

# Trasparenze su Sistemi Operativi I

Copyright © 2000-03 Marino Miculan (miculan@dimi.uniud.it)

La copia letterale e la distribuzione di questa presentazione nella sua integrità sono permesse con qualsiasi mezzo, a condizione che questa nota sia riprodotta.

2004-05 Modificati ed integrati con altro materiale da Giorgio Delzanno su autorizzazione di Marino Miculan

1

## Introduzione

- Cosa è un sistema operativo?
- Evoluzione dei sistemi operativi
- Tipi di sistemi operativi
- Concetti fondamentali
- Chiamate di sistema
- Struttura dei Sistemi Operativi

2

## Cosa è un sistema operativo?

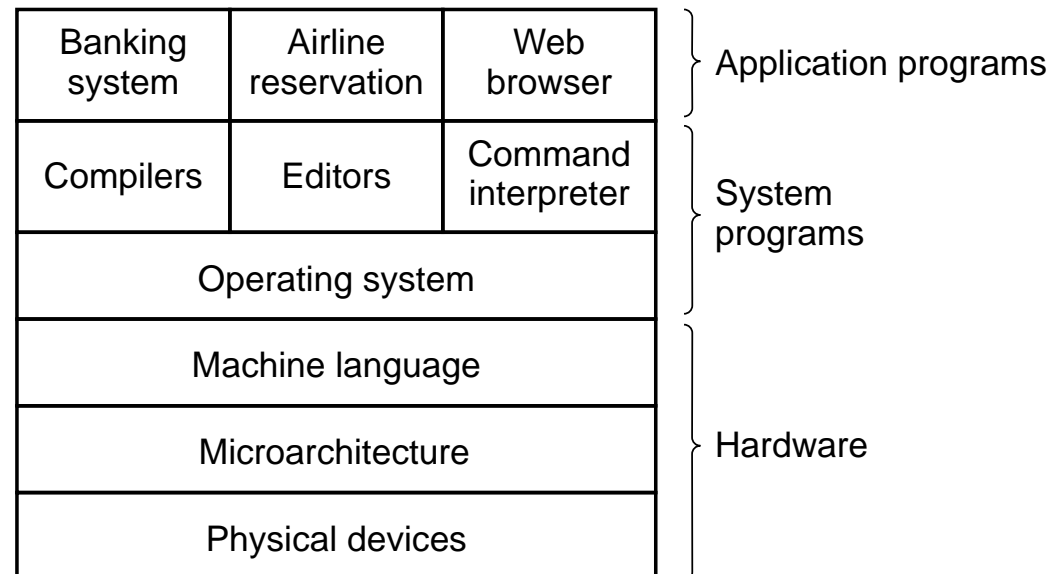
Possibili risposte:

- È un *programma di controllo*
- È un *gestore di risorse*
- È un *fornitore di servizi*
- ...

Nessuna di queste definizioni è completa

3

## Visione astratta delle componenti di un sistema di calcolo



4

## Componenti di un sistema di calcolo

1. Hardware – fornisce le risorse computazionali di base: (CPU, memoria, dispositivi di I/O).
2. Sistema operativo – controlla e coordina l'uso dell'hardware tra i programmi applicativi per i diversi utenti
3. Programmi applicativi — definiscono il modo in cui le risorse del sistema sono usate per risolvere i problemi computazionali dell'utente (database, videogiochi, programmi di produttività personale, . . . )
4. Utenti (persone, macchine, altri calcolatori)

5

## Cosa è un sistema operativo? (2)

- Un programma che agisce come intermediario tra l'utente di un calcolatore e l'hardware del calcolatore stesso.
- Obiettivi di un sistema operativo:
  - Eseguire programmi utente e rendere più facile la soluzione dei problemi dell'utente
  - Rendere il sistema di calcolo più facile da utilizzare
  - Utilizzare l'hardware del calcolatore in modo efficiente

Questi obiettivi sono in contrapposizione. A quale obiettivo dare priorità dipende dal contesto.

6

## Alcune definizioni di Sistema Operativo

- Macchina astratta  
Implementa funzionalità di alto livello, nascondendo dettagli di basso livello.
- Allocatore di risorse  
Gestisce ed alloca le risorse finite della macchina.
- Programma di controllo  
Controlla l'esecuzione dei programmi e le operazioni sui dispositivi di I/O.  
Condivisione rispetto al tempo e rispetto allo spazio

7

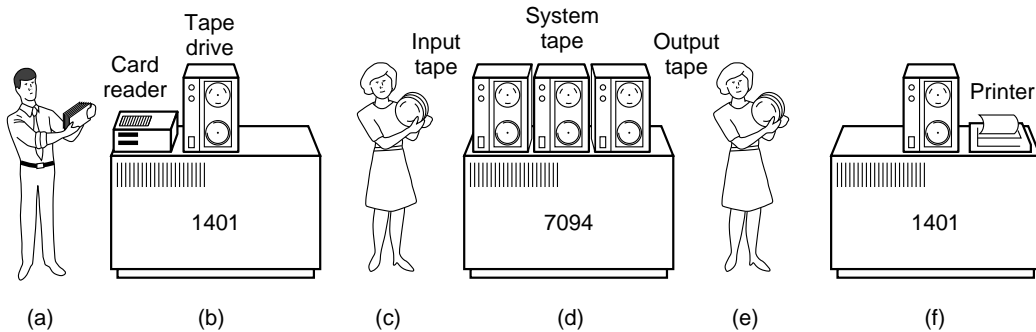
## Primi sistemi – Macchine nude e crude (primi anni '50)

- Struttura
  - Grossi calcolatori funzionanti solo da console
  - Sistemi single user; il programmatore era anche utente e operatore
  - I/O su nastro perforato o schede perforate
- Primi Software
  - Assemblatori, compilatori, linker, loader
  - Librerie di subroutine comuni
  - Driver di dispositivi
- Uso inefficiente di risorse assai costose
  - Bassa utilizzazione della CPU
  - Molto tempo impiegato nel setup dei programmi

8

## Semplici Sistemi Batch

- Utente  $\neq$  operatore
- Aggiungere un lettore di schede



9

- Ridurre il tempo di setup raggruppando i job simili (*batch*)
- Sequenzializzazione automatica dei job – automaticamente, il controllo passa da un job al successivo. Primo rudimentale sistema operativo
- Monitor residente
  - inizialmente, il controllo è in monitor
  - poi viene trasferito al job
  - quando il job è completato, il controllo torna al monitor

