Università di Bologna
Corso di Studi: Laurea in Ingegneria Informatica
A.A. 2005-2006

Sistemi Operativi L-A

Prof. Paolo Bellavista

`http://www.lia.deis.unibo.it/Courses/sola0506-info/`
`http://www.lia.deis.unibo.it/Staff/PaoloBellavista/`

Obiettivi del Corso

- Fornire alcuni concetti fondamentali della ***teoria dei sistemi operativi***
- Illustrare le caratteristiche di ***sistemi operativi reali (UNIX/Linux e MSWin2000)*** e gli strumenti a disposizione di utenti e programmatori per il loro utilizzo
- Sperimentare in ***laboratorio*** i concetti e gli strumenti visti in aula

Capacità richieste in ingresso:

- conoscenza dei ***linguaggi C e Java***
- uso dei linguaggi C e Java nello sviluppo di applicazioni e programmazione di sistema
- fondamenti di architettura degli elaboratori

Capacità ottenute in uscita:

- conoscenza dei concetti alla base dei sistemi operativi moderni
- capacità di sviluppare programmi di sistema e applicazioni nell'ambiente UNIX/Linux (e in parte MSWin2000)

Argomenti trattati

- Che cos'è un sistema operativo: ruolo, funzionalità e struttura
- ***Organizzazione*** e ***struttura*** di un sistema operativo
- ***Processi e thread***
- ***Scheduling*** della CPU
- Interazione tra processi mediante ***memoria condivisa*** e ***scambio di messaggi***
- Cenni di ***sincronizzazione*** dei processi
- Gestione della ***memoria***
- Gestione del ***file system***
- Gestione dei dispositivi di ***Input/Output***
- ***Protezione***

Panoramica sul Corso

Introduzione:

- Che cos'è un sistema operativo: ruolo, funzionalità e struttura
- Evoluzione dei sistemi operativi: ***batch***, ***multiprogrammazione***, ***time-sharing***
- Richiami sul funzionamento di un elaboratore: ***interruzioni e loro gestione, I/O, modi di funzionamento single e dual, system call***

Panoramica sul Corso

Organizzazione di un sistema operativo:

- Funzionalità
- Classificazione in base a struttura: sistemi ***monolitici*** e ***modulari***, sistemi ***stratificati***, ***macchina virtuale***
- Cenni introduttivi di organizzazione e funzionalità del sistema operativo ***UNIX/Linux***
- Cenni introduttivi di organizzazione e funzionalità del sistema operativo ***MSWindows2000***

Panoramica sul Corso

Processi e thread:

- ***Concetto di processo pesante/leggero*** e sua ***rappresentazione*** nel sistema operativo
- ***Stati di un processo***
- ***Gestione dei processi pesanti/leggeri*** da parte del SO
- Operazioni sui processi
- ***Classificazione*** dei processi
- La gestione dei processi in UNIX/Linux: stati, rappresentazione, gestione (scheduling), operazioni e comandi relativi ai processi

Panoramica sul Corso

Scheduling della CPU:

- Concetti generali: ***code, preemption, dispatcher***
- ***Criteri*** di scheduling
- ***Algoritmi di scheduling***: FCFS, SJF, con priorità, round-robin, con code multiple, …
- Scheduling in UNIX/Linux e MSWin2000

Panoramica sul Corso

Interazione tra processi:

- **Mediante memoria condivisa**

  Il problema della ***sincronizzazione tra processi***
  Sezione critica e mutua esclusione, i semafori, strumenti hardware per la sincronizzazione: test-and-set
- **Mediante scambio di messaggi**
  - ***Comunicazione***
    diretta/indiretta,simmetrica/asimmetrica, buffering
  - ***Interazione tra processi UNIX***: comunicazione mediante pipe e fifo, sincronizzazione tramite segnali

Panoramica sul Corso

Gestione della memoria:

- Spazi degli indirizzi e binding
- Allocazione della memoria
  - ❖ ***Contigua***: a partizione singola e partizioni multiple; frammentazione;
  - ❖ ***Non contigua***: paginazione, segmentazione
- ***Memoria virtuale***
- Gestione della memoria in UNIX

Panoramica sul Corso

Gestione del file system e dei dispositivi di I/O:

- file system e sua realizzazione
- il file system di UNIX: organizzazione logica e fisica, comandi e system call per la gestione e l'accesso a file/direttori
- Driver di dispositivi

Panoramica sul Corso

Cenni di problematiche e soluzioni per la *protezione* :

- Scopi e principi di protezione
- ***Domini di protezione***
- Matrice di accesso
- Controllo degli accessi
- Sistemi basati su ***capabilities***

Percorso didattico

- **Argomenti teorici**
- **Esemplificazioni**: sui sistemi operativi UNIX/Linux e MSWin2000, sia tramite ***programmazione di sistema in linguaggio C*** che tramite sviluppo di ***file comandi in shell*** e di ***applicazioni concorrenti in Java***
- **Esercitazioni**:
  - ⇨ **Attività in laboratorio**
- Progetto opzionale di approfondimento finale?

Attività in laboratorio

- Esattamente come le lezioni in aula, è ***parte integrante dell'attività didattica!***
- Ogni settimana verrà svolta in Lab3 una esercitazione con una ***prima parte di lezione di programmazione*** ed una ***seconda parte di risoluzione di esercizi proposti***
- Le esercitazioni cominciano ***giovedì 11 maggio***
- L'attività sarà assistita da due **tutor**:
  - **Ing. Eugenio Magistretti**
  - **Ing. Marco Montali**

Accesso al Laboratorio

- L'attività si svolgerà in sala terminali (Lab3) su sistemi dual-boot RedHatLinux/MSWinXP
- Necessità di organizzare due turni (alternati fra AK e LZ di settimana in settimana):
  - **giovedì ore 09:00-11:00**
  - **venerdì ore 09:00-11:00**
- Per partecipare alle esercitazioni è necessario registrarsi al più presto:

  http://lia.deis.unibo.it/Courses/sola0506-info/

- **Account: sono già attivi per tutti gli studenti**
  - **Username determinato in base a matricola e cognome**
  - **Password: è il pin associato al vostro badge**

Esame

- Una ***prova scritta*** obbligatoria (in parte teorica e in parte di sviluppo di codice in Lab3)**:**

  martedì 4 luglio 2006, ore 09:00

  martedì 25 luglio 2006, ore 09:00

- Una ***prova orale*** facoltativa (non per tutti ☹) dopo il superamento dello scritto (sostituibile con ***progetto di approfondimento***, anche a piccoli gruppi di 2/3 allievi?)
- Valutazione della partecipazione in aula e della ***consegna di alcune esercitazioni?***

Materiale Didattico

- **Copia** delle diapositive mostrate a lezione (scaricabili mano a mano dalla pagina Web del corso)
- **Libro adottato:**
  - A. Silbershatz, P.B. Galvin, G. Gagne: ***Sistemi Operativi – Concetti ed Esempi*** (VII edizione), Pearson, 2006

  oppure
  - P. Ancilotti, M. Boari, A. Ciampolini, G. Lipari: ***Sistemi Operativi***, McGraw-Hill, 2004
- **Libri consigliati**:
  - A. Tanenbaum: ***I Moderni Sistemi Operativi***, Jackson Libri, 2002
  - H.M. Deitel, P.J. Deitel, D.R. Choffnes: ***Sistemi Operativi***, Pearson, 2005
  - W. Stallings: ***Sistemi Operativi***, Jackson Libri, 2000
  - K. Havilland, B. Salama: ***Unix System Programming***, Addison Wesley, 1987

