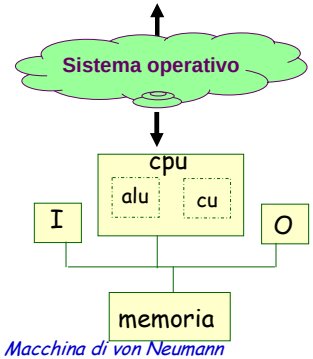
Lezione 2:

Come interagiscono hardware e sistemi operativi

- supporto hardware ai s.o.
- hardware che deve essere gestito dal s.o.
- come il s.o. operativo protegge se stesso e i programmi

Compiti del S.O.



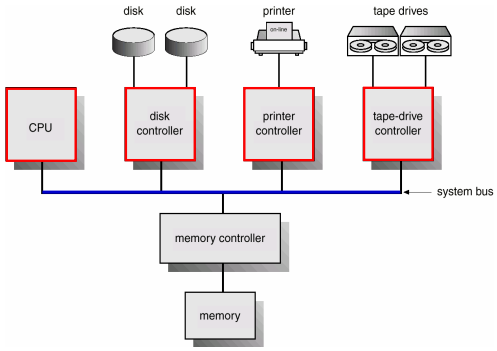
Macchina di von Neumann

Un S.O. ha il compito di rendere semplice (all'utente), l'utilizzo del calcolatore

Cosa deve fare un S.O. ?

Interazioni con:
- CPU
- Memoria
- Dispositivi di I/O

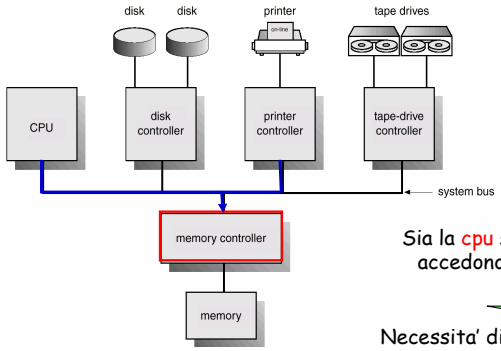
Architettura di un Sistema di Calcolo



Controller
=
dispositivo hw per la gestione delle periferiche

- la cpu e i controller operano in modo concorrente
- la cpu e i controller comunicano attraverso un bus

Architettura di un Sistema di Calcolo



Sia la cpu sia i controller accedono alla memoria

Necessita' di sincronizzazione degli accessi

Controller della memoria

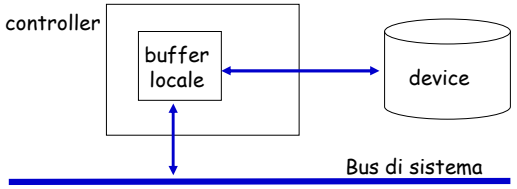
la CPU

Compiti della CPU:
- recuperare una istruzione dalla memoria
- Decodificarla per determinare tipo e operandi
- eseguirla

Tutte le CPU hanno un insieme di registri:
- per contenere dati in transito da e per la memoria
- program counter (indirizzo della prossima istruzione)
- stack pointer (indirizzo dello stack)
- program status word (insieme di bit di controllo)
- per delimitare lo spazio di indirizzamento del programma in esecuzione

I Dispositivi di I/O

- Ogni *controller* ha un suo buffer locale.
- Le operazioni di I/O avvengono dal dispositivo verso i buffer locali e viceversa.
- La CPU sposta i dati dalla memoria ai buffer e viceversa.



problema

Come fa la cpu a sapere che i dati sono stati trasferiti nel buffer locale del controller?