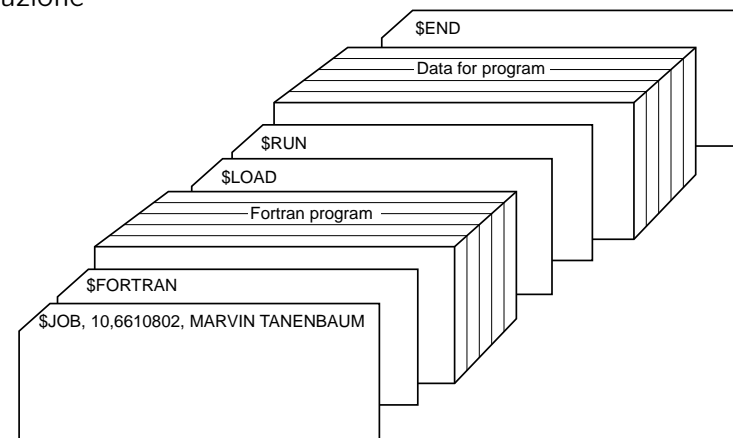
## Semplici Sistemi Batch (Cont.)

- Problemi
  1. Come fa il monitor a sapere la natura del job (e.g., Fortran o assembler?) o quale programma eseguire sui dati forniti?
  2. Come fa il monitor a distinguere
    - (a) un job da un altro
    - (b) dati dal programma
- Soluzione: schede di controllo

10

## Schede di controllo

- Schede speciali che indicano al monitor residente quali programmi mandare in esecuzione



- Caratteri speciali distinguono le schede di controllo dalle schede di programma o di dati.

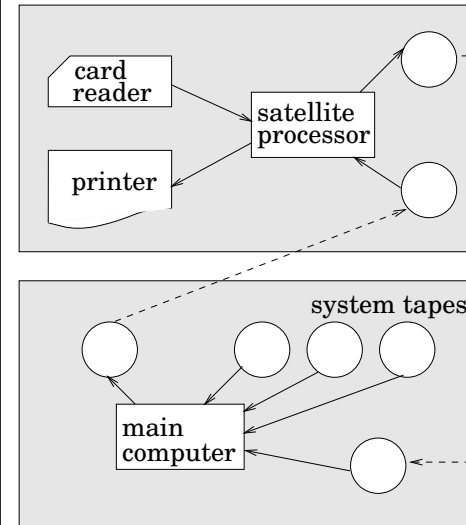
11

## Schede di controllo (Cont.)

- Una parte del monitor residente è
  - Inteprete delle schede di controllo – responsabile della lettura e esecuzione delle istruzioni sulle schede di controllo
  - Loader – carica i programmi di sistema e applicativi in memoria
  - Driver dei dispositivi – conoscono le caratteristiche e le proprietà di ogni dispositivo di I/O.
- Problema: bassa performance – I/O e CPU non possono sovrapporsi; i lettori di schede sono molto lenti.
- Soluzione: operazioni off-line – velocizzare la computazione caricando i job in memoria da nastri, mentre la lettura e la stampa vengono eseguiti off-line

12

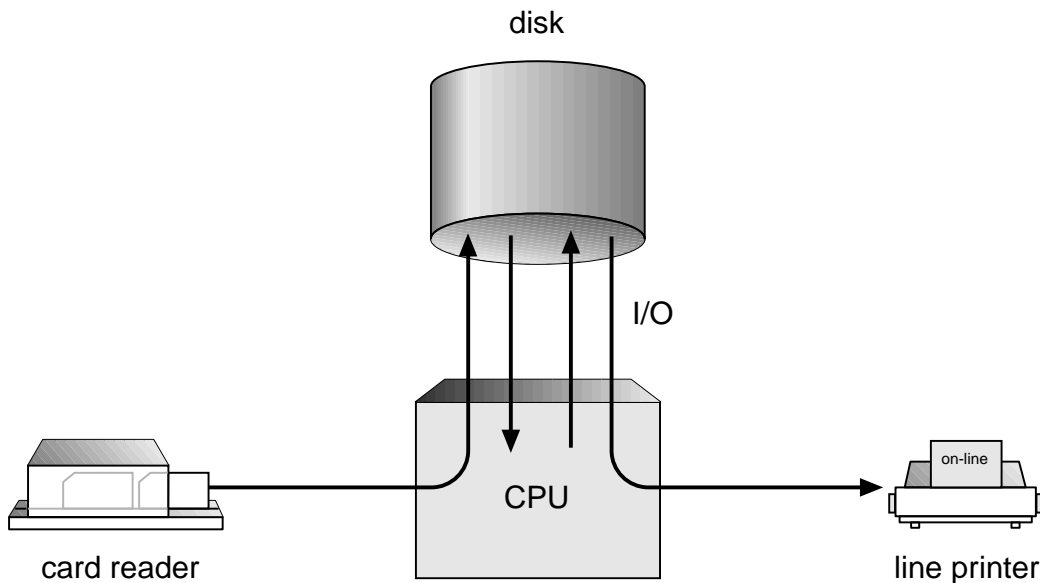
## Funzionamento Off-Line



- Il computer principale non è limitato dalla velocità dei lettori di schede o stampanti, ma solo dalla velocità delle unità nastro.
- Non si devono fare modifiche nei programmi applicativi per passare dal funzionamento diretto a quello off-line
- Guadagno in efficienza: si può usare più lettori e più stampanti per una CPU.

13

## Spooling

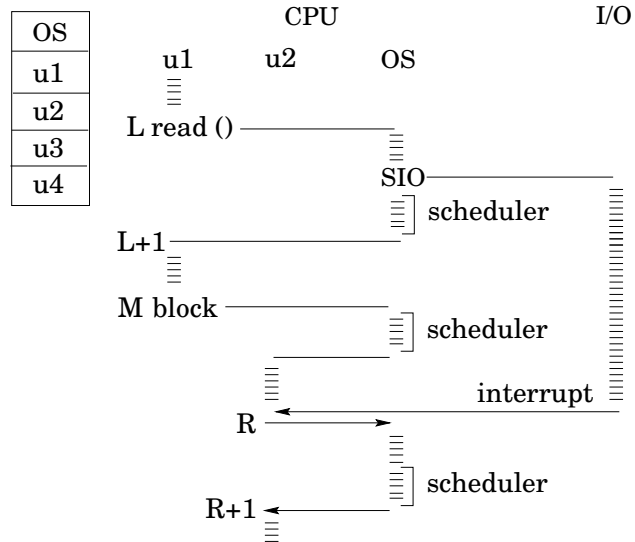


14

- Spool = Simultaneous peripheral operation on-line
- Sovrapposizione dell'I/O di un job con la computazione di un altro job. Mentre un job è in esecuzione, il sistema operativo
  - legge il prossimo job dal lettore di schede in un'area su disco (coda dei job)
  - trasferisce l'output del job precedente dal disco alla stampante
- *Job pool* – struttura dati che permette al S.O. di scegliere quale job mandare in esecuzione al fine di aumentare l'utilizzazione della CPU.