Ricevimento Studenti

- **Paolo Bellavista**
  lunedì ore 16:00-18:00 e venerdì ore 11:00-13:00
  c/o nuovi studi – edificio aule nuove (di fianco aula 5.7)
  E-mail: **`pbellavista@deis.unibo.it`**
- **Eugenio Magistretti**
  mercoledì ore 14:00-16:00
  c/o LIA – Lab1, edificio aule nuove, piano terra
  E-mail: **`emagistretti@deis.unibo.it`**
- **Marco Montali**
  martedì ore 15:00-17:00
  c/o LIA – Lab1, edificio aule nuove, piano terra
  E-mail: **`mmontali@deis.unibo.it`**

Interazione docente-studenti

- **Ricevimento** (lunedì 16:00-18:00 e venerdì 11:00-13:00)
- **E-mail** **`pbellavista@deis.unibo.it`**
- **Lista di distribuzione del corso**: è un servizio del portale di ateneo che consente di inviare, via e-mail, comunicazioni, messaggi e materiali di approfondimento agli studenti
  - Accesso mediante lo stesso account della "mia e-mail" alla pagina:

`http://www.unibo.it/Portale/Servizi+online/Liste+distribuzione/default.htm`

nome della lista: **sola0506-info**

Orario delle Lezioni

Normalmente:

- **Lun 11-12 [ora Q], aula 6.2:** verrà utilizzata ogni 2 settimane, a partire da 8 maggio
- **Lun 12-14, aula 6.2**
- **Mar 11-14, aula 6.2**
- **Gio 09-11, aula 6.1:** solo le prime due settimane, poi sostituita da esercitazione in ***laboratorio Lab3*** (giovedì/venerdì, 09-11, in alternanza per lettere)

  - Eventuali variazioni verranno comunicate via sito Web e mailing list di distribuzione del corso

Università di Bologna

Corso di Studi: Laurea in Ingegneria Informatica

Sistemi Operativi L-A

A.A. 2005-2006

Prof. Paolo Bellavista

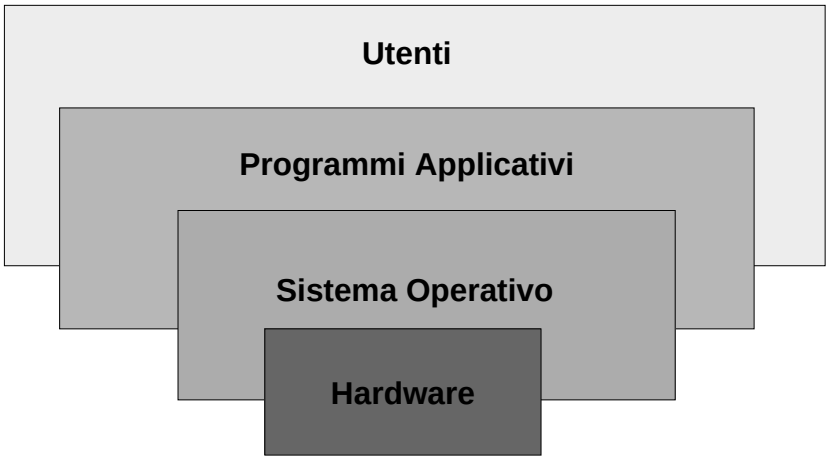
Che cos'è un Sistema Operativo?

- È un ***programma*** (o un insieme di programmi) che agisce come ***intermediario tra l'utente e l'hardware*** del computer:
  - fornisce un **ambiente di sviluppo e di esecuzione** per i programmi applicativi
  - fornisce una **visione astratta** dell'HW
  - **gestisce** efficientemente le risorse del sistema di calcolo

Il Sistema Operativo e l'Hardware

- Il Sistema Operativo (SO) interfaccia programmi applicativi o di sistema con le risorse HW:
  - **- CPU** **- memoria** volatile e persistente
  - **- dispositivi** di I/O - **connessione di rete**
  - **- dispositivi di comunicazione** **- …**
- SO *mappa* le risorse HW in **risorse logiche**, accessibili attraverso interfacce ben definite:
  - ***processi*** (CPU)
  - ***file system*** (dischi)
  - ***memoria virtuale*** (memoria), ...

Che cos'è un Sistema Operativo?



Che cos'è un Sistema Operativo (SO)?

- Un programma che ***gestisce risorse*** del sistema di calcolo in modo ***corretto ed efficiente*** e le ***alloca*** ai programmi/utenti

- Un programma che innalza il ***livello di astrazione*** con cui utilizzare le ***risorse logiche*** a disposizione

Aspetti importanti di un SO

- **Struttura**: come è organizzato un SO?
- **Condivisione**: quali risorse vengono condivise tra utenti e/o programmi? In che modo?
- **Protezione**: SO deve impedire ***interferenze*** tra programmi/utenti. In che modo?
- **Efficienza**: come massimizzare l'utilizzo delle risorse disponibili?
- **Affidabilità**: come reagisce SO a malfunzionamenti (HW/SW)?
- **Estendibilità**: è possibile aggiungere funzionalità al sistema?
- **Conformità a standard**: portabilità, estendibilità, apertura

Evoluzione dei Sistemi Operativi

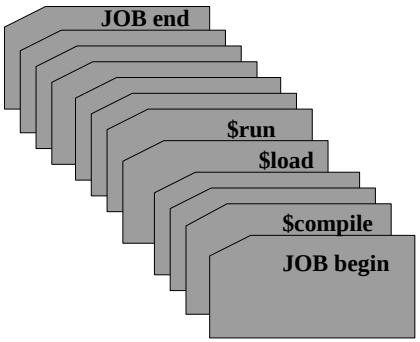
- **Prima generazione** (anni '50)
  - linguaggio macchina
  - dati e programmi su schede perforate

- **Seconda generazione** ('55-'65):

  sistemi batch semplici
  - linguaggio di alto livello (fortran)
  - input mediante schede perforate
  - aggregazione di programmi in **lotti** (batch) con esigenze simili

Sistemi batch semplici

Batch: insieme di programmi (*job*) da eseguire **<u>in modo sequenziale</u>**

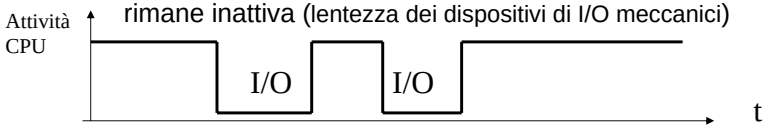


Sistemi batch semplici

Compito del Sistema Operativo (*monitor*):
trasferimento di controllo da un job (appena terminato) al prossimo da eseguire

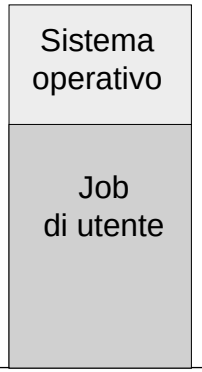
Caratteristiche dei sistemi batch semplici:

- SO ***residente in memoria*** (monitor)
- ***assenza di interazione*** tra utente e job
- ***scarsa efficienza***: durante l'I/O del job corrente, la CPU rimane inattiva (lentezza dei dispositivi di I/O meccanici)