I soluzione:
La CPU interroga in continuazione il controller

Tale tecnica, chiamata attesa attiva (o polling), tiene impegnata la CPU fino al completamento dell'operazione di I/O

II soluzione:
Il controller avvisa la CPU della fine dell'operazione di I/O mediante un evento

Gli eventi

Gli eventi (oppure segnali, oppure interruzioni) sono il principale meccanismo con cui si sincronizzano le azioni di un moderno sistema operativo

La ricezione di un evento avvisa la CPU che deve fare qualcosa

Se non ci sono processi, dispositivi da servire o utenti con cui interagire il sistema operativo rimane inattivo nell'attesa di un evento (i S.O. sono *events driven*)

eventi: dipendono dall'architettura

In generale due tipi di eventi

esempio: processori Intel

evento segnalato da una componente hardware (disp. di I/O, memoria, timer)

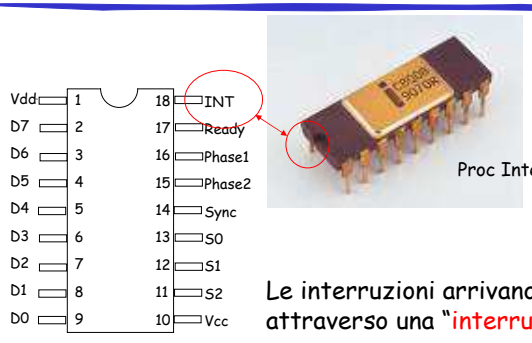
segnali di interruzione (interrupt)

evento segnalato da un programma in esecuzione

Segnali di eccezione
(trap → es: div per 0
fault → es: overflow
abort → es: errore in un evento)

In generale si parla sempre di interruzioni

La interrupt line



Proc Intel 8008 (1974)

Le interruzioni arrivano alla CPU attraverso una "interrupt line"

Alla fine di ogni ciclo macchina la CPU controlla la presenza di un segnale sulla interrupt line

Come viene gestito una interruzione

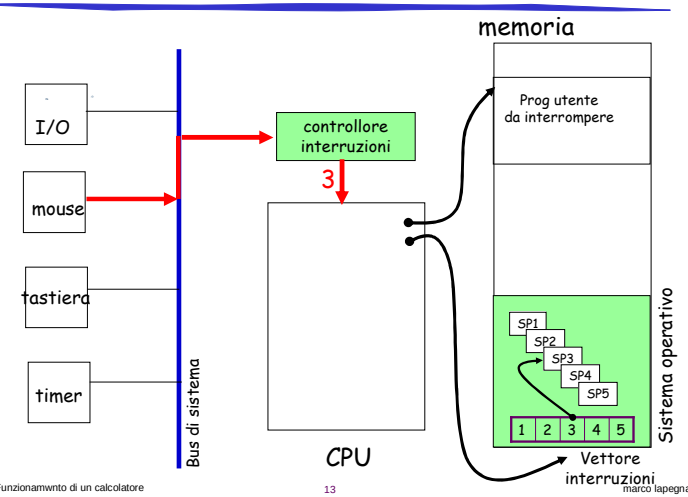
- Un dispositivo attiva una comunicazione elettrica con un controllore delle interruzioni della CPU (separato da essa)
- Il controllore delle interruzioni informa la CPU codificando il tipo di interruzione
- La CPU interrompe l'esecuzione dell'istruzione corrente e salva lo stato del programma in esecuzione (dati e registri della CPU).
- trasferisce il controllo ad una *interrupt service routine* (ISR) il cui codice si trova in una prefissata zona della memoria
- Le ISR (Interrupt Service Routine) eseguono il codice specifico per gestire il segnale appena giunto (ad es. trasferisce il dato dal buffer locale del controller alla memoria)
- Al termine dell'esecuzione della ISR, il S.O. ripristina lo stato del programma e riprende l'esecuzione

La gestione delle interruzioni

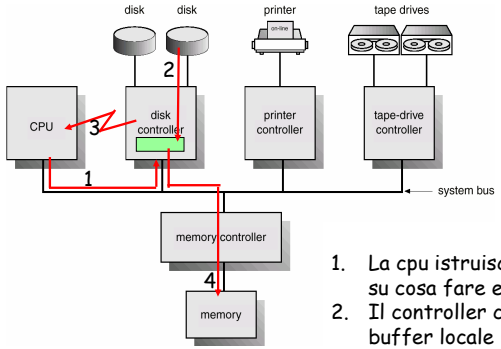
L'efficienza di un Sistema Operativo dipende dall'efficienca con cui sono gestite le interruzioni