## Anni 60: Sistemi batch Multiprogrammati

Più job sono tenuti in memoria nello stesso momento, e la CPU fa a turno su tutti i job



15

## Caratteristiche dell'OS richieste per la multiprogrammazione

- routine di I/O devono essere fornite dal sistema
- Gestione della Memoria – il sistema deve allocare memoria per più job
- Scheduling della CPU – il sistema deve scegliere tra più job pronti per l'esecuzione
- Allocazione dei dispositivi

16

## Anni 70: Sistemi Time-Sharing – Computazione Interattiva

- La CPU è condivisa tra più job che sono tenuti in memoria e su disco (la CPU è allocata ad un job solo se questo si trova in memoria)
- Un job viene caricato dal disco alla memoria, e viceversa (*swapping*)
- Viene fornita una comunicazione on-line tra l'utente e il sistema; quando il sistema operativo termina l'esecuzione di un comando, attende il prossimo "statement di controllo" non dal lettore di schede bensì dalla tastiera dell'utente.
- Deve essere disponibile un file system on-line per poter accedere ai dati e al codice

17

## Anni 80: Personal Computer

- *Personal computers* – sistemi di calcolo dedicati ad un singolo utente
- I/O devices – tastiere, mouse, schermi, piccole stampanti
- Comodità per l'utente e reattività
- Interfaccia utente evoluta (GUI)
- Spesso gli individui hanno un uso esclusivo del calcolatore, e non necessitano di avanzate tecniche di sfruttamento della CPU o sistemi di protezione.

18

## Anni 90: Sistemi operativi di rete

- Distribuzione della computazione tra più processori
- Sistemi *debolmente accoppiati* – ogni processore ha la sua propria memoria; i processori comunicano tra loro attraverso linee di comunicazione (e.g., bus ad alta velocità, linee telefoniche, fibre ottiche, . . . )
- In un sistema operativi di rete, l'utente ha coscienza della differenza tra i singoli nodi.
  - Trasferimenti di dati e computazioni avvengono in modo esplicito
  - Poco tollerante ai guasti
  - Complesso per gli utenti

19

## Il futuro: Sistemi operativi distribuiti

- In un sistema operativo distribuito, l'utente ha una visione *unitaria* del sistema di calcolo.
  - Condivisione delle risorse (dati e computazionali)
  - Aumento della velocità – bilanciamento del carico
  - Tolleranza ai guasti
- Un sistema operativo distribuito è molto più complesso di un SO di rete.
- Esempi di servizi (non sistemi) di rete: NFS, P2P (KaZaA, Gnutella, . . . ), Grid computing. . .

20

## Riepilogo

- I generazione ('45-'55): relè/valvole, no sistema operativo
- II generazione ('55-'65): transistor e schede perforate
  - sistemi batch: IBM 1401 (scheda ⇔ nastro) e IBM 7094 (calcolo)
- III generazione ('65-'80): circuiti integrati
  - compatibilità tra macchine IBM diverse (360,370, . . . )
  - OS/360 con spooling e multiprogrammazione
  - MULTICS: servizio centralizzato e time-sharing
  - PDP-1 . . . -11: minicalcolatori a 18bit
  - UNIX: Versione singolo utente di MULTICS per PDP-7

21

- IV Generazione ('80-oggi): circuiti integrati su larga scala
  - Personal Computer IBM e MS-DOS
  - MacIntosh di Apple con GUI (Graphical User Interface)
  - Sistema operativo Windows
    - \* Windows costruito su DOS
    - \* Windows 95 e Windows 98 (ancora con codice assembly a 16bit)
    - \* Windows NT e Windows 2000 (a 32bit)
    - \* Windows Me (update di Windows 98)
    - \* Windows XP
  - Linux versione open source di Unix

## Tipologie di Sistemi Operativi

Diversi obiettivi e requisiti a seconda delle situazioni

- Supercalcolatori
- Mainframe
- Server
- Multiprocessore
- Personal Computer
- Real Time
- Embedded

22

## Sistemi operativi per mainframe

- Grandi quantità di dati ( $> 1TB \simeq 10^{12}B$ )
- Grande I/O
- Elaborazione "batch" non interattiva
- Assoluta stabilità (uptime  $> 99,999\%$ )
- Applicazioni: banche, amministrazioni, ricerca...
- Esempi: IBM OS/360, OS/390

23

## Sistemi operativi per supercalcolatori

- Grandi quantità di dati ( $> 1TB$ )
- Enormi potenze di calcolo (es. NEC Earth-Simulator, 40 TFLOP)
- Architetture con migliaia di CPU
- Elaborazione "batch" non interattiva
- Esempi: Unix, o ad hoc

24

## Sistemi per server

- Sistemi multiprocessore con spesso più di una CPU in comunicazione stretta.
- Rilevamento automatico dei guasti
- Elaborazione su richiesta (semi-interattiva)
- Applicazioni: server web, di posta, dati, etc.
- Esempi: Unix, Linux, Windows NT e derivati

25

## **Sistemi Real-Time**

- Vincoli temporali fissati e ben definiti
- Sistemi *hard real-time*: i vincoli devono essere soddisfatti (es. fermare il braccio meccanico)
  - la memoria secondaria è limitata o assente; i dati sono memorizzati o in memoria volatile, o in ROM.
  - In conflitto con i sistemi time-sharing; non sono supportati dai sistemi operativi d'uso generale
  - Usati in robotica, controlli industriali, software di bordo. . .
- Sistemi *soft real-time*: i vincoli possono anche non essere soddisfatti, ma il sistema operativo deve fare del suo meglio
  - Uso limitato nei controlli industriali o nella robotica

26

- Utili in applicazioni (multimedia, virtual reality) che richiedono caratteristiche avanzate dei sistemi operativi

## **Sistemi operativi embedded**

- Per calcolatori palmari (PDA), cellulari, ma anche televisori, forni a microonde, lavatrici, etc.
- Hanno spesso caratteristiche di real-time
- Limitate risorse hardware
- esempio: PalmOS, Epoc, PocketPC, QNX.

