

Prof. Paolo Bellavista

<http://lia.deis.unibo.it/Courses/sola0708-info/>
<http://lia.deis.unibo.it/Staff/PaoloBellavista/>

Obiettivi del Corso

- Fornire i principali concetti alla base della **teoria dei sistemi operativi**
- Illustrare le caratteristiche di **sistemi operativi reali (UNIX/Linux e MSWinXP)** e gli strumenti a disposizione di utenti e programmatori per il loro utilizzo
- Sperimentare ampiamente in **laboratorio** i concetti e gli strumenti visti in aula tramite **numerosi esercizi di programmazione C e Java**

Capacità richieste in ingresso:

- conoscenza dei **linguaggi C e Java**
- 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 principalmente in ambiente UNIX/Linux (e in parte MSWinXP)

Argomenti trattati

- Che cos'è un sistema operativo: ruolo e funzionalità
- **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 (prima settimana):

- 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 (prima settimana):

- 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 **MSWindowsXP**

Panoramica sul Corso

Processi e thread (seconda-quarta settimana):

- **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
- **Java thread e memoria condivisa**

Panoramica sul Corso

Scheduling della CPU (parte della quarta settimana):

- 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 MSWinXP

Panoramica sul Corso

Interazione tra processi (quinta e sesta settimana):

- **Mediante memoria condivisa**
Cenni sul problema della **sincronizzazione tra processi**. Sezione critica, mutua esclusione, semafori
- **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 (parte della settima settimana):

- 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 (settima e ottava settimana):

- 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
- **File comandi bash shell**: esercizi

Panoramica sul Corso

Cenni di problematiche e soluzioni per la protezione (forse ottava settimana):

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

Percorso didattico

- **Argomenti teorici**
- **Esemplificazioni:** sui sistemi operativi UNIX/Linux e MSWinXP, sia tramite *programmazione di sistema in linguaggio C* che tramite sviluppo di *file comandi in shell* e di *applicazioni concorrenti in Java*
- **Esercitazioni:**
 - ⇒ **Attività OBBLIGATORIA in laboratorio**

Anche possibilità di progetto opzionale di approfondimento finale (per alleggerire ☺ la prova orale)

Attività in laboratorio

- Esattamente come le lezioni in aula, è **parte integrante dell'attività didattica!**
- Ogni settimana verrà svolta in Lab4 una esercitazione con una **prima piccola parte di lezione di programmazione** e una **seconda parte di risoluzione assistita di esercizi proposti**
- Le esercitazioni cominciano **giovedì 24 aprile**
- L'attività sarà assistita da un **tutor**:
 - **Ing. Stefano Monti**

Accesso al Laboratorio

- L'attività si svolgerà in sala terminali (Lab4) su sistemi dual-boot GentooLinux/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/sola0708-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 per la maggior parte di sviluppo di codice in Lab4):
venerdì 20 giugno 2008, ore 13:00
venerdì 18 luglio 2008, ore 13:00

- Una **prova orale** facoltativa dopo il superamento dello scritto (sostituibile con **progetto di approfondimento**, anche a piccoli gruppi di 2 allievi)
- Valutazione della partecipazione in aula e in LABORATORIO; **ultime 2 esercitazioni con consegna**

Materiale Didattico

- **Copia** delle diapositive mostrate a lezione (scaricabili mano a mano dalle pagine 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, seconda edizione, 2008
- **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: paolo.bellavista@unibo.it
- **Stefano Monti**
giovedì ore 15:00-17:00
c/o Lab2, edificio aule nuove, piano terra
Qualora si intenda presentarsi a ricevimento, si prega di contattare anticipatamente il tutor tramite e-mail
E-mail: stefano.monti@unibo.it

Interazione docente-studenti

- **Ricevimento** (lunedì 16:00-18:00 e venerdì 11:00-13:00)
- **E-mail** paolo.bellavista@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: **sola0708-info**

Orario delle Lezioni

Normalmente:

- **Lun 11-12 [ora Q], aula 6.2:** verrà utilizzata a settimane alterne, tutte le settimane dispari
- **Lun 12-14, aula 6.2**
- **Mar 11-14, aula 6.2**
- Esercitazioni in **laboratorio Lab4** (giovedì/venerdì, 09:00-11:00, in alternanza per lettere)
 - Eventuali variazioni verranno comunicate via sito Web e mailing list di distribuzione del corso

Prof. Paolo Bellavista

<http://lia.deis.unibo.it/Courses/sola0708-info/>
<http://lia.deis.unibo.it/Staff/PaoloBellavista/>

Che cos'è un Sistema Operativo (SO)?

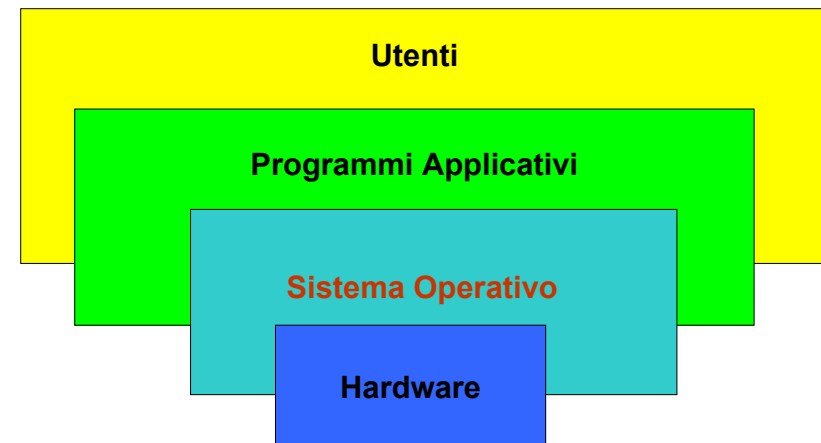
È un **programma** (o un insieme di programmi) che agisce come **intermediario tra l'utente e l'hardware** del computer:

- 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

SO e Hardware

- 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?

