

Corso di Sistemi Operativi I

Laurea in Informatica

a.a. 2003/04

1

Docenti

- Teoria:
 - Giorgio Delzanno
 - ufficio 104
 - tel. 6638
 - e-mail: giorgio@disi.unige.it
- Laboratorio:
 - Stefano Bencetti
 - e-mail: bencetti@disi.unige.it

2

Informazioni

- Orario (da Ottobre-a Dicembre)
 - Lunedì 11-13 (?)
 - Martedì 14-16
 - Venerdì 9-11
- 56 ore di teoria
 - Introduzione ai Sistemi Operativi
 - Gestione processi, memoria, file system, device
- 16 di laboratorio
 - Gestione di una macchina come amministratore di sistema
 - Installazione e configurazione di hardware, applicazioni e servizi

3

Esame

- Il corso vale 9 crediti
- Scritto, probabilmente spezzato in compitini, con esercizi e domande di teoria
- Orale per la parte di laboratorio e per la verifica dello scritto

4

Testi, Appunti e Info sul corso

- Riferimenti
 - I moderni sistemi operativi - Seconda ed. - Andrew S. Tanenbaum - Jackson libri Università
- Letture integrative
 - Sistemi Operativi: Concetti ed esempi - Sesta edizione - Silberschatz, Galvin, Gagne - Addison Wesley
 - UNIX: Architettura di sistema - Maurice J. Bach - Jackson
- Pagina web:
<http://www.disi.unige.it/person/DelzannoG/SO1>

5

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.

6

Introduzione

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

7

Cosa è un sistema operativo?

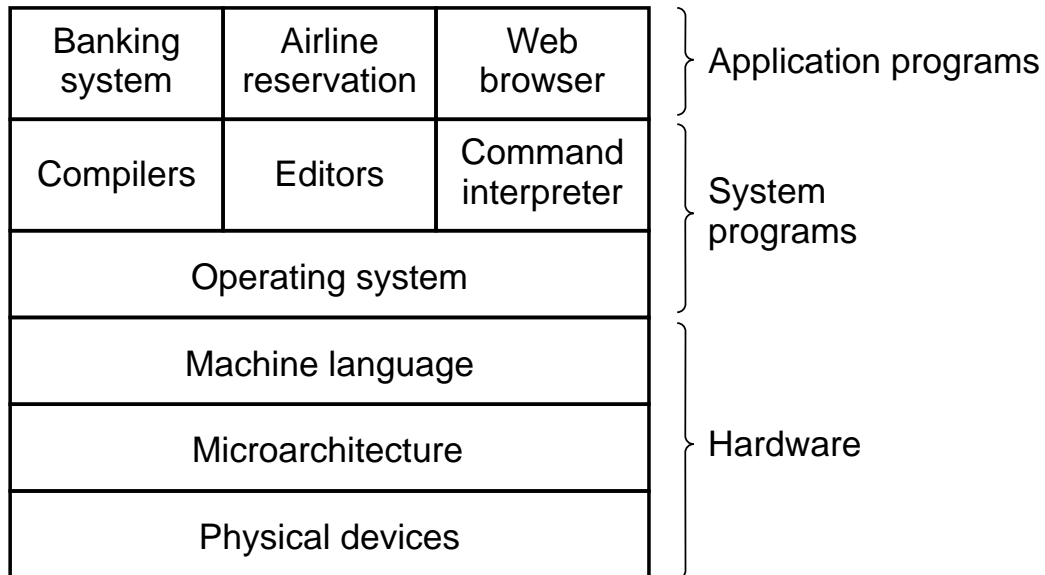
Possibili risposte:

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

Nessuna di queste definizioni è completa

8

Visione astratta delle componenti di un sistema di calcolo



9

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)

10

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.

11

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

12

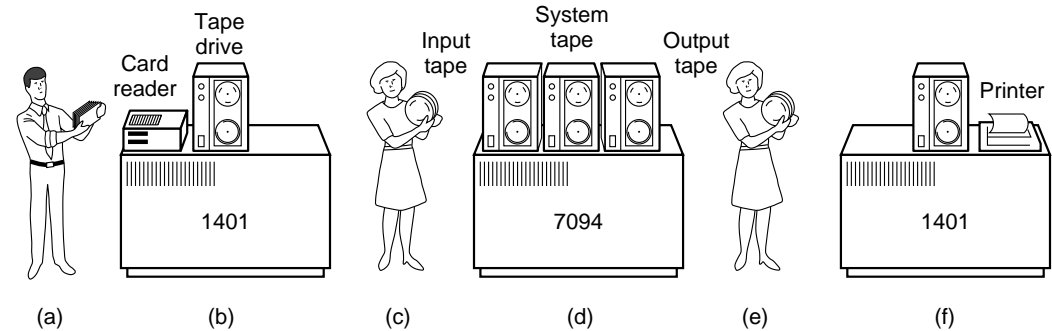
Primi sistemi – Macchine nude e crude (primi anni '50)