27

## **Sistemi operativi per smart card**

- Girano sulla CPU delle smartcard
- Stretti vincoli sull'uso di memoria e alimentazione
- implementano funzioni minime
- Esempio: JavaCard

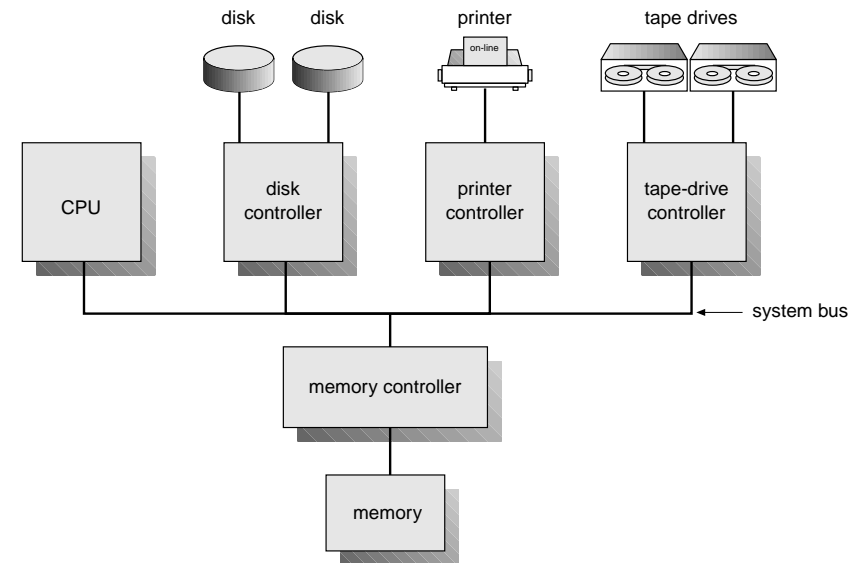
28

## Struttura dei Sistemi di Calcolo

- Operazioni dei sistemi di calcolo
- Struttura dell'I/O
- Struttura della memoria
- Gerarchia delle memorie
- Protezione hardware
- Invocazione del Sistema Operativo

29

## Architettura dei calcolatori



30

## Operazioni dei sistemi di calcolo

- I dispositivi di I/O e la CPU possono funzionare concorrentemente
- Ogni controller gestisce un particolare tipo di dispositivo
- Ogni controller ha un buffer locale
- La CPU muove dati da/per la memoria principale per/da i buffer locali dei controller
- L'I/O avviene tra il dispositivo e il buffer locale del controller
- Il controller informa la CPU quando ha terminato la sua operazione, generando un *interrupt*.

31

## Funzionamento event-driven di un S.O.

- Il programma di *bootstrap* inizializza i registri della CPU e dei controller dei dispositivi di I/O
- Quindi carica in memoria il nucleo del sistema operativo e lo esegue
- Il sistema operativo lancia un processo speciale (init in Unix) e poi attende segnali di interrupt (eventi che possono cambiare il corso dell'esecuzione del programma corrente)

32

## Interrupt

- Visono due tipi di interrupt
  - Segnali di interrupt che arrivano da un controller
  - Interrupt generate da software, chiamate *trap*: un trap può essere causato o da un errore o da una esplicita richiesta dell'utente, ad. es., una chiamata di una funzione di sistema (*system call*)

33

## Funzioni comuni degli Interrupt

- Gli interrupt trasferiscono il controllo alla routine di servizio dell'interrupt, generalmente attraverso il *vettore di interruzioni*, che contiene gli indirizzi di tutte le routine di servizio.
- L'hardware deve salvare l'indirizzo dell'istruzione interrotta.
- Interrupt in arrivo sono *disabilitati* mentre un altro interrupt viene gestito, per evitare che vadano perduti.

34

## Funzioni comuni degli Interrupt

- Gli interrupt trasferiscono il controllo alla routine di servizio dell'interrupt, generalmente attraverso il *vettore di interruzioni*, che contiene gli indirizzi di tutte le routine di servizio.
- L'hardware deve salvare l'indirizzo dell'istruzione interrotta.
- Interrupt in arrivo sono *disabilitati* mentre un altro interrupt viene gestito, per evitare che vadano perduti.
- Un *trap* è un interrupt generato da software, causato o da un errore o da una esplicita richiesta dell'utente.
- Un sistema operativo è *guidato da interrupt*

35

## Gestione degli Interrupt

- Il sistema operativo preserva lo stato della CPU salvando registri e program counter.
- Determinazione di quale tipo di interrupt è avvenuto:
  - Se i controller non inviano direttamente segnali di interrupt alla CPU, si utilizza il *polling*: ciclicamente la CPU controlla lo stato (registri) dei dispositivi per vedere se si sono verificati eventi rilevanti (ad esempio fine di un trasferimento I/O) e quindi invoca la routine per gestire tale evento (ad es. trasferimento dei dati da buffer a memoria)
  - Se il controller invia un segnale di interrupt (con informazioni sul tipo di interrupt, ad es. un indice per il vettore delle interr.) il sistema operativo utilizza il selezionare la routine di gestione attraverso il vettore delle interruzioni

36

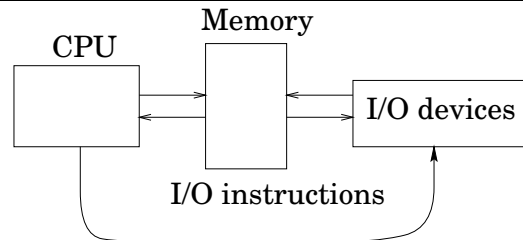
## Struttura dell'I/O

- I/O sincrono: dopo che l'I/O è partito, il controllo ritorna al programma utente solo dopo che l'I/O è stato completato
  - l'istruzione hardware **wait** (se esiste) blocca la CPU fino alla prossima interruzione
  - oppure, la CPU aspetta la prossima interruzione tramite un ciclo di attesa (chiamato *busy wait*)
  - in questo caso al più una richiesta di I/O è eseguita alla volta; non ci sono I/O paralleli

37

- I/O asincrono: dopo che l'I/O è partito, il controllo ritorna al programma utente senza aspettare che l'I/O venga completato
  - *chiamata di sistema (System call)* – richiede al sistema operativo di sospendere il processo in attesa del completamento dell'I/O.
  - Se non ci sono processi da eseguire la CPU esegue un'istruzione **wait**
  - una *tabella dei dispositivi* mantiene tipo, indirizzo e stato di ogni dispositivo di I/O.
  - Il sistema operativo accede alla tabella dei dispositivi per determinare lo stato, e per mantenere le informazioni relative agli interrupt.