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

Evoluzione SO

Prima generazione (anni '50)

- linguaggio macchina
- dati e programmi su schede perforate

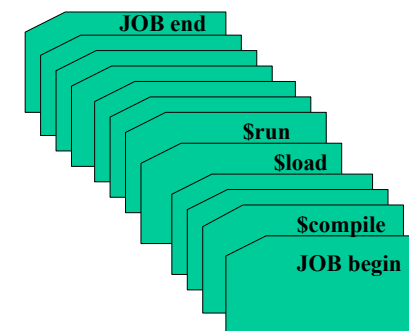
Seconda generazione ('55-'65):

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 **in modo sequenziale**

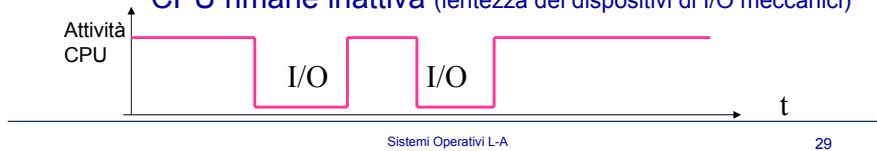


Sistemi batch semplici

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

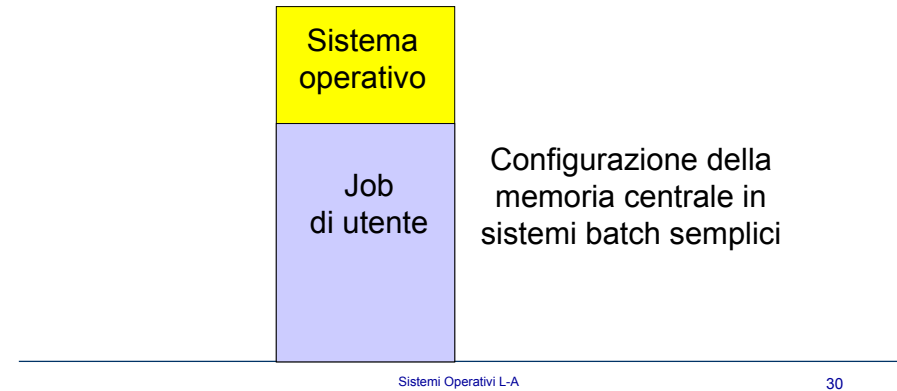
Caratteristiche dei sistemi batch semplici:

- 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**:

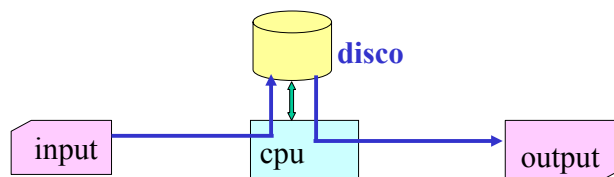


Sistemi batch semplici

Spooling (Simultaneous Peripheral Operation On Line):
simultaneità di I/O e attività di CPU

disco viene impiegato come **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. Per evitare il problema

⇒ **Multiprogrammazione**

Pool di job contemporaneamente presenti su disco:

- SO seleziona un **sottoinsieme dei job** appartenenti al pool da **caricare in memoria centrale**
- mentre un job è in **attesa di un evento**, il sistema operativo **assegna CPU a un altro job**

Sistemi batch multiprogrammati

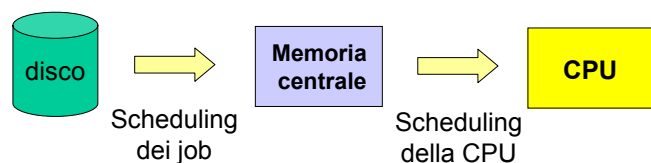
SO è in grado di **portare avanti** l'esecuzione di più job **contemporaneamente**

- 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**)

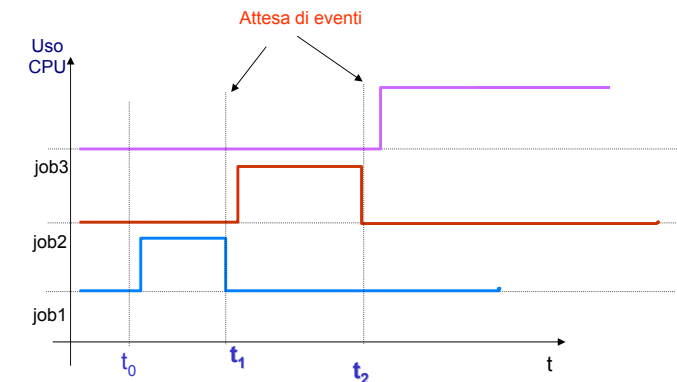
Sistemi batch multiprog.: scheduling

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*)