Per rendere efficiente la gestione delle interruzioni si usa di solito una sola ISR che gestisce una tabella di puntatori contenente gli indirizzi delle varie procedure di servizio (vettore delle interruzioni)

Gestione delle interruzioni

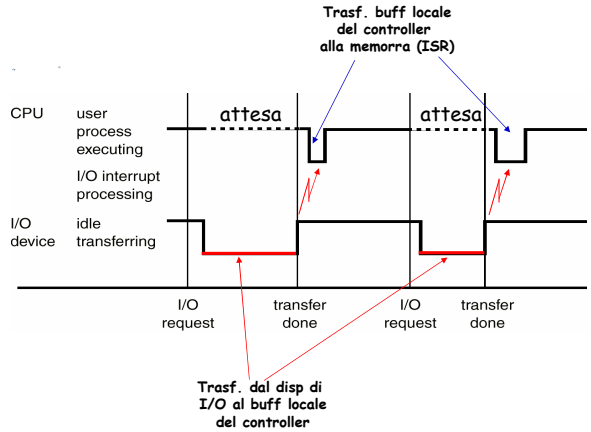


Esempio: lettura di un dato (I/O sincrono)



1. La cpu istruisce il controller su cosa fare e attende
2. Il controller carica i dati nel buffer locale
3. Il controller manda un interrupt di fine I/O alla cpu
4. La cpu trasferisce il dato dal buffer locale alla memoria

Time Line per l'I/O di un singolo processo



Problema

Se la CPU effettua una operazione di I/O, deve attendere il completamento del trasferimento dei dati dal dispositivo al alla CPU (I/O sincrono)

Ma i dispositivi sono molto piu' lenti della CPU

Uso inefficiente della CPU

I/O sincrono

- vantaggi:
  - una sola richiesta di I/O pendente alla volta
  - la CPU riconosce subito da quale dispositivo arriva il segnale di interruzione

MA

- svantaggi:
  - la CPU rimane inattiva per tutta la durata dell'I/O
  - no a operazioni I/O in parallelo
  - no a sovrapposizione di I/O e calcolo

I/O asincrono

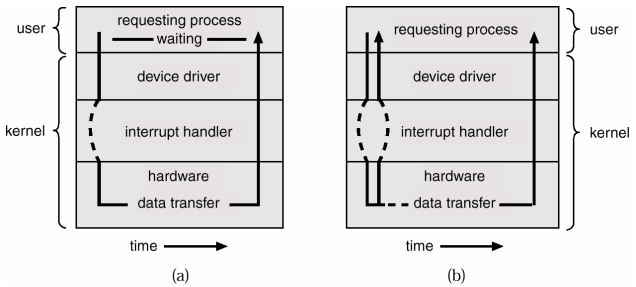
Invece di rimanere inattiva, la CPU potrebbe essere impiegata a gestire altri programmi (multiprogrammazione)

Dopo l'avvio dell'I/O il controllo deve tornare subito alla CPU (I/O asincrono)

se c'e' un solo processo in esecuzione, tale processo deve comunque attendere la fine dell'operazione di I/O

Possibilita' di numerose richieste di I/O pendenti

I/O sincrono vs asincrono

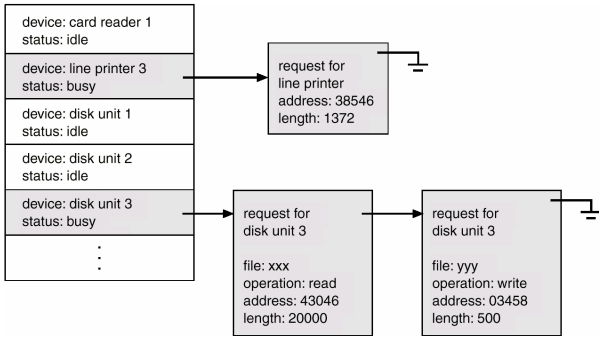


Dopo l'inizio dell'I/O, il controllo ritorna al programma solo dopo la fine dell'operazione dell'I/O