Sistemi batch semplici

In memoria centrale, ad ogni istante, è ***caricato (al più) un solo job*:**



Configurazione della memoria centrale in sistemi batch semplici

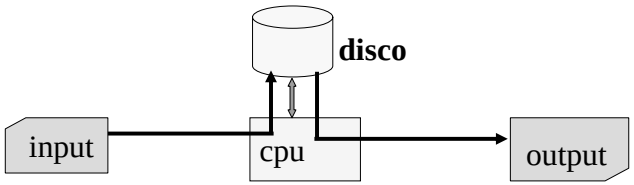
Sistemi batch semplici

Spooling

(Simultaneous Peripheral Operation On Line)

Obiettivo: **aumentare *l'efficienza*** del sistema

Avvento dei dischi + DMA: ⇒ ***I/O in parallelo con l'attività della CPU***

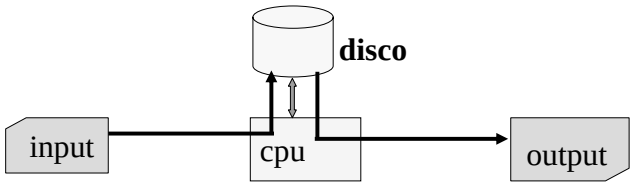


Sistemi batch semplici

Spooling: sumultaneità di I/O e attività di CPU

il disco viene impiegato come un **buffer** molto ampio, dove

- ***leggere*** in anticipo i dati
- ***memorizzare*** temporaneamente i risultati (in attesa che il dispositivo di output sia pronto)
- caricare ***codice e dati del job successivo***: -> possibilità di **sovrapporre I/O** di un job **con elaborazione** di un altro job



Sistemi batch semplici

Problemi:

- finché il job corrente non è terminato, il ***successivo non può iniziare l'esecuzione***
- se un job si ***sospende*** in attesa di un evento, la CPU rimane ***inattiva***
- ***non c'è interazione*** con l'utente

Sistemi batch multiprogrammati

Sistemi batch semplici: ***l'attesa*** di un ***evento*** causa inattività della CPU

⇒ **Multiprogrammazione**

Pool di job contemporaneamente presenti su disco:

- SO seleziona un sottoinsieme dei job appartenenti al pool da caricare in memoria centrale

  più job in memoria centrale
- mentre un job è in ***attesa di un evento***, il sistema operativo ***assegna CPU a un altro job***

Sistemi batch multiprogrammati

SO è in grado di ***portare avanti*** l'esecuzione di più job ***contemporaneamente***

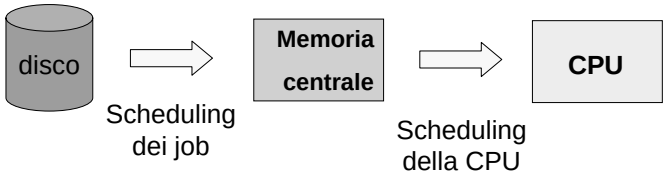
- Ad ogni istante:
  - **un solo job** utilizza la CPU
  - **più job,** appartenenti al pool selezionato e caricati in memoria centrale, attendono di acquisire la CPU
- Quando il job che sta utilizzando la CPU si ***sospende in attesa di un evento***:
  - SO ***decide*** a quale job assegnare la CPU ed effettua lo scambio (***scheduling***)

Sistemi batch multiprogrammati

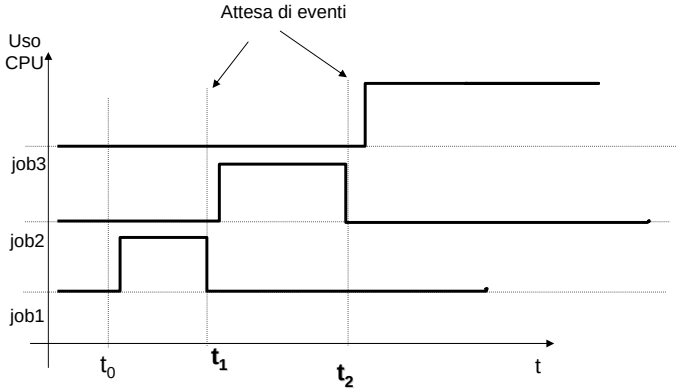
Scheduling

SO effettua delle scelte tra tutti i job

- quali job caricare in memoria centrale: **scheduling dei job** (*long-term scheduling*)
- a quale job assegnare la CPU: **scheduling della CPU** o (*short-term scheduling*)

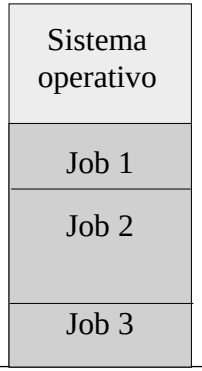


Sistemi batch multiprogrammati



Sistemi batch multiprogrammati

In memoria centrale, ad ogni istante, possono essere caricati più job:



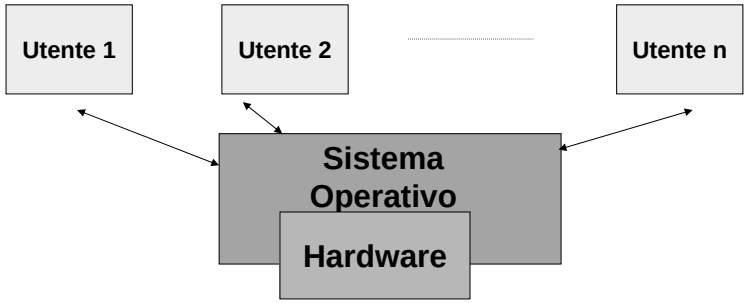
Configurazione della ***memoria centrale*** in sistemi batch multiprogrammati

Necessità di protezione

Sistemi Time-Sharing (Multics, 1965)

Nascono dalla necessità di:

- ***interattività*** con l'utente
- ***multi-utenza*:** più utenti interagiscono contemporaneamente con il sistema



Sistemi time-sharing

- **Multiutenza**: il sistema presenta ad ogni utente una ***macchina virtuale completamente dedicata*** in termini di
  - utilizzo della CPU
  - utilizzo di altre risorse, ad es. file system
- **Interattività**: per garantire un'accettabile velocità di "reazione" alle richieste dei singoli utenti, SO ***interrompe l'esecuzione*** di ogni job dopo un intervallo di tempo prefissato (***quanto di tempo***, o ***time slice***), e assegna la CPU ad un altro job

Sistemi time-sharing (oppure, a *divisione di tempo*)

Sono sistemi in cui:

- attività della ***CPU è dedicata a job diversi*** che si alternano ***ciclicamente*** nell'uso della risorsa
- frequenza di commutazione della CPU è tale da fornire l'illusione ai vari utenti di una macchina completamente dedicata (***macchina virtuale***)

Cambio di contesto (***context switch***):

operazione di trasferimento del controllo da un job al successivo ⇨ costo aggiuntivo (*overhead*)

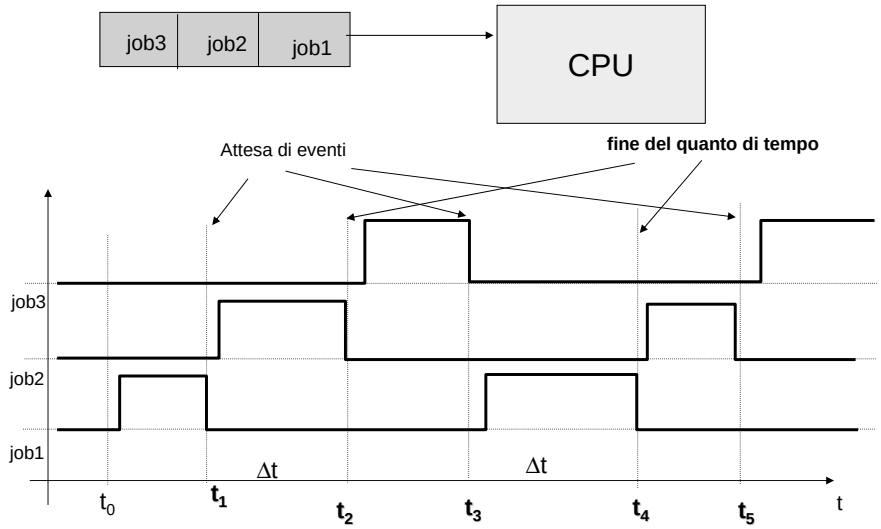
Sistemi time-sharing

Estensione dei sistemi multiprogrammati

Un job può sospendersi:

- perchè ***in attesa di un evento***
- perchè è ***terminato il quanto di tempo***

Sistemi time-sharing



Time-sharing: requisiti

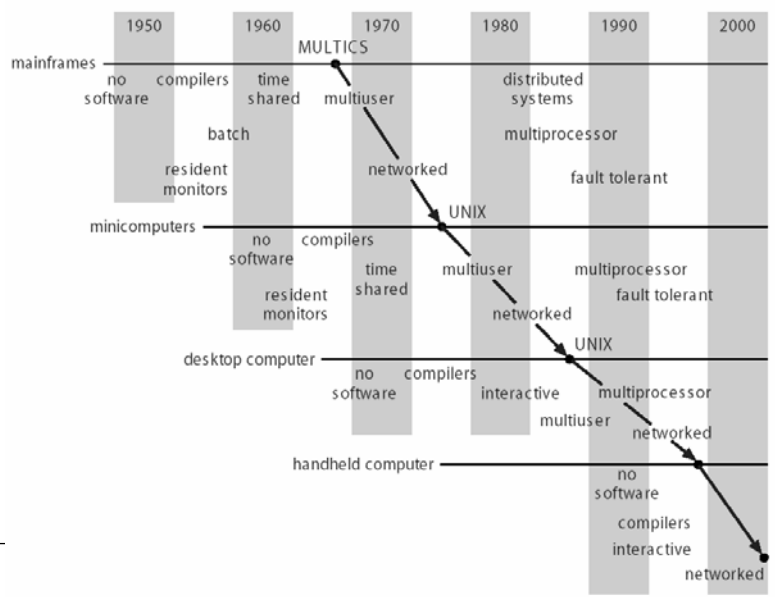
- **Gestione/protezione** della memoria:
  - trasferimenti memoria-disco
  - ***separazione degli spazi*** assegnati ai diversi job
  - molteplicità job + limitatezza della memoria

    ⇒ ***memoria virtuale***
- **Scheduling** CPU
- **Sincronizzazione/comunicazione** tra job:
  - interazione
  - prevenzione/trattamento di blocchi critici (***deadlock***)
- **Interattività**: ***file system on line*** per permettere agli utenti di accedere semplicemente a codice e dati

Sistemi operativi attuali

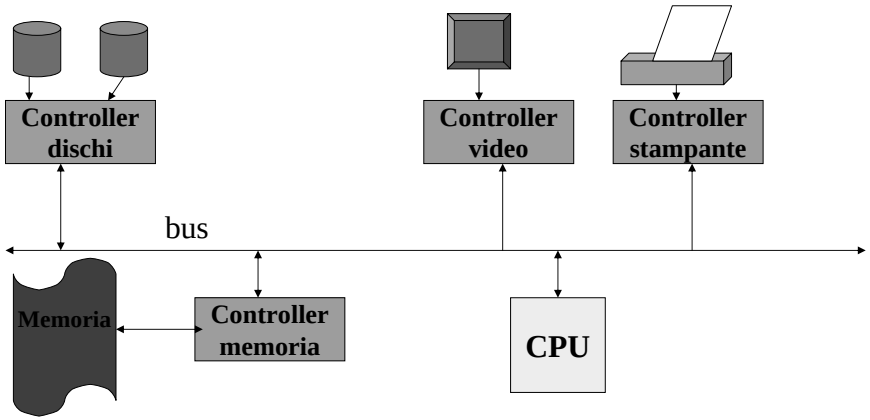
- **MSDOS**: monoprogrammato, monoutente
- **Windows 95/98, molti SO attuali per dispositivi portabili (Symbian, PalmOS)**: multiprogrammato (time sharing), tipicamente monoutente
- **Windows NT/2000/XP**: multiprogrammato, "multiutente"
- **MacOSX**: multiprogrammato, multiutente
- **UNIX/Linux**: multiprogrammato, multiutente

Evoluzione dei concetti nei SO



Rapidi richiami sul funzionamento di un sistema di elaborazione

Architettura di un sistema di calcolo



Controller: interfaccia HW delle periferiche il bus di sistema

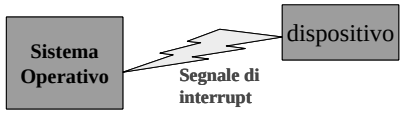
Funzionamento di un sistema di calcolo

Funzionamento a interruzioni:

- le varie *componenti* (HW e SW) del sistema interagiscono con SO mediante ***interruzioni*** **asincrone *(interrupt)***
- ogni interruzione è causata da un ***evento*****,** ad es:
  - richiesta di servizi al SO
  - completamento di I/O
  - accesso non consentito alla memoria
- ad ogni interruzione è associata una ***routine di servizio (handler)***, per la ***gestione dell'evento***

Funzionamento di un sistema di calcolo

- ***Interruzioni hardware***:
  dispositivi inviano segnali a SO per richiedere l'esecuzione di servizi di SO



- ***Interruzioni software***:
  programmi in esecuzione
  possono generare interruzioni SW
  - quando tentano l'esecuzione di ***operazioni non lecite*** (ad es., divisione per 0): ***trap***
  - quando richiedono l'esecuzione di servizi al SO - ***system call***

Gestione delle interruzioni

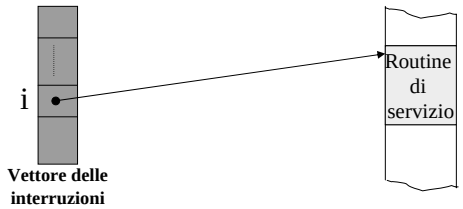
Alla ricezione di un'**interruzione**, SO:

1] interrompe la sua esecuzione => ***salvataggio dello stato*** in memoria (locazione fissa, stack di sistema, ...)

2] attiva la ***routine di servizio all'interruzione*** (handler)

3] ***ripristina lo stato*** salvato

Per individuare la routine di servizio, SO può utilizzare un **vettore delle interruzioni**



Input/Output

Come avviene l'I/O in un sistema di elaborazione?