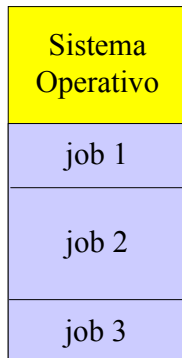


Sistemi batch multiprogrammati



Sistemi batch multiprogrammati

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



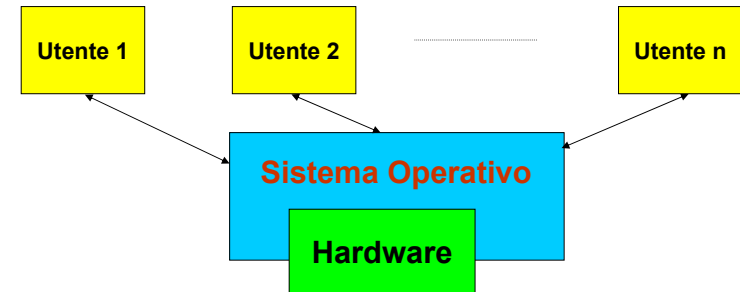
Configurazione della **memoria centrale** in sistemi batch multiprogrammati

Necessità di protezione

Sistemi time-sharing (Multics, 1965)

Nascono dalla necessità di:

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



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**), assegnando la CPU a un altro job

Sistemi time-sharing

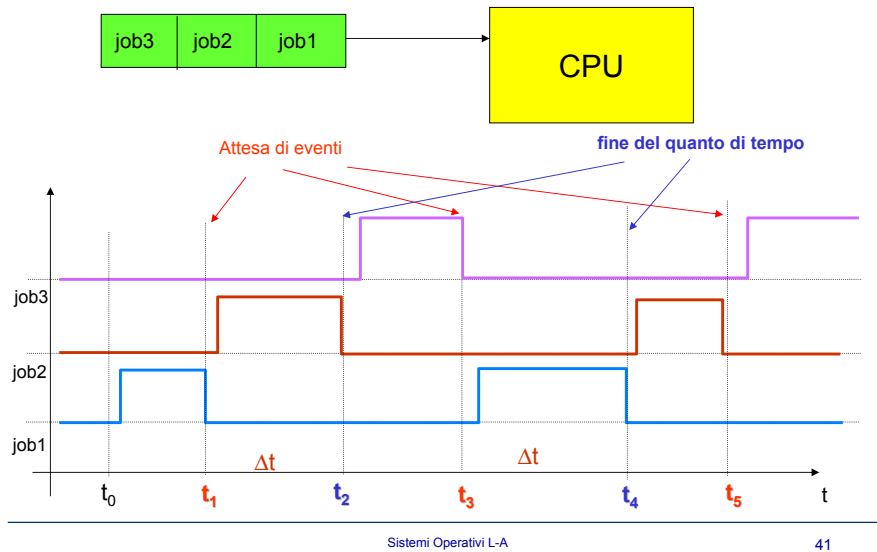
Sono sistemi in cui:

- 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



Time-sharing: requisiti

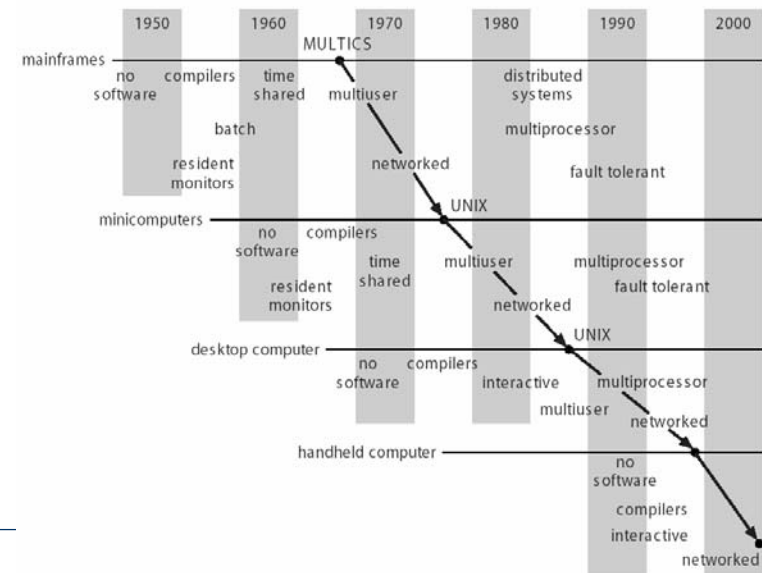
- **Gestione/protezione della memoria:**
 - trasferimenti memoria-disco
 - **separazione degli spazi** assegnati ai diversi job
 - molteplicità job + limitatezza della memoria