Dopo l'inizio dell'I/O, il controllo ritorna al programma immediatamente

Device-Status Table

La presenza di piu' operazioni di I/O pendenti impone la presenza di una tabella di stato dei dispositivi



problema

Alcuni dispositivi di I/O veloci
(ad es. un lettore di nastri o dischi)
sono capaci di tasferire dati
a una velocita' simile a quelle della memoria

La CPU e' costretta a interrompere di continuo il suo lavoro per gestire le continue interruzioni in presenza di numerosi dati di I/O

Importanza del controllore della memoria

DMA (Direct Memory Access)

- E' un dispositivo dedicato in grado di accedere autonomamente alla memoria
- La CPU avvia la procedura di I/O avvisando il DMA.
- Il DMA accede direttamente alla memoria per effettuare le operazioni di I/O, trasferendo blocchi di dati (da 128 a 4096 byte)
- La CPU è libera di procedere autonomamente
- viene generata una sola interruzione per blocco di dati

La memoria centrale

La memoria centrale e' il supporto su cui sono conservati dati e istruzioni

E' vista dalla CPU come una sequenza lineare di locazioni con un indirizzo

Le uniche operazioni permesse sui dati residenti in memoria sono load e store

E' il solo dispositivo di memorizzazione di grandi dimensioni direttamente accessibile dalla CPU

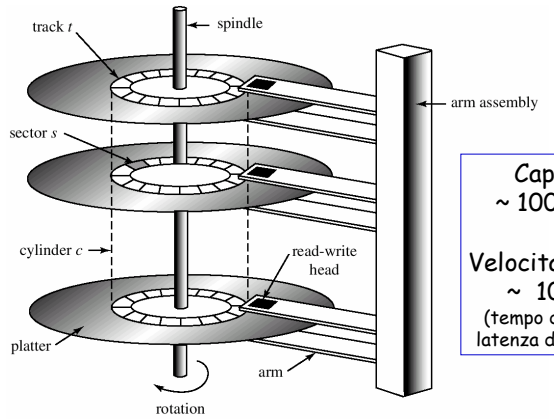
Memorie di massa

La memoria centrale e' un dispositivo volatile

Ha una capicita' dell'ordine dei 10^9 byte (Gbyte) e non e' sufficiente a contenere i modo permanente tutti i dati e i programmi di un sistema operativo

necessita' di dispositivi piu' capienti e non volatili
(memoria di massa / dischi magnetici)

Dischi magnetici



Capacita'
~ 100 Gbyte

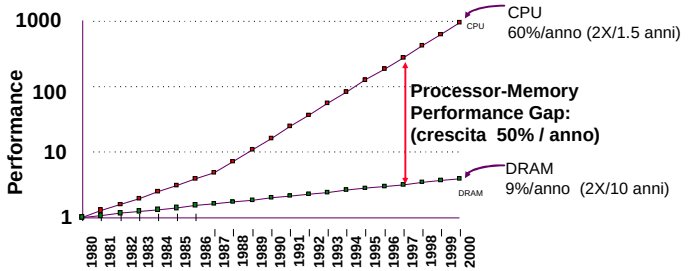
Velocita' accesso
~ 10^{-6}sec.
(tempo di ricerca + latenza di rotazione)

Gap tra memoria e CPU

Tempo per una operazione aritmetica
1 ciclo di clock (~ 10^{-9} sec)

Tempo di accesso alla memoria
4-5 cicli di clock (~ 10^{-8} sec)