Controller: interfaccia HW delle periferiche verso il bus di sistema

ogni controller è dotato di

- ***un buffer*** (ove ***memorizzare temporaneamente*** le informazioni da ***leggere*** o ***scrivere***)
- alcuni ***registri speciali***, ove ***memorizzare le specifiche delle operazioni*** di I/O da eseguire

Input/Output

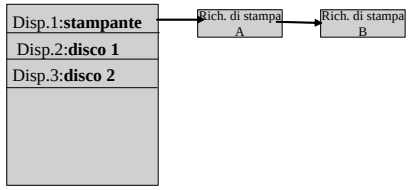
Quando un job richiede un'operazione di I/O (ad esempio, **lettura** da un dispositivo):

- CPU ***scrive nei registri speciali*** del dispositivo coinvolto le ***specifiche dell'operazione*** da eseguire
- controller esamina i registri e provvede a ***trasferire i dati richiesti dal dispositivo al buffer***
- invio di ***interrupt alla CPU*** (completamento del trasferimento)
- CPU esegue l'operazione di I/O tramite la routine di servizio (***trasferimento dal buffer del controller alla memoria centrale***)

Input/Output

2 tipi di I/O

- **Sincrono**: il ***job viene sospeso*** fino al completamento dell'operazione di I/O
- **Asincrono**: il sistema restituisce ***immediatamente il controllo al job***
  - operazione di ***wait*** per il completamento dell'I/O
  - possibilità di più I/O ***pendenti*** -> tabella di stato dei dispositivi



I/O asincrono = maggiore efficienza

Direct Memory Access

I/O asincrono: se i dispositivi di I/O sono **veloci** (tempo di trasferimento dispositivo-buffer paragonabile al tempo di esecuzione della routine di servizio)

→ l'esecuzione dei programmi può effettivamente ***riprendere soltanto al completamento tramite CPU dell'operazione di I/O***

Direct Memory Access (DMA):

è una tecnica che consente di ***migliorare l'efficienza*** del sistema durante le operazioni di I/O

Direct Memory Access

Il trasferimento tra memoria e dispositivo viene effettuato direttamente, ***senza intervento della CPU***

Introduzione di un dispositivo HW per controllare l'I/O: ***DMA controller***

- **driver di dispositivo:** componente del SO che
  - ***copia nei registri del DMA controller*** i dati relativi al trasferimento da effettuare
  - ***invia al DMA controller il comando*** di I/O
- **interrupt** alla CPU (inviato dal DMA controller) ***solo alla fine del trasferimento dispositivo/memoria***

Protezione HW degli accessi a risorse

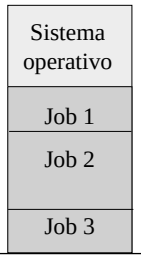
- Nei sistemi che prevedono multiprogrammazione e multiutenza sono necessari alcuni ***meccanismi HW (e non solo…) per esercitare protezione***
- Le risorse allocate a programmi/utenti devono essere protette nei confronti di ***accessi illeciti di altri programmi/utenti***:
  - ❑ dispositivi di I/O
  - ❑ memoria
  - ❑ CPU

Ad esempio: accesso a ***locazioni esterne allo spazio di indirizzamento del programma***

Protezione della memoria

In un sistema **multiprogrammato** o **time sharing,** ogni *job* ha un suo spazio di indirizzi:

➔ è necessario ***impedire al programma in esecuzione di accedere ad aree di memoria esterne al proprio spazio*** (ad esempio del SO oppure di altri *job*)



Se fosse consentito: un programma potrebbe modificare codice e dati di altri programmi o del SO

Protezione

Per garantire protezione, molte architetture prevedono un ***duplice modo di funzionamento (dual mode):***

- ❐ ***user*** mode
- ❐ ***kernel*** mode (***supervisor***, ***monitor*** mode)

Realizzazione: l'architettura prevede un ***bit di modo***

- **kernel: 0**
- **user: 1**

Dual mode

Istruzioni privilegiate*:** sono quelle più *pericolose* e possono essere eseguite soltanto se il sistema si trova in ***kernel mode

- accesso a dispositivi di I/O (dischi,schede di rete, ...)
- gestione della memoria (accesso a strutture dati di sistema per il controllo e l'accesso alla memoria, …)
- istruzione di ***shutdown*** (arresto del sistema)
- etc

- ❐ **SO esegue in modo kernel**
- ❐ **Ogni programma utente esegue in user mode:**
  - quando un ***programma utente tenta l'esecuzione di una istruzione privilegiata***, viene generato un ***trap***
  - se necessita di ***operazioni privilegiate*:**

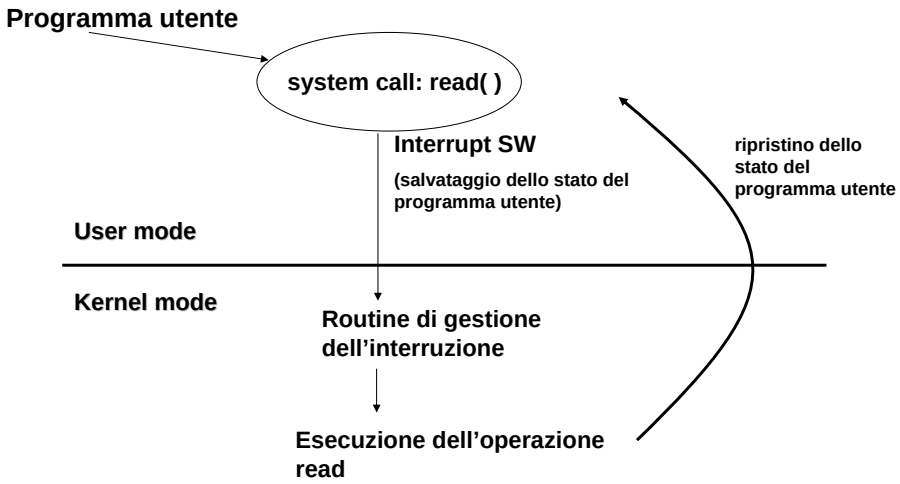
chiamata a ***system call***

System call

Per ottenere l'esecuzione di ***istruzioni privilegiate*, *un programma di utente deve chiamare una system call:***

- invio di ***un'interruzione software*** al SO
- ***salvataggio dello stato*** (PC, registri, bit di modo, ...) del programma chiamante e trasferimento del controllo al SO
- SO esegue in ***modo kernel*** l'operazione richiesta
- al termine dell'operazione, il controllo ritorna al programma chiamante (***ritorno al modo user***)

System call



System call

La ***comunicazione*** tra il programma chiamante e il sistema operativo avviene ***mediante i parametri della system call***: come vengono trasferiti?

- mediante ***registri*** (problema: dimensione limitata)
- mediante ***blocchi di memoria*** indirizzati da registri
- mediante ***stack di sistema***

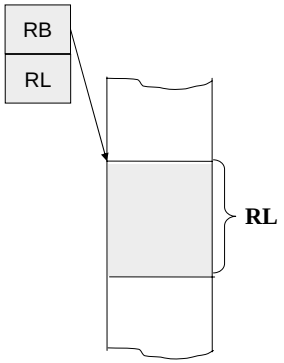
Protezione della memoria

SO deve fornire gli strumenti per separare e proteggere gli spazi di indirizzi dei programmi:

Registri base e limite

- memorizzano, per il programma in esecuzione (se viene allocato su parole contigue tra loro)
  - ***l'indirizzo della prima parola (RB)***
  - ***la dimensione (RL)***

  dello spazio degli indirizzi associato al programma
- HW può controllare ogni indirizzo, per verificare se appartiene all'intervallo **[RB, RB+RL]**



Protezione della CPU

SO deve anche occuparsi di evitare che un ***programma utente monopolizzi la CPU*** (ad es., loop):

- **uso di timer, per interrompere il programma dopo un intervallo di tempo prefissato (*time sharing*)**
- **allo scadere dell'intervallo:**

  interrupt ⇨ cambio di contesto

Introduzione all'Organizzazione dei Sistemi Operativi

Struttura dei SO

Quali sono le ***componenti*** di un SO?

Quali sono le ***relazioni mutue*** tra le componenti?