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

Esempi di SO attuali

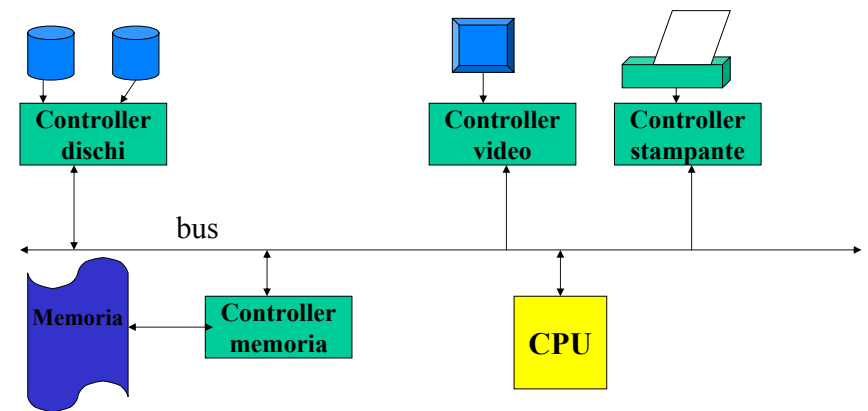
- **MSDOS:** monoprogrammato, monoutente
- **Windows 95/98, molti SO attuali per dispositivi portabili (Symbian, PalmOS):** multiprogrammato (time sharing), tipicamente monoutente
- **Windows NT/2000/XP:** multiprogrammato, "multiutente"
- **MacOSX:** multiprogrammato, multiutente
- **UNIX/Linux:** multiprogrammato,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

Evoluzione dei concetti nei SO



Rapidi richiami su alcuni concetti di base riguardo al funzionamento hardware di un sistema di elaborazione

Architettura di un sistema di elaborazione



Controller: interfaccia HW delle periferiche il bus di sistema

Hardware di un sistema di elaborazione

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*

Interruzioni hardware e software

- **Interruzioni hardware:**
dispositivi inviano segnali 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**

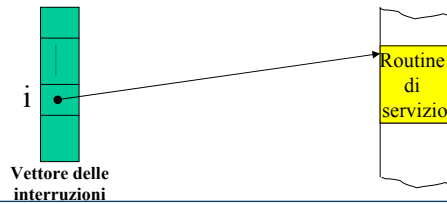


Gestione delle interruzioni

Alla ricezione di un'interruzione, SO (lo vedremo in seguito nel dettaglio per il cambio di contesto):

- 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**):

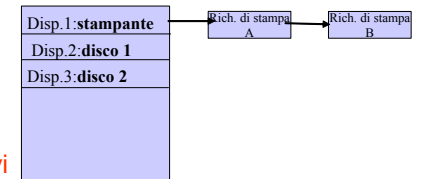
- ❑ CPU **scrive nei registri speciali** del dispositivo coinvolto le **specifiche dell'operazione** da eseguire
- ❑ controller esamina i registri e provvede a **trasferire i dati richiesti dal dispositivo al buffer**
- ❑ invio di **interrupt alla CPU** (completamento del trasferimento)
- ❑ CPU esegue 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** in attesa del completamento dell'operazione di I/O
- ❑ **Asincrono:** il sistema restituisce **immediatamente il controllo al job**

- ❑ se necessario, funzionalità di blocco in attesa di completamento dell'I/O
- ❑ possibilità di più I/O **pendenti**
-> **tabella di stato dei dispositivi**



I/O asincrono = maggiore efficienza

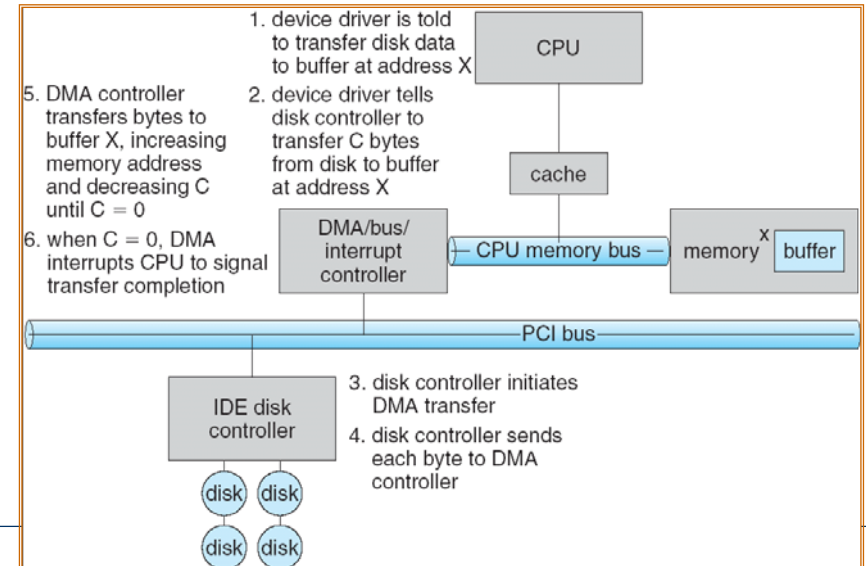
Direct Memory Access

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

Introduzione di un dispositivo HW per controllare 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 comando* richiesto *al DMA controller*
- **interrupt** alla CPU (inviato dal DMA controller) solo alla fine del trasferimento dispositivo -> memoria, usualmente di grandi quantità di dati

Passi per effettuare DMA Transfer



Protezione HW degli accessi a risorse

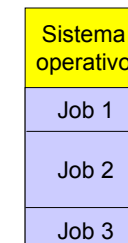
- Nei sistemi che prevedono multiprogrammazione e multiutenza sono necessari alcuni **meccanismi HW** (e non solo...) **per esercitare protezione**
- Le risorse allocate a programmi/utenti devono essere protette nei confronti di **accessi illeciti di altri programmi/utenti:**
 - dispositivi di I/O
 -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

Protezione

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

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

Realizzazione: l'architettura hardware della CPU 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 controllo e accesso alla memoria, ...)
- istruzione di **shutdown** (arresto del sistema)
- ...