## Struttura del Direct Memory Access (DMA)



- Usata per dispositivi in grado di trasferire dati a velocità prossime a quelle della memoria
- I controller trasferiscono blocchi di dati dal buffer locale direttamente alla memoria, senza intervento della CPU (per evitare di dover gestire troppe interruzioni)
- Viene generato un solo interrupt per blocco, invece di uno per ogni byte trasferito.

38

## Struttura della Memoria

- Memoria principale (RAM) – la memoria che la CPU può accedere direttamente.
- Memoria secondaria (Dischi, floppy, CD, ...) – estensione della memoria principale che fornisce una memoria non volatile (e solitamente più grande)

39

## Gerarchia della Memoria

I sistemi di memorizzazione sono organizzati gerarchicamente, secondo

- velocità
- costo
- volatilità

*Caching* – duplicare i dati più frequentemente usati di una memoria, in una memoria più veloce. La memoria principale può essere vista come una cache per la memoria secondaria.

40

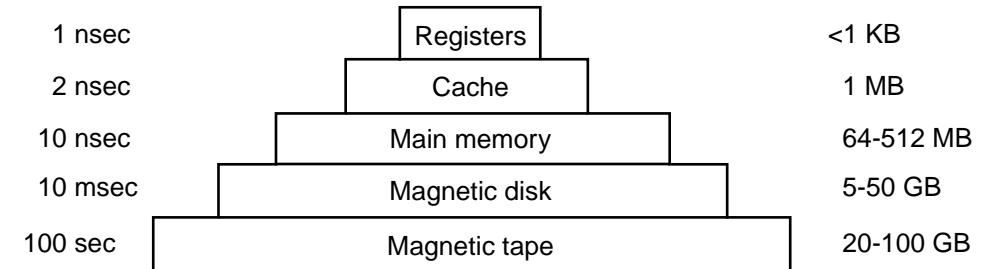
## Protezione hardware

- Funzionamento in dual-mode
- Protezione dell'I/O
- Protezione della Memoria
- Protezione della CPU

41

Typical access time

Typical capacity



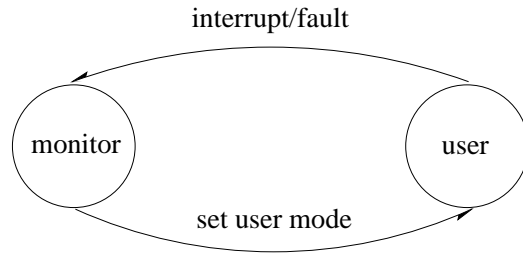
## Funzionamento Dual-Mode

- La condivisione di risorse di sistema richiede che il sistema operativo assicuri che un programma scorretto non possa portare altri programmi (corretti) a funzionare non correttamente.
- L'hardware deve fornire un supporto per differenziare almeno tra due modi di funzionamento
  1. *User mode* – la CPU sta eseguendo codice di un utente
  2. *Monitor mode* (anche *supervisor mode*, *system mode*, *kernel mode*) – la CPU sta eseguendo codice del sistema operativo

42

## Funzionamento Dual-Mode (Cont.)

- La CPU ha un *Mode bit* che indica in quale modo si trova: supervisor (0) o user (1).
- Quando avviene un interrupt, l'hardware passa automaticamente in modo supervisore



- Le *istruzioni privilegiate* possono essere eseguite solamente in modo supervisore

43

## Protezione dell'I/O

- Tutte le istruzioni di I/O sono privilegiate
- Si deve assicurare che un programma utente non possa mai passare in modo supervisore (per esempio, andando a scrivere nel vettore delle interruzioni)

44

## Protezione della Memoria

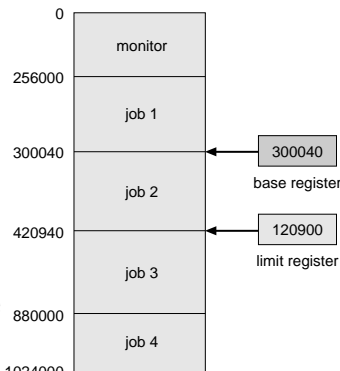
Si deve proteggere almeno il vettore delle interruzioni e le routine di gestione degli interrupt

- Per avere la protezione della memoria, si aggiungono due registri che determinano il range di indirizzi a cui un programma può accedere:

**registro base** contiene il primo indirizzo fisico legale

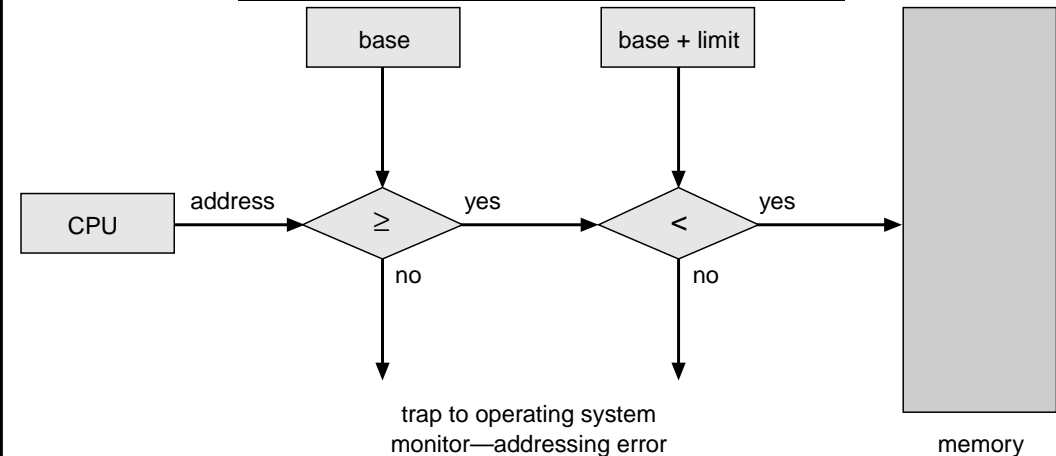
**registro limite** contiene la dimensione del range di memoria accessibile

- la memoria al di fuori di questo range è protetta



45

## Protezione della Memoria (Cont.)



- Essendo eseguito in modo monitor, il sistema operativo ha libero accesso a tutta la memoria, sia di sistema sia utente