Componenti dei SO

- gestione dei ***processi***
- gestione della ***memoria centrale***
- gestione di ***memoria secondaria*** e ***file system***
- gestione dell'***I/O***
- ***protezione e sicurezza***
- interfaccia utente/programmatore

Processi

Processo = programma in esecuzione

- il ***programma*** è ***un'entità passiva*** (un insieme di byte contenente le istruzioni che dovranno essere eseguite
- ***il processo è un'entità attiva:***
  - è l'unità di lavoro all'interno del sistema. ***Ogni attività all'interno del SO è rappresentata da un processo***
  - ***è l'istanza di un programma in esecuzione***

Processo = programma +
contesto di esecuzione (PC, registri, …)

Gestione dei processi

In un sistema multiprogrammato: più processi possono essere simultaneamente presenti nel sistema

Compito cruciale del SO

- ***creazione/terminazione*** dei processi
- ***sospensione/ripristino*** dei processi
- ***sincronizzazione/comunicazione*** dei processi
- ***gestione del blocco critico (deadlock)*** di processi

Gestione della memoria centrale

HW di sistema di elaborazione è equipaggiato con ***un'unico spazio di memoria*** accessibile direttamente da CPU e dispositivi

Compito cruciale di SO

- ***separare gli spazi di indirizzi*** associati ai processi
- ***allocare/deallocare memoria*** ai processi
- ***memoria virtuale - gestire spazi logici di indirizzi di dimensioni complessivamente superiori allo spazio fisico***
- realizzare i collegamenti (***binding***) tra ***memoria logica e memoria fisica***

Gestione del file system

Ogni sistema di elaborazione dispone di uno o più dispositivi per la memorizzazione persistente delle informazioni (***memoria secondaria***)

Compito di SO

fornire una ***visione logica uniforme della memoria secondaria*** (indipendente dal tipo e dal numero dei dispositivi):

- realizzare il ***concetto astratto di file***, come unità di memorizzazione logica
- fornire una struttura astratta per ***l'organizzazione*** dei file (***direttorio***)

Gestione del file system

Inoltre, SO si deve occupare di:

- creazione/cancellazione di file e direttori
- manipolazione di file/direttori
- associazione tra file e dispositivi di memorizzazione secondaria

Spesso file, direttori e dispositivi di I/O vengono ***presentati*** a utenti/programmi ***in modo uniforme***

Gestione dei dispositivi di I/O

Gestione dell'I/O rappresenta una parte importante di SO:

- ***interfaccia*** tra programmi e dispositivi
- per ogni dispositivo: ***device driver***
  - ***routine per l'interazione con un particolare dispositivo***
  - contiene ***conoscenza specifica*** sul dispositivo (ad es., routine di gestione delle interruzioni)

Gestione della memoria secondaria

Tra tutti i dispositivi, la ***memoria secondaria*** riveste un ruolo particolarmente importante**:**

- ***allocazione/deallocazione*** di spazio
- gestione dello ***spazio libero***
- ***scheduling*** delle operazioni sul disco

Di solito:

- la **gestione dei file** usa i meccanismi di gestione della memoria secondaria
- la **gestione della memoria secondaria** è indipendente dalla gestione dei file

Protezione e sicurezza

In un sistema multiprogrammato, più entità (processi o utenti) possono utilizzare le risorse del sistema contemporaneamente: ***necessità di protezione***

Protezione: controllo dell'accesso alle risorse del sistema da parte di processi (e utenti) mediante

- ***autorizzazioni***
- ***modalità di accesso***

Risorse da proteggere:

- memoria
- processi
- file
- dispositivi

Protezione e sicurezza

Sicurezza:

se il sistema appartiene ad una rete, la ***sicurezza misura l'affidabilità del sistema nei confronti di accessi (attacchi) dal mondo esterno***

Non ce ne occuperemo all'interno di questo corso…

Interfaccia utente

SO presenta un'interfaccia che consente l'interazione con l'utente

- **interprete comandi** (***shell***): l'interazione avviene mediante una linea di comando
- **interfaccia grafica** (graphical user interface, ***GUI***): l'interazione avviene mediante ***click*** del mouse su elementi grafici; di solito è organizzata a finestre

Interfaccia programmatore

L'interfaccia del SO verso i processi è rappresentato dalle ***system call***:

- mediante la system call il ***processo richiede al sistema operativo*** l'esecuzione di un servizio
- la system call esegue ***istruzioni privilegiate***: passaggio da modo ***user*** a modo ***kernel***

Classi di system call:

- gestione dei processi
- gestione di file e di dispositivi (spesso trattati in modo omogeneo)
- gestione informazioni di sistema
- comunicazione/sincronizzazione tra processi

Programma di sistema **= programma che chiama system call**

Struttura e organizzazione di SO

Sistema operativo = insieme di componenti

- **gestione dei processi**
- **gestione della memoria centrale**
- **gestione dei file**
- **gestione dell'I/O**
- **gestione della memoria secondaria**
- **protezione e sicurezza**
- **interfaccia utente/programmatore**

Le componenti non sono indipendenti tra loro, ma interagiscono

Struttura e organizzazione di SO

Come sono organizzate le varie componenti all'interno di SO?

Vari approcci:

- **struttura *monolitica***
- **struttura *modulare: stratificazione***
- ***microkernel***

Struttura monolitica

SO è costituito da un ***unico modulo*** contenente un **insieme di procedure**, che realizzano le varie componenti:

l'interazione tra le componenti avviene mediante il meccanismo di chiamata a procedura

Ad esempio:

- MS-DOS
- prime versioni di UNIX

SO monolitici

Principale vantaggio: basso costo di interazione tra le componenti -> efficienza

Svantaggio: SO è un sistema complesso e presenta gli stessi requisiti delle applicazioni ***in-the-large***

- estendibilità
- manutenibilità
- riutilizzo
- portabilità
- affidabilità
- ...

Soluzione: organizzazione **modulare**

Struttura modulare

Le varie componenti del SO vengono organizzate in ***moduli caratterizzati da interfacce ben definite***

Sistemi stratificati (a livelli)

(THE, Dijkstra1968)

SO è costituito da ***livelli sovrapposti***, ognuno dei quali realizza un insieme di funzionalità:

- ogni livello realizza un'insieme di funzionalità che vengono ***offerte al livello superiore*** mediante un'interfaccia
- ogni livello ***utilizza le funzionalità offerte dal livello sottostante***, per realizzare altre funzionalità

Struttura a livelli

Ad esempio: **THE (5 livelli)**

livello 5: programmi di utente
livello 4: buffering dei dispositivi di I/O
livello 3: driver della console
livello 2: gestione della memoria
livello 1: scheduling CPU
livello 0: hardware

Struttura a livelli

Vantaggi

- ***Astrazione***: ogni livello è un oggetto astratto, che fornisce ai livelli superiori una visione astratta del sistema (***macchina virtuale***), limitata alle astrazioni presentate nell'interfaccia
- ***Modularità***: le relazioni tra i livelli sono chiaramente esplicitate dalle interfacce ⇨ possibilità di sviluppo, verifica, modifica in modo indipendente dagli altri livelli

Svantaggi

- ***organizzazione gerarchica*** tra le componenti: non sempre è possibile -> difficoltà di realizzazione
- ***scarsa efficienza*** (costo di attraversamento dei livelli)

Soluzione: limitare il numero dei livelli

Nucleo (kernel) di SO

È la parte di SO che esegue ***in modo privilegiato*** (modo ***kernel***)

- È la parte ***più interna*** di SO che si interfaccia direttamente con l'hardware della macchina
- Le funzioni realizzate all'interno del nucleo variano a seconda del particolare SO