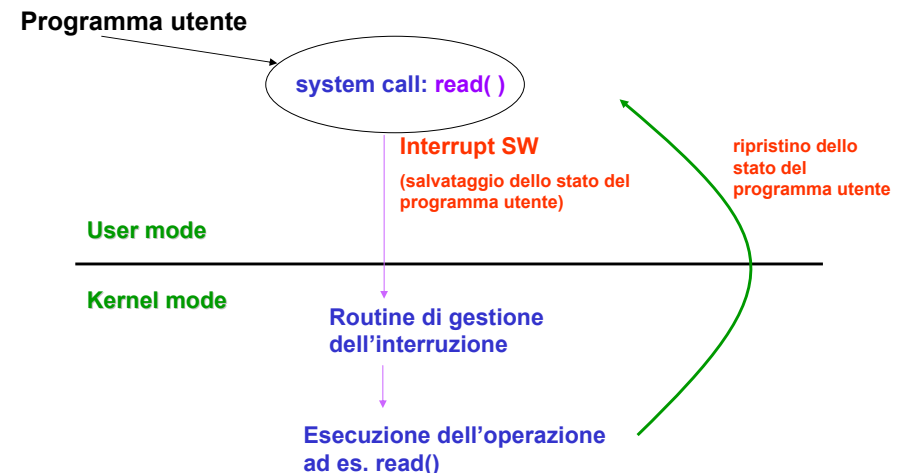
- **SO esegue in modo kernel**
- **Ogni programma utente esegue in user mode:**
 - quando un **programma utente tenta l'esecuzione di una istruzione privilegiata**, viene generato un **trap**
 - se necessita di **operazioni privilegiate**:
chiamata a system call

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



Introduzione all'Organizzazione dei Sistemi Operativi

Struttura dei SO

Quali sono le **componenti** di un 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

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

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/esecuzione** 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 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

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**

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 a 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 **interazione** mouse-elementi grafici su desktop; 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 a SO** 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

Visto che le varie componenti di un SO sono interagenti, come sono organizzate nella struttura di un 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, Dijkstra 1968)

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:** relazioni tra 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 (context-switching)*
- 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 CPU
- gestione memoria
- gestione meccanismi di comunicazione I/O

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

Caratteristiche:

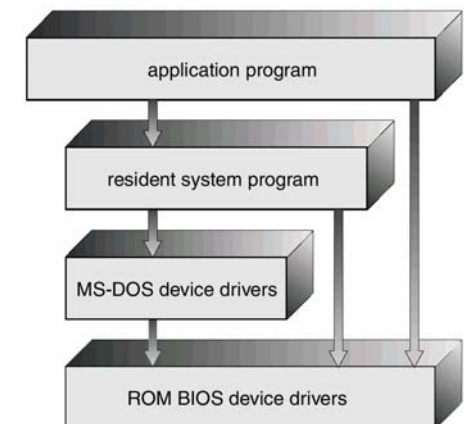
- *affidabilità* (separazione tra componenti)
- possibilità di *estensioni* e personalizzazioni
- *scarsa efficienza* (molte chiamate a system call)