- Le istruzioni di caricamento dei registri base e limite sono privilegiate

46

## Protezione della CPU

- il *Timer* interrompe la computazione dopo periodi prefissati, per assicurare che periodicamente il sistema operativo riprenda il controllo
  - Il timer viene decrementato ad ogni *tick* del clock (1/50 di secondo, tipicamente)
  - Quanto il timer va a 0, avviene l'interrupt
- Il timer viene usato comunemente per implementare il time sharing
- Serve anche per mantenere la data e l'ora
- Il caricamento del timer è una istruzione privilegiata

47

## Invocazione del sistema operativo

- Dato che le istruzioni di I/O sono privilegiate, come può il programma utente eseguire dell'I/O?
- Attraverso le *system call* – il metodo con cui un processo richiede un'azione da parte del sistema operativo
  - Solitamente sono un interrupt software (**trap**)
  - Il controllo passa attraverso il vettore di interrupt alla routine di servizio della trap nel sistema operativo, e il mode bit viene impostato a "monitor".
  - Il sistema operativo verifica che i parametri siano legali e corretti, esegue la richiesta, e ritorna il controllo all'istruzione che segue la system call.
  - Con l'istruzione di ritorno, il mode bit viene impostato a "user"

48

## Struttura dei Sistemi Operativi

- Componenti del sistema
- Servizi del Sistema Operativo
- Chiamate di sistema (*system calls*)
- Programmi di Sistema
- Struttura del Sistema
- Macchine Virtuali

49

## Componenti comuni dei sistemi

1. Gestione dei processi
2. Gestione della Memoria Principale
3. Gestione della Memoria Secondaria
4. Gestione dell'I/O
5. Gestione dei file
6. Sistemi di protezione
7. Connessioni di rete (*networking*)
8. Sistema di interpretazione dei comandi

50

## Gestione dei processi

- Un *processo* è un programma in esecuzione. Un processo necessita di certe risorse, tra cui tempo di CPU, memoria, file, dispositivi di I/O, per assolvere il suo compito.
- Il sistema operativo è responsabile delle seguenti attività, relative alla gestione dei processi:
  - creazione e cancellazione dei processi
  - sospensione e riesumazione dei processi
  - fornire meccanismi per
    - \* sincronizzazione dei processi
    - \* comunicazione tra processi
    - \* evitare, prevenire e risolvere i *deadlock*

51

## Gestione della Memoria Principale

- La *memoria principale* è un (grande) array di parole (byte, words. . . ), ognuna identificata da un preciso indirizzo. È un deposito di dati rapidamente accessibili dalla CPU e dai dispositivi di I/O.
- La memoria principale è *volatile*. Perde il suo contenuto in caso di *system failure*.
- Il sistema operativo è responsabile delle seguenti attività relative alla gestione della memoria:
  - Tener traccia di quali parti della memoria sono correntemente utilizzate, e da chi.
  - Decidere quale processo caricare in memoria, quando dello spazio si rende disponibile.
  - Allocare e deallocare spazio in memoria, su richiesta.

52

## Gestione della memoria secondaria

- Dal momento che la memoria principale è volatile e troppo piccola per contenere tutti i dati e programmi permanentemente, il calcolatore deve prevedere anche una *memoria secondaria* di supporto a quella principale.
- La maggior parte dei calcolatori moderni utilizza *dischi* come principale supporto per la memoria secondaria, sia per i programmi che per i dati.
- Il sistema operativo è responsabile delle seguenti attività relative alla gestione dei dischi:
  - Gestione dello spazio libero
  - Allocazione dello spazio
  - Schedulazione dei dischi

53

## Gestione del sistema di I/O

- Il sistema di I/O consiste in
  - un sistema di cache a buffer
  - una interfaccia generale ai gestori dei dispositivi (*device driver*)
  - i driver per ogni specifico dispositivo hardware (controller)

54

## Gestione dei File

- Un *file* è una collezione di informazioni correlate, definite dal suo creatore. Comunemente, i file rappresentano programmi (sia sorgenti che eseguibili (*oggetti*)) e dati.
- Il sistema operativo è responsabile delle seguenti attività connesse alla gestione dei file:
  - Creazione e cancellazione dei file
  - Creazione e cancellazione delle directory
  - Supporto di primitive per la manipolazione di file e directory
  - Allocazione dei file nella memoria secondaria
  - Salvataggio dei dati su supporti non volatili

55

## Sistemi di protezione

- Per *Protezione* si intende un meccanismo per controllare l'accesso da programmi, processi e utenti sia al sistema, sia alle risorse degli utenti.
- Il meccanismo di protezione deve:
  - distinguere tra uso autorizzato e non autorizzato.
  - fornire un modo per specificare i controlli da imporre
  - forzare gli utenti e i processi a sottostare ai controlli richiesti

56

## Networking (Sistemi Distribuiti)

- Un *sistema distribuito* è una collezione di processori che non condividono memoria o clock. Ogni processore ha una memoria propria.
- I processori del sistema sono connessi attraverso una *rete di comunicazione*.
- Un sistema distribuito fornisce agli utenti l'accesso a diverse risorse di sistema.
- L'accesso ad una risorsa condivisa permette:
  - Aumento delle prestazioni computazionali
  - Incremento della quantità di dati disponibili
  - Aumento dell'affidabilità

57

## Interprete dei comandi

- Molti comandi sono dati al sistema operativo attraverso *control statement* che servono per
  - creare e gestire i processi
  - gestione dell'I/O
  - gestione della memoria secondaria
  - gestione della memoria principale
  - accesso al file system
  - protezione
  - networking

58

## Interprete dei comandi (Cont.)

- Il programma che legge e interpreta i comandi di controllo ha diversi nomi:
  - interprete delle schede di controllo (sistemi batch)
  - interprete della linea di comando (DOS, Windows)
  - shell (in UNIX)
  - interfaccia grafica: Finder in MacOS, Explorer in Windows, gnome-session in Unix. . .

La sua funzione è di ricevere un comando, eseguirlo, e ripetere.

59

## Servizi dei Sistemi Operativi

- Esecuzione dei programmi: caricamento dei programmi in memoria ed esecuzione.
- Operazioni di I/O: il sistema operativo deve fornire un modo per condurre le operazioni di I/O, dato che gli utenti non possono eseguirle direttamente,
- Manipolazione del file system: capacità di creare, cancellare, leggere, scrivere file e directory.
- Comunicazioni: scambio di informazioni tra processi in esecuzione sullo stesso computer o su sistemi diversi collegati da una rete. Implementati attraverso *memoria condivisa* o *passaggio di messaggi*.
- Individuazione di errori: garantire una computazione corretta individuando errori nell'hardware della CPU o della memoria, nei dispositivi di I/O, o nei programmi degli utenti.