Nucleo (kernel) di SO

Per un sistema multiprogrammato a divisione di tempo, il nucleo deve, almeno:

- gestire il ***salvataggio/ripristino dei contesti***
- realizzare lo ***scheduling della CPU***
- gestire le ***interruzioni***
- realizzare il meccanismo di ***chiamata a system call***

SO a microkernel

La struttura del nucleo è ridotta a ***poche funzionalità di base***:

- gestione della CPU
- della memoria
- meccanismi di comunicazione

il resto del SO è mappato su ***processi di utente***

Caratteristiche:

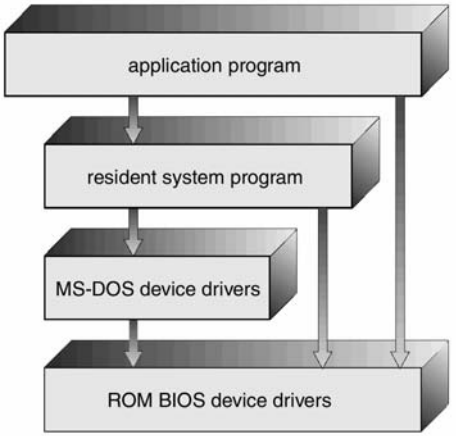
- ***affidabilità*** (separazione tra componenti)
- possibilità di ***estensioni*** e personalizzazioni
- ***scarsa efficienza*** (molte chiamate a system call)

Esempi: Mach, L4, Hurd, MSWindows

Una piccola panoramica: organizzazione di MS-DOS

MS-DOS – progettato per avere ***minimo footprint***

- ***non diviso in moduli***
- sebbene abbia una qualche struttura, ***interfacce e livelli di funzionalità non sono ben separati***

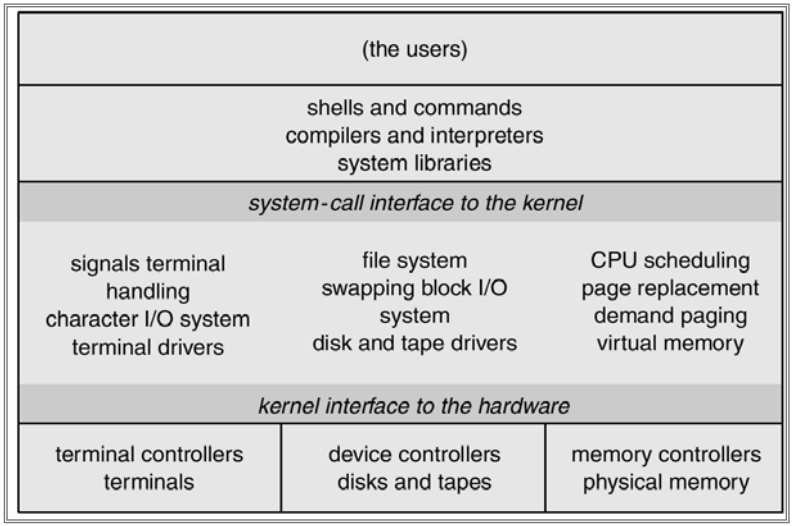


Una piccola panoramica: organizzazione di UNIX

UNIX – dati I limiti delle risorse hw del tempo, originariamente UNIX sceglie di avere una ***strutturazione limitata***. Consiste di due parti separabili:

- ***programmi di sistema***
- ***kernel***
  - costituito da tutto ciò che è sotto l'interfaccia delle system-call interface e sopra hw fisico
  - fornisce funzionalità di file system, CPU scheduling, gestione memoria, …; ***molte funzionalità tutte allo stesso livello***

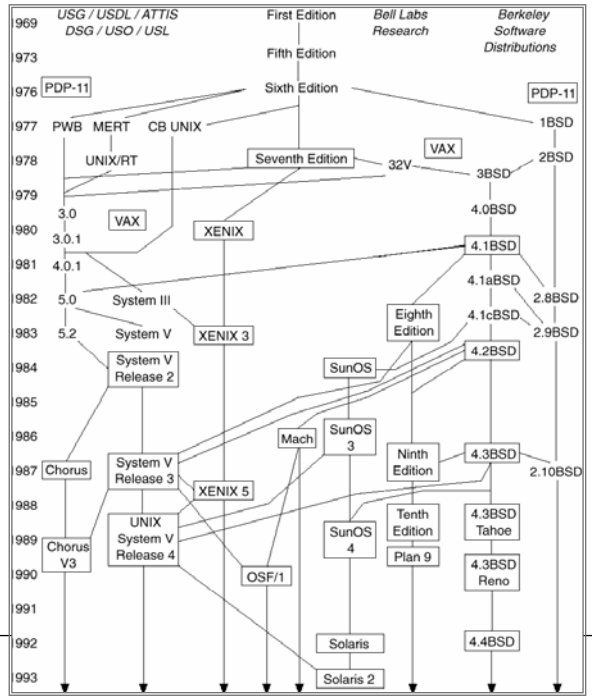
Organizzazione di UNIX



UNIX: qualche cenno storico

- Ken Thompson e Dennis Ritchie, gruppo di ricerca ai Bell Laboratories (1969). Raccolti diversi spunti dalle caratteristiche di altri SO contemporanei, specie ***MULTICS***
- Terza versione del sistema fu ***scritta in C***, ***specificamente sviluppato*** ai Bell Labs per supportare UNIX
- Gruppo di sviluppo UNIX più influente (escludendo Bell Labs e AT&T) - University of California at Berkeley (***Berkeley Software Distributions***):
  - ***4.0BSD UNIX*** fu il risultato di finanziamento DARPA per lo sviluppo di una ***versione standard*** di UNIX per usi governativi
  - ***4.3BSD UNIX***, sviluppato per VAX, influenzò molti dei SO successivi
- Numerosi progetti di ***standardizzazione*** per giungere ad una interfaccia di programmazione uniforme

⇒ ***POSIX***

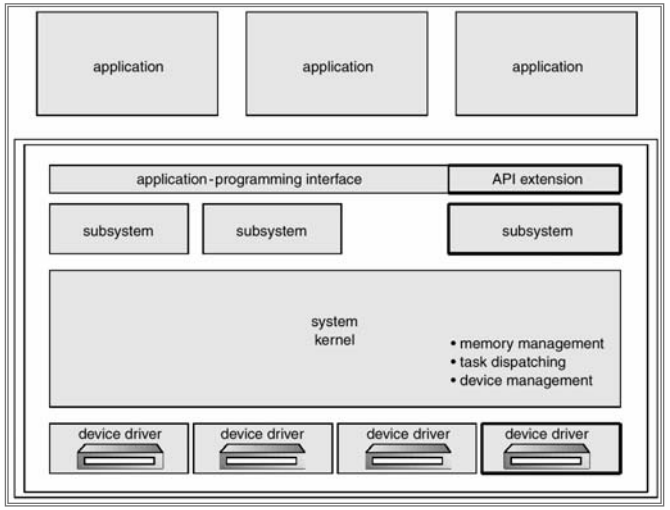


UNIX: principi di progettazione e vantaggi

- Progetto ***snello, pulito e modulare***
- Scritto in ***linguaggio di alto livello***
- Disponibilità codice sorgente
- ***Potenti primitive di SO*** su una piattaforma a ***basso prezzo***