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*

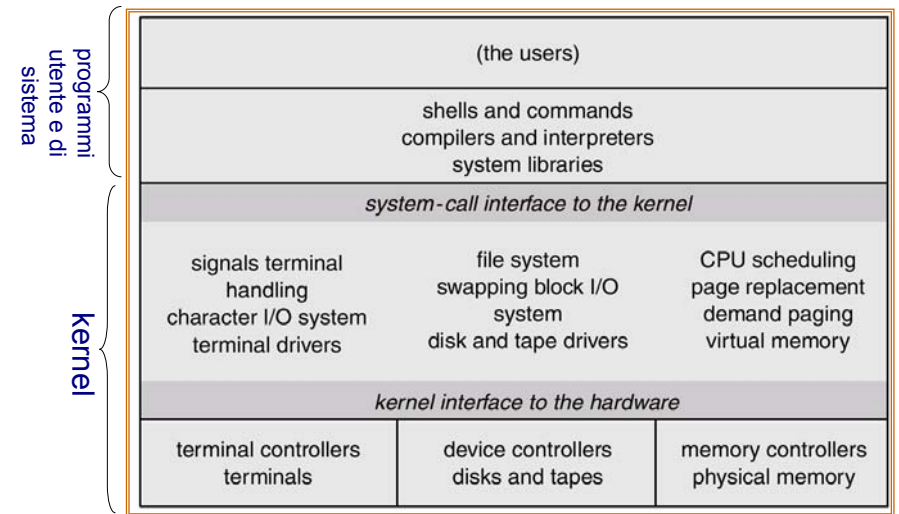


Una piccola panoramica: organizzazione di UNIX

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

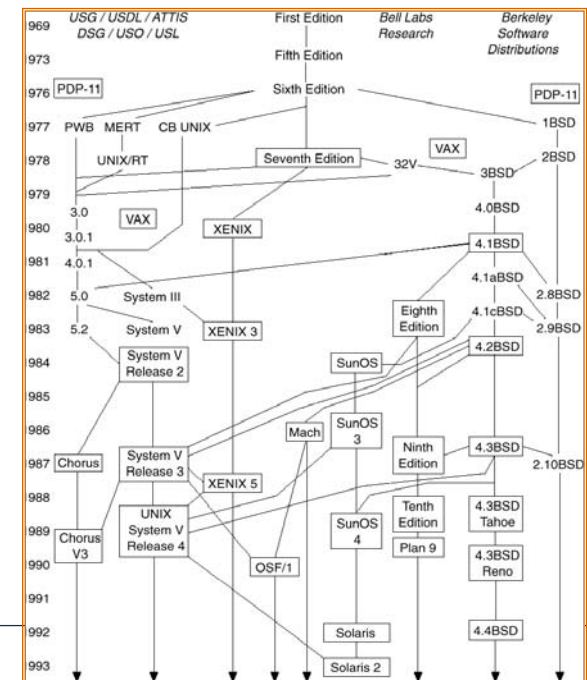
- **programmi di sistema**
- **kernel**
 - costituito da tutto ciò che è sotto l'interfaccia delle system-call interface e sopra hw fisico
 - fornisce funzionalità di file system, CPU scheduling, gestione memoria, ...; **molte funzionalità tutte allo stesso livello**

Organizzazione di UNIX



UNIX: qualche cenno storico

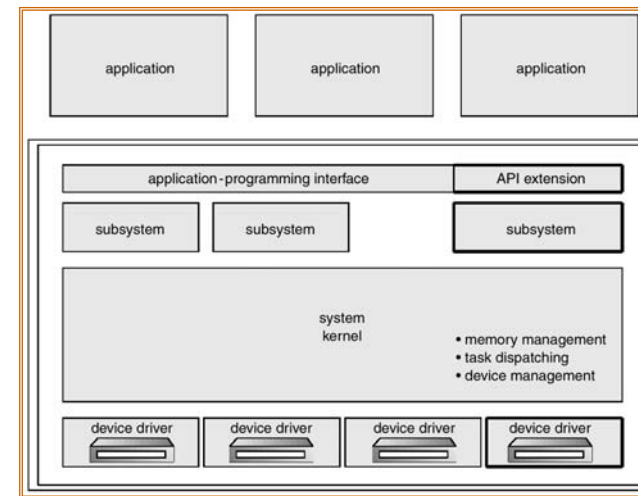
- Thompson e Ritchie, Bell Laboratories (1969). Raccolti diversi spunti dalle caratteristiche di altri SO contemporanei, specie **MULTICS**
- Terza versione del sistema **scritta in C, specificamente sviluppato** ai Bell Labs per supportare e implementare UNIX
- Gruppo di sviluppo UNIX più influente (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
 - **4.3BSD UNIX**, sviluppato per VAX, influenzò molti dei SO successivi
- Numerosi progetti di **standardizzazione** per giungere a interfaccia di programmazione uniforme



UNIX: principi di progettazione e vantaggi

- Progetto **snello, pulito e modulare**
- Scritto in **linguaggio di alto livello** (linguaggio C)
- 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



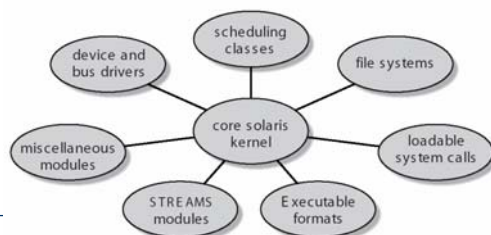
Buona
strutturazione
a livelli e
modulare

Modularità

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

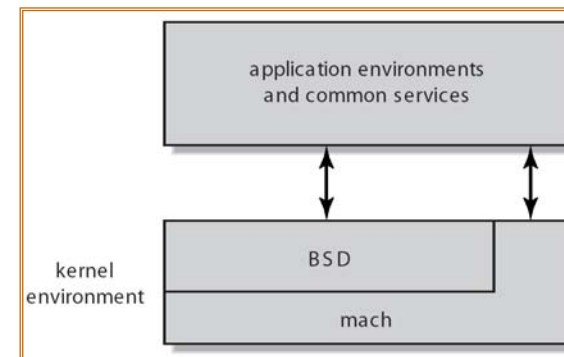
- ogni modulo core è **separato**
- ogni modulo interagisce con gli altri tramite **interfacce note**
- ogni modulo può essere **caricato nel kernel quando** e ove **necessario**
- possono usare tecniche object-oriented

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 piccola panoramica: MSWinXP

Rapida storia delle versioni: anche se il nome è cambiato, internamente SO viene identificato da un numero di build