60

## Funzionalità aggiuntive dei sistemi operativi

Le funzionalità aggiuntive esistono per assicurare l'efficienza del sistema, piuttosto che per aiutare l'utente

- Allocazione delle risorse: allocare risorse a più utenti o processi, allo stesso momento
- Accounting: tener traccia di chi usa cosa, a scopi statistici o di rendicontazione
- Protezione: assicurare che tutti gli accessi alle risorse di sistema siano controllate

61

## Chiamate di Sistema (System Calls)

- Le chiamate di sistema formano l'interfaccia tra un programma in esecuzione e il sistema operativo.
- Generalmente, sono disponibili come speciali istruzioni assembler
- Alcuni linguaggi pensati per programmazione di sistema permettono di eseguire direttamente system call (e.g., Bliss e PL 360 (anni 70), C).
- In questo il Run Time System (RTS) (insieme di funzioni predefinite associate ad una particolare piattaforma fornite con il compilatore) fornisce l'interfaccia per la chiamata a basso livello
- Tre metodi generali per passare parametri tra il programma e il sistema operativo:

62

- Passare i parametri nei *registri*.
- Memorizzare i parametri in una tabella in memoria, il cui indirizzo è passato come parametro in un registro
- Il programma mette i parametri sullo *stack*, da cui il sistema operativo li recupera.

## Tipi di chiamate di sistema

- Controllo dei processi:** creazione/terminazione processi, esecuzione programmi, (de)allocazione memoria, attesa di eventi, impostazione degli attributi,...
- Gestione dei file:** creazione/cancellazione, apertura/chiusura, lettura/scrittura, impostazione degli attributi,...
- Gestione dei dispositivi:** allocazione/rilascio dispositivi, lettura/scrittura, collegamento logico dei dispositivi (e.g. mounting)...
- Informazioni di sistema:** leggere/scrivere data e ora del sistema, informazioni sull'hardware/software installato,...
- Comunicazioni:** creare/cancellare connessioni, spedire/ricevere messaggi,...

63

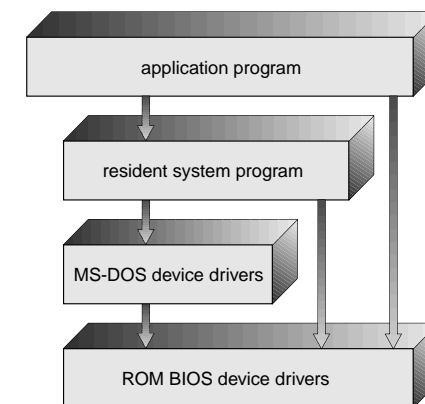
## Programmi di sistema

- I programmi di sistema forniscono un ambiente per lo sviluppo e l'esecuzione dei programmi. Si dividono in
  - Gestione dei file
  - Modifiche dei file
  - Informazioni sullo stato del sistema e dell'utente
  - Supporto dei linguaggi di programmazione
  - Caricamento ed esecuzione dei programmi
  - Comunicazioni
  - Programmi applicativi
- La maggior parte di ciò che un utente vede di un sistema operativo è definito dai programmi di sistema, non dalle reali chiamate di sistema.

64

## Struttura dei Sistemi Operativi -Approccio semplice

- MS-DOS – pensato per fornire le massime funzionalità nel minore spazio possibile.
  - non è diviso in moduli (è cresciuto oltre il previsto)
  - nonostante ci sia un po' di struttura, le sue interfacce e livelli funzionali non sono ben separati.



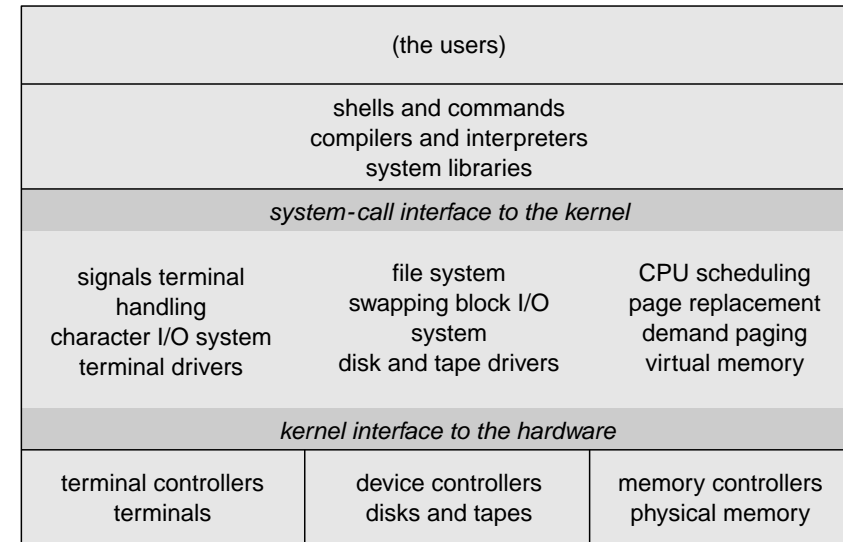
65

## Struttura dei Sistemi Operativi - Approccio semplice

- UNIX – limitato dalle funzionalità hardware, lo UNIX originale aveva una debole strutturazione. Consiste almeno in due parti ben separate:
  - Programmi di sistema
  - Il kernel
    - \* consiste in tutto ciò che sta tra le system call e l'hardware
    - \* implementa il file system, lo scheduling della CPU, gestione della memoria e altre funzioni del sistema operativo: molte funzionalità in un solo livello.

66

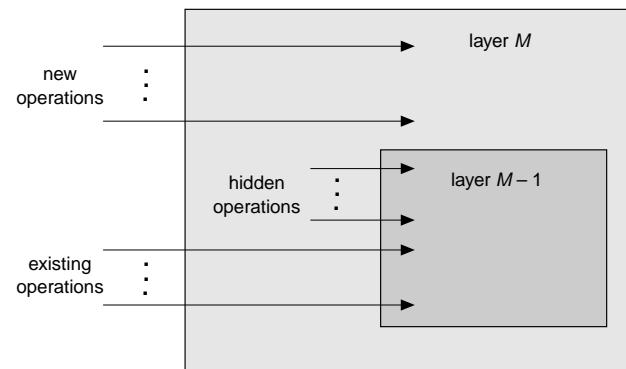
## Struttura dei Sistemi Operativi – Unix originale



67

## Struttura dei sistemi operativi – Approccio stratificato

- Il sistema operativo è diviso in un certo numero di strati (livelli); ogni strato è costruito su quelli inferiori. Lo strato di base (livello 0) è l'hardware; il più alto è l'interfaccia utente.
- Secondo la modularità, gli strati sono pensati in modo tale che ognuno utilizza funzionalità (operazioni) e servizi solamente di strati inferiori.