Rallentamento della CPU



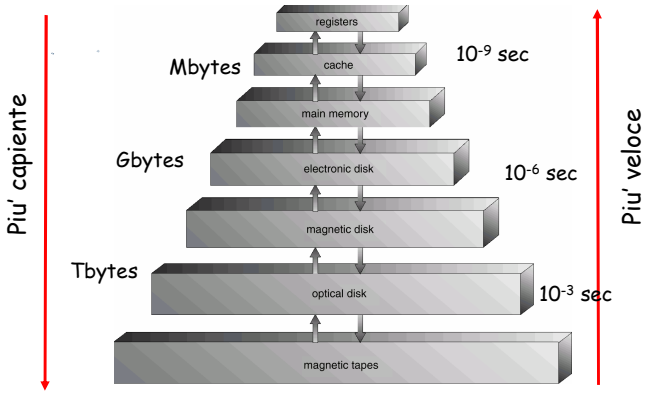
Registri e cache

Soluzione:
uso di dispositivi di memorizzazione 'on chip' capaci di supportare la velocita' operativa della CPU (cache e registri)

Al momento del loro uso, blocchi di dati sono copiati dalla memoria nella cache

Caratteristiche:
- limitata capacita' (~1 Mbyte)
- alta velocita' di accesso (~ 10^{-9} sec)

La gerarchia delle Memorie



Caching

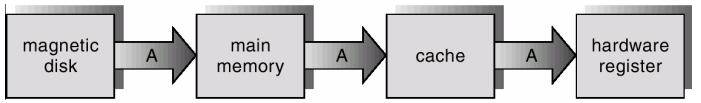
Caching – utilizzo di una memoria a piu' alta velocità per mantenere informazioni cui si accede più spesso.

- Richiede un politica di gestione della cache.
- Provoca una replicazione dei dati, quindi necessita di una politica di gestione che garantisca la *consistenza* dei dati (di solito gestita dall'hardware).

E' un concetto che puo' essere applicato a piu' livelli

- La memoria centrale puo' essere vista come una *cache* per i dischi magnetici
- I dischi possono essere visti come una cache per i supporti di backup

Migrazione di un dato dal disco ai registri



Problemi :

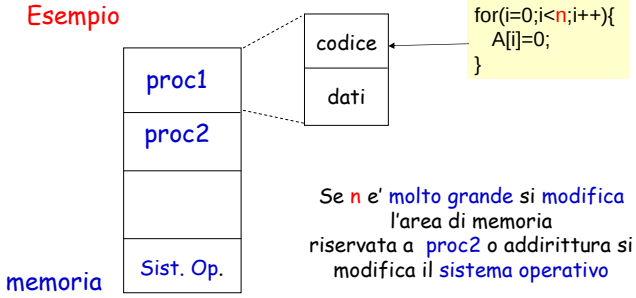
- dimensioni della cache (piu' grande = piu' costoso)
- criteri di aggiornamento ('prevedere' quali dati saranno utilizzati dalla cpu)

Gestione efficiente della cache ➔ 80% dei dati deve trovarsi nella cache quando servono

problema

La condivisione delle risorse (memoria, cpu, I/O) da parte dei processi di un sistema operativo, comporta che un errore in un programma puo' alterare il funzionamento di tutto il sistema

Esempio




```
for(i=0;i<n;i++){
   A[i]=0;
}
```

Se n e' molto grande si modifica l'area di memoria riservata a proc2 o addirittura si modifica il sistema operativo

Protezione delle risorse

L'esistenza di risorse condivise richiede che il sistema operativo garantisca che un programma scorretto non possa effettuare operazioni non consentite.

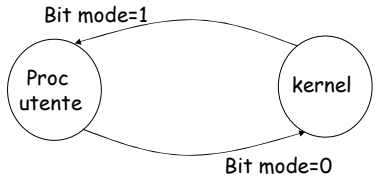
Le istruzioni possono essere eseguite in due modalità:

1. *User mode* – (un utente qualsiasi puo' eseguire un insieme ristretto di istruzioni).
2. kernel *mode* (anche *monitor mode*, *system mode* o *superuser mode*) – (il sistema operativo puo' eseguire tutte le istruzioni).