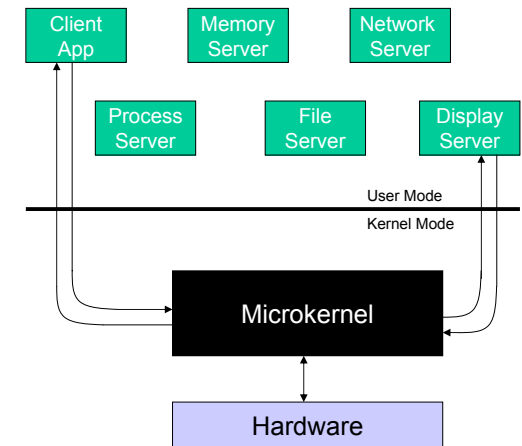
Build#	Versione	Data
297	PDC developer release	Luglio 1992
511	NT 3.1	Luglio 1993
807	NT 3.5	Sett 1994
1057	NT 3.51	Maggio 1995
1381	NT 4.0	Luglio 1996
2195	Windows 2000 (NT 5.0)	Dic 1999
2600	Windows XP (NT 5.1)	Ago 2001
3790	Windows Server 2003 (NT 5.2)	Mar 2003
4051	Longhorn PDC Developer Preview	Ott2003
6000	Windows Vista	Nov 2006

MSWinXP: SO Microkernel

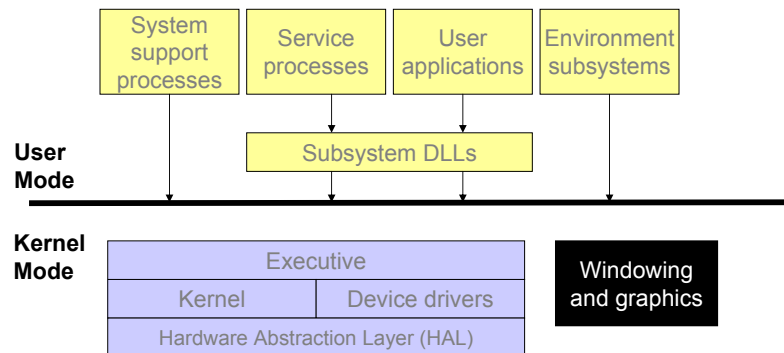
Kernel implementa:

- Scheduler
- Gestore della memoria
- Interprocess communication (IPC)

Server in user-mode



Architettura di WinXP: vista semplificata

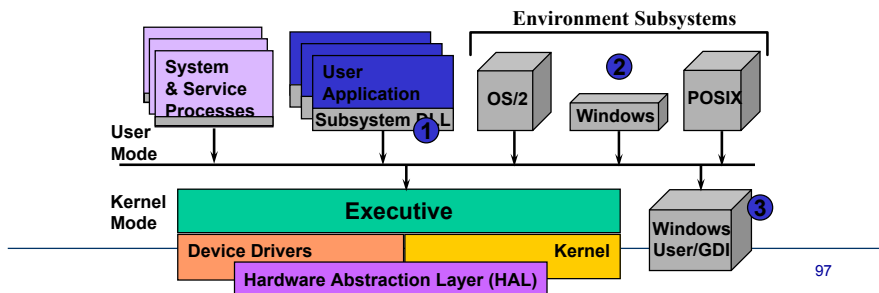


Cenni di architettura WinXP

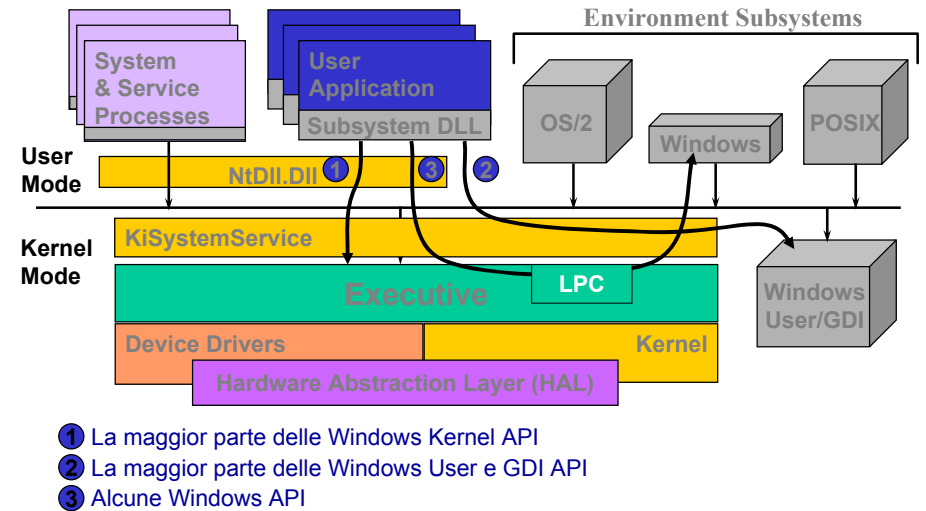
- Progettato per avere più “personalità”
 - Applicazioni utente non chiamano servizi di sistema direttam.
- **DLL di sottosistema** per tradurre una funzione nella corrispondente chiamata di sistema interna
- Processi di **Sottosistema (Environment Subsystem)**
 - Espongono una serie di funzionalità sottostanti alle applic.
 - Possono fare cose diverse nei diversi sottosistemi (e.g., POSIX fork)
- Originariamente tre sottosistemi: Windows, POSIX e OS/2
 - Windows 2000 include solo sottosistemi Windows e POSIX
 - Windows XP/Vista include solo il sottosistema Windows
 - “Services for Unix” offrono un sottosistema POSIX
 - Inclusi in Windows Server 2003 R2