68

## Struttura dei sistemi operativi – Stratificazione di THE

- La prima stratificazione fu usata nel sistema operativo THE per un calcolatore olandese nel 1969 da Dijkstra e dai suoi studenti.
- THE consisteva dei seguenti sei strati:

layer 5:	user programs
layer 4:	buffering for input and output devices
layer 3:	operator-console device driver
layer 2:	memory management
layer 1:	CPU scheduling
layer 0:	hardware

69

## Stratificazione

- Il sistema MULTICS era organizzato ad anelli concentrici (livelli)
- Per accedere ad un livello piú interno occorre una chiamata di sistema che attivava una TRAP
- L'organizzazione ad anelli si poteva estendere anche a sottosistemi utente (studente lavora a livello  $n + 1$ , programma di correzioni lavora a livello  $n$  per evitare interferenze)

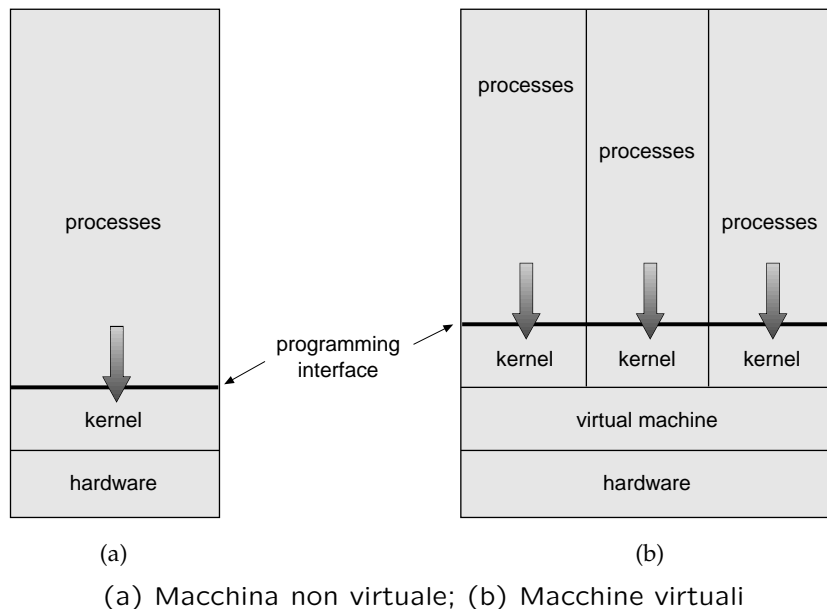
70

## Macchine Virtuali

- Una *macchina virtuale* porta l'approccio stratificato all'estremo: tratta hardware e il sistema operativo come se fosse tutto hardware.
- Una macchina virtuale fornisce una interfaccia *identica* all'hardware nudo e crudo sottostante.
- Il sistema operativo impiega le risorse del calcolatore fisico per creare le macchine virtuali:
  - Lo scheduling della CPU crea l'illusione che ogni processo abbia il suo processore dedicato.
  - La gestione della memoria crea l'illusione di una memoria virtuale per ogni processo
  - Lo spooling può implementare delle stampanti virtuali
  - Spazio disco può essere impiegato per creare "dischi virtuali"

71

## Macchine Virtuali (Cont.)



72

## Vantaggi/Svantaggi delle Macchine Virtuali

- Il concetto di macchina virtuale fornisce una protezione completa delle risorse di sistema, dal momento che ogni macchina virtuale è isolata dalle altre. Questo isolamento non permette però una condivisione diretta delle risorse.
- Un sistema a macchine virtuali è un mezzo perfetto per l'emulazione di altri sistemi operativi, o lo sviluppo di nuovi sistemi operativi: tutto si svolge sulla macchina virtuale, invece che su quella fisica, quindi non c'è pericolo di far danni.
- Implementare una macchina virtuale è complesso, in quanto si deve fornire un *perfetto* duplicato della macchina sottostante. Può essere necessario dover emulare ogni singola istruzione macchina.
- Approccio seguito in molti sistemi: Windows, Linux, MacOS, JVM,...

73

## Exokernel

- Estensione dell'idea di macchina virtuale
- Ogni macchina virtuale di livello utente vede solo un *sottoinsieme* delle risorse dell'intera macchina
- Ogni macchina virtuale può eseguire il proprio sistema operativo
- Le risorse vengono richieste all'exokernel, che tiene traccia di quali risorse sono usate da chi
- Semplifica l'uso delle risorse allocate: l'exokernel deve solo tenere separati i domini di allocazione delle risorse

74

## Meccanismi e Politiche

- I kernel tradizionali (monolitici) sono poco flessibili
  - Distinguere tra *meccanismi* e *politiche*:
    - i meccanismi determinano *come* fare qualcosa;
    - le politiche determinano *cosa* deve essere fatto.
- Ad esempio: assegnare l'esecuzione ad un processo è un meccanismo; scegliere *quale* processo attivare è una politica.
- Questa separazione è un principio molto importante: permette la massima flessibilità, nel caso in cui le politiche debbano essere cambiate.
  - Estremizzazione: il kernel fornisce solo i meccanismi, mentre le politiche vengono implementate in user space.

75

## Sistemi con Microkernel

- *Microkernel*: il kernel è ridotto all'osso, fornisce soltanto i meccanismi:
  - Un meccanismo di comunicazione tra processi
  - Una minima gestione della memoria e dei processi
  - Gestione dell'hardware di basso livello (driver)
- Tutto il resto viene gestito da processi in spazio utente: ad esempio, tutte le politiche di gestione del file system, dello scheduling, della memoria sono implementate come processi.
- Meno efficiente del kernel monolitico
- Grande flessibilità; immediata scalabilità in ambiente di rete
- Sistemi operativi recenti sono basati, in diverse misure, su microkernel (AIX4, BeOS, GNU HURD, MacOS X, QNX, Tru64, Windows NT . . .)

76