine virtuali (virtual appliances)
 - l'amministrazione di macchine virtuali (reboot, ricompilazione kernel, etc.)
 - migrazione a caldo di macchine virtuali tra macchine fisiche:
 - possibilità di manutenzione hw senza interrompere i servizi forniti dalle macchine virtuali
 - disaster recovery
 - workload balancing: alcuni prodotti prevedono anche meccanismi di migrazione automatica per far fronte in modo “autonomico” a situazioni di sbilanciamento

Virtualizzazione a livello del SO

- Una modalità di virtualizzazione in cui il kernel consente l'esistenza di molteplici istanze isolate nello spazio utente
- Tali istanze, ad es. i container come Docker, rendono disponibili ai programmi contenuti solo i contenuti (librerie, porzioni di file system) e dispositivi che sono state assegnati all'istanza.



- Consentono di migliorare la sicurezza, l'indipendenza dall'hw e la gestione delle risorse
- Rispetto alla virtualizzazione piena (basata su VMM), l'overhead è inferiore ma è anche inferiore la flessibilità, dato che non si possono utilizzare SO guest, o anche solo il kernel, differenti da quello dell'host.