Componenti di sottosistema

- 1 DLL per le API
 - per Windows: Kernel32.DLL, Gdi32.DLL, User32.DLL, etc.
- 2 Processi di sottosistema
 - per Windows: CSRSS.EXE (Client Server Runtime SubSystem)
- 3 Solo per Windows: kernel-mode GDI code
 - Win32K.SYS - (il codice era originariamente parte di CSRSS)



Comunicazione applicazioni con SO



- 1 La maggior parte delle Windows Kernel API
- 2 La maggior parte delle Windows User e GDI API
- 3 Alcune Windows API

File importanti

Componenti core:

- NTOSKRNL.EXE Executive e kernel
- HAL.DLL Hardware abstraction layer
- NTDLL.DLL funzioni interne di supporto e stub verso funzioni dell'executive

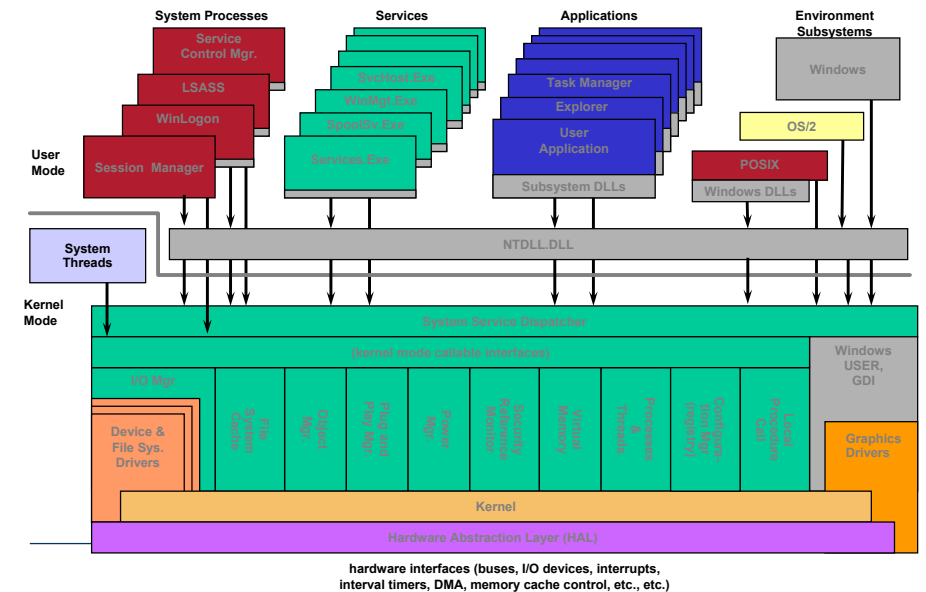
Processi di sistema Core:

- SMSS.EXE Session manager
- WINLOGON.EXE processo di Logon
- SERVICES.EXE processo per gestione dei Servizi
- LSASS.EXE Local Security Authority Subsystem
- WININIT.EXE (in Vista) processo per start-up applicazioni
- LSM.EXE (in Vista) Local Session Manager

Sottosistema Windows:

- CSRSS.EXE Windows subsystem
- WIN32K.SYS Componenti kernel-mode di USER e GDI
- KERNEL32/USER32/GDI32.DLL DLL del sottosistema Windows

WinXP: architettura complessiva

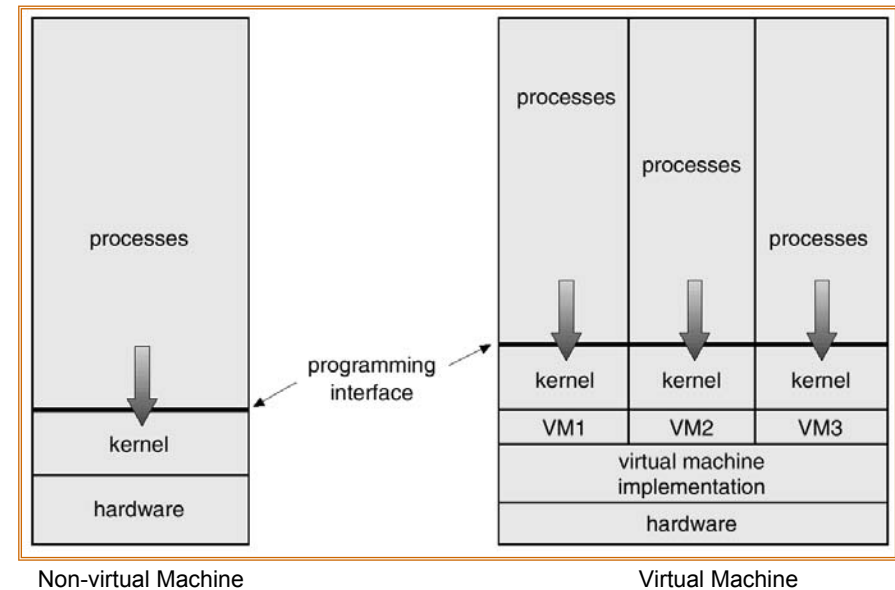


hardware interfaces (buses, I/O devices, interrupts, interval timers, DMA, memory cache control, etc., etc.)

Una panoramica: le macchine virtuali

Macchine virtuali (VMWare, VirtualPC, 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 SO, che può essere diverso per processi diversi**
- 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



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 rapida prototipazione di SO e di applicazioni multi-piattaforma**. 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