(dual mode operation)

Supporto hardware per la dual mode operation

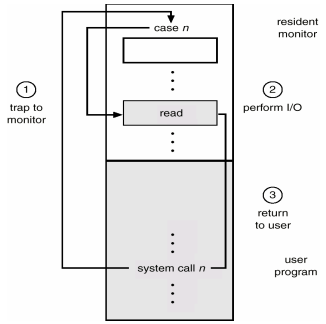
- La CPU deve essere dotata di un *Mode bit* che indica lo stato corrente: system (0) or user (1).
- Quando giunge una interruzione o avviene un errore il sistema passa in modalità "sistema" e viene attivata la procedura di servizio.



Esistono *istruzioni privilegiate* che possono essere eseguite solo in modalità superutente.

Protezione dello I/O

- Ad es. tutte le istruzioni di I/O sono di tipo privilegiato.
- Per effettuare delle operazioni di I/O vengono utilizzate delle chiamate di sistema (*System call*).
- Il Sistema operativo deve garantire che un utente non possa mai avere pieno controllo del sistema in modalità superutente.



Le system call

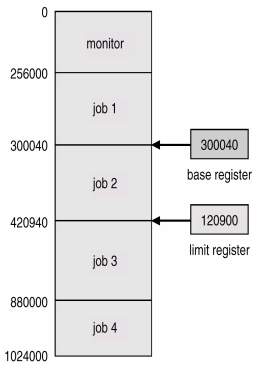
Quando un programma effettua una system call, il S.O.

1. manda una interruzione alla CPU
2. pone il bit mode =0 (modalita' sistema)
3. esegue la procedura di servizio
4. ripone il bit mode=1 (modalita' utente)
5. restituisce il controllo del sistema al programma interrotto

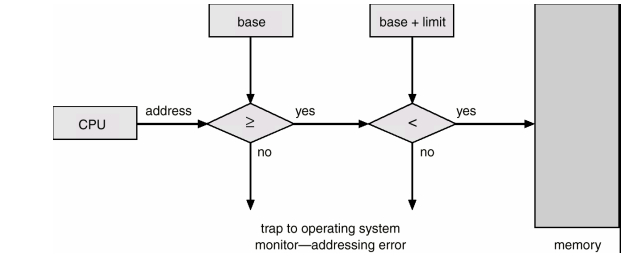
Una chiamata di sistema viene gestita come una interruzione

Supporto hardware per la protezione della memoria

- E' necessario garantire protezione della memoria , come minimo per le informazioni che sitrovano nel vettore delle interruzioni e nelle ISR.
- La CPU deve possedere due registri per stabilire le locazioni di memoria cui ogni programma ha diritto ad accedere.:
  - **Base register** – memorizza il più piccolo indirizzo di memoria cui l'accesso è lecito.
  - **Limit register** – Contiene la dimensione massima di memoria ad accesso consentito
- La memoria al di là dell'intervallo indicato è *protetta*.



Protezione della memoria



- Quando una istruzione viene eseguita in modalià sistema ha accesso completo a TUTTA la memoria.
- Le istruzioni per caricare i valori dei registri *base* e *limit* sono di tipo privilegiato.

Supporto hardware per la protezione della CPU

- *Timer* – genera una interruzione dopo un intervallo di tempo specificato, per garantire che il sistema operativo mantenga il controllo del sistema.
  - Il Timer viene decrementato ad ogni colpo di clock.
  - Quando il timer raggiunge il valore zero, viene generata una interruzione e il controllo passa al s.o..
- Viene in genere utilizzato per realizzare sistemi di tipo *time sharing*.
- E' utilizzato anche per calcolare l'ora attuale.
- Il Caricamento del timer è una istruzione privilegiata.

In conclusione

Il Sistema Operativo deve:

- Interagire con i controller e il DMA
- gestire le interruzioni
- gestire la coerenza dei valori nelle memorie
- gestire il passaggio del controllo della CPU da un programma ad un altro
- gestire le modalita' di esecuzione
- proteggere la memoria
- proteggere la cpu