- ❑ Progettato per essere ***time-sharing***
- ❑ ***User interface semplice (shell),*** anche sostituibile
- ❑ File system con ***direttori organizzati ad albero***
- ❑ ***Concetto unificante di file***, come ***sequenza non strutturata*** di byte
- ❑ Supporto semplice a ***processi multipli e concorrenza***
- ❑ Supporto ampio allo ***sviluppo di programmi*** applicativi e/o di sistema

Una piccola panoramica: organizzazione di OS/2



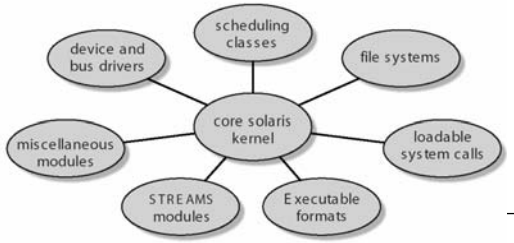
Buona strutturazione ***a livelli e modulare***

Modularità

La maggior parte dei moderni SO implementano il ***kernel in maniera modulare***

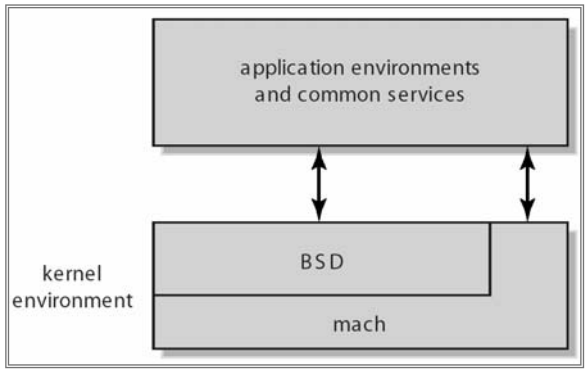
- ogni modulo core è ***separato***
- ogni modulo interagisce con gli altri tramite ***interfacce note***
- ogni modulo può essere ***caricato nel kernel quando*** e ove ***necessario***
- possono usare tecniche object-oriented

Strutturazione simile ai livelli, ma con ***maggiore flessibilità***



Esempio di SO Solaris di SUN

Una piccola panoramica: organizzazione di MacOS X



Esempio di organizzazione a ***micro-kernel***

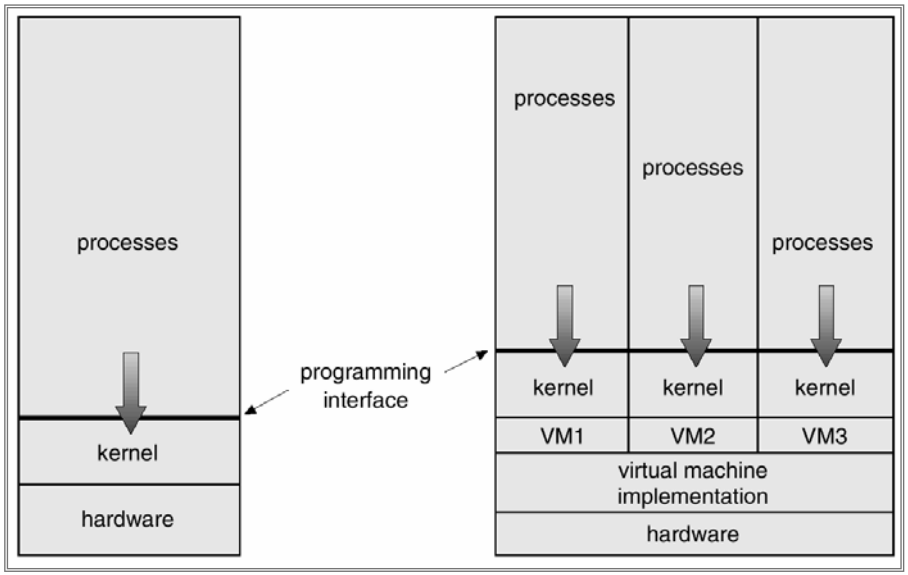
Alta modularità

Una parentesi: le macchine virtuali

Macchine virtuali *(VMWare, VirtualPC, Java?, .NET?)* sono la logica evoluzione dell'approccio a livelli. Virtualizzano come un tutt'uno sia hardware che kernel del SO

- SO crea l'illusione di ***processi multipli***, ciascuno in esecuzione sul suo ***processore privato*** e con la propria ***memoria virtuale privata***

- Ovviamente le ***risorse fisiche sono condivise*** fra le macchine virtuali:
  - ***CPU scheduling*** deve creare l'apparenza di processore privato
  - ***Spooling e file system*** devono fornire l'illusione di dispositivi di I/O virtuali privati



Non-virtual Machine Virtual Machine

Vantaggi/svantaggi delle macchine virtuali

- Il concetto di macchina virtuale permette la ***protezione completa*** delle risorse di sistema dato che ogni VM è ***isolata*** dalle altre. Tuttavia, ***questo isolamento non permette la condivisione diretta di risorse***
- Un sistema basato su VM è perfetto per fare ***ricerca, sviluppo e prototipazione di SO***. Infatti, lo sviluppo può essere fatto su una VM isolata ***senza interferire con la normale operatività delle altre VM*** nel sistema
- Macchina virtuale ***difficile da implementare*** (e ***problemi di efficienza***) dato lo sforzo di fornire un esatto duplicato della macchina sottostante

Una piccola panoramica: organizzazione di MSWindows2000

- SO ***multitasking e time-sharing a 32bit*** per microprocessori Intel
- Architettura a ***micro-kernel***
- Obiettivi primari:
  - portabilità
  - sicurezza
  - ***POSIX compliance***
  - supporto multiprocessore
  - estensibilità
  - ***back-compatibility*** con applicazioni MS-DOS e MS-Windows

Disponibile in varie versioni: Professional, Server, Advanced Server, …

MSWindows 2000: cenni storici

- MS ha deciso nel 1988 di sviluppare una “new technology” (NT) per un ***SO portabile*** capace di supportare sia le ***API OS/2 che POSIX***
- In origine, NT doveva utilizzare le API OS/2 API come ambiente nativo ma durante lo sviluppo si decise di passare alle API dette Win32, anche in risposta alla ***popolarità di MSWindows3.0***

Principi di progetto:

- ***Estensibilità*** — architettura a livelli
  - Livello executive esegue in ***protected mode*** e fornisce I servizi di SO di base
  - Altri sotto-sistemi operano in ***user mode*** sopra executive
  - ***Struttura modulare*** permette ***l’aggiunta di sotto-sistemi*** senza impatto su executive

Windows2000: principi di progetto (cont)

- ***Portabilità***
  - Sviluppato in C e C++
  - Codice dipendente dal processore isolato in una dynamic link library (DLL) chiamata Hardware Abstraction Layer (HAL)
- ***Reliability*** — Win2000 sfrutta protezione hw per memoria virtuale e meccanismi di protezione sw per le risorse del SO
- ***Compatibilità POSIX*** — conformità completa a IEEE 1003.1 (POSIX) standard
- ***Performance*** — comunicazione mediante ***high-performance message passing***
  - Preemption di thread a bassa priorità
- ***Supporto a linguaggi diversi*** – differenti ***linguaggi locali*** mediante national language support (NLS)

Windows 2000: architettura

- ***Moduli*** organizzati in una ***struttura a livelli***
- ***Protected mode*** — HAL, kernel, executive
- ***User mode*** — insieme di sotto-sistemi
  - ***Sotto-sistemi di ambiente*** possono ***emulare diversi SO***
  - ***Sotto-sistemi di protezione*** possono fornire funzionalità di sicurezza