- 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

13

Semplici Sistemi Batch

- Utente \neq operatore
- Aggiungere un lettore di schede



14

- 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

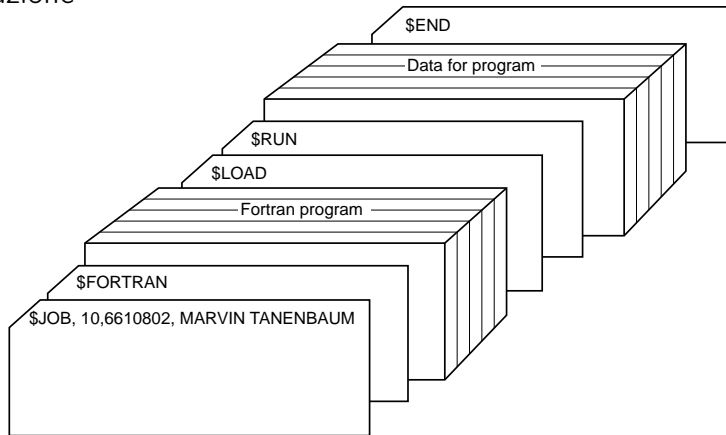
Semplici Sistemi Batch (Cont.)

- 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

15

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.

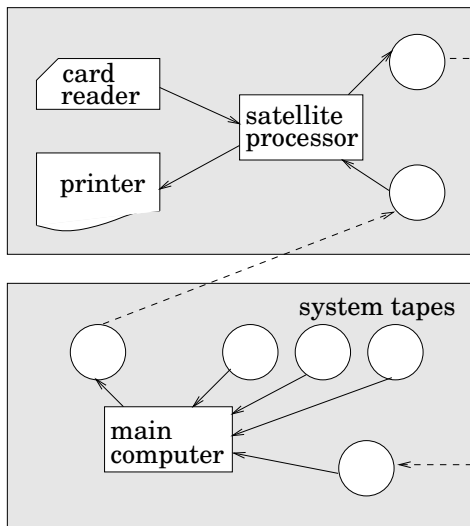
16

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

17

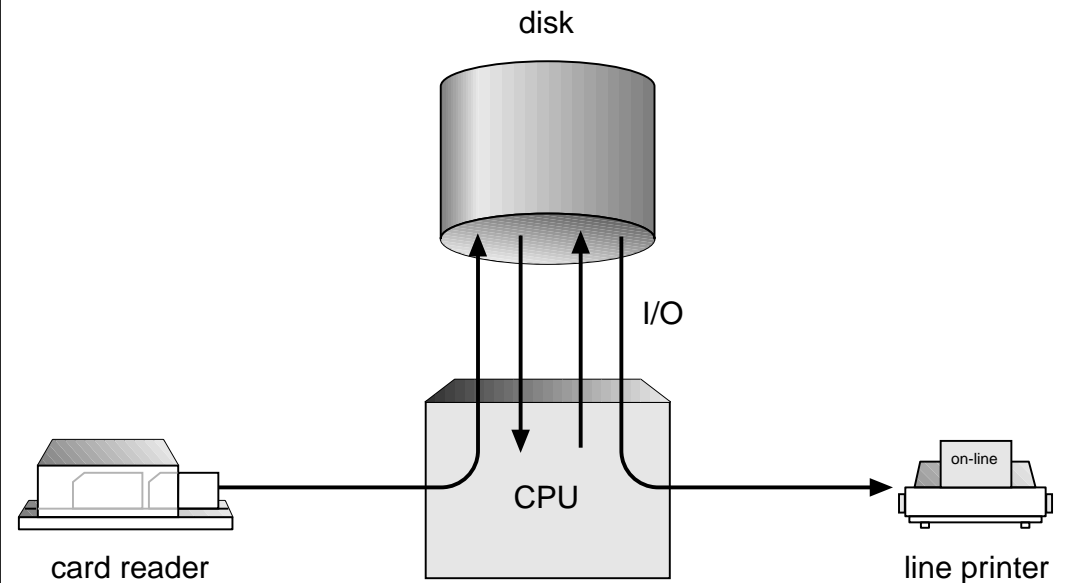
Funzionamento Off-Line



- 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.

18

Spooling

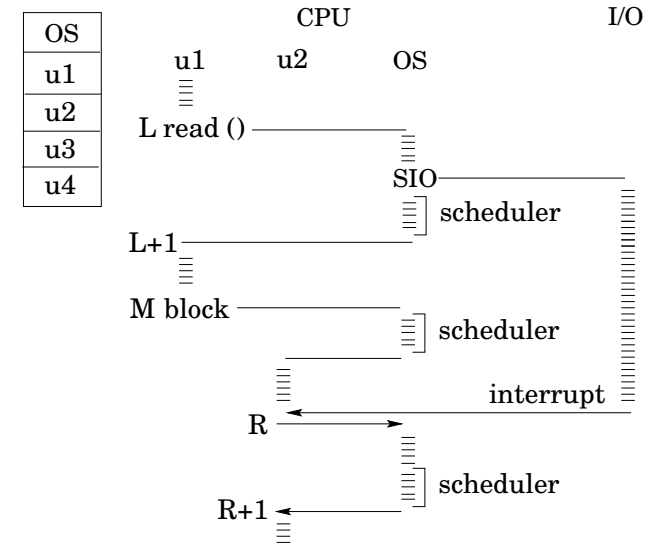


19

- 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



20

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

21

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

22

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.

23

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

24

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. . .

25

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

26

Tipologie di Sistemi Operativi

Diversi obiettivi e requisiti a seconda delle situazioni

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

27

- 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

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

28

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

29

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

30

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

31

- 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.

32

Sistemi operativi per smart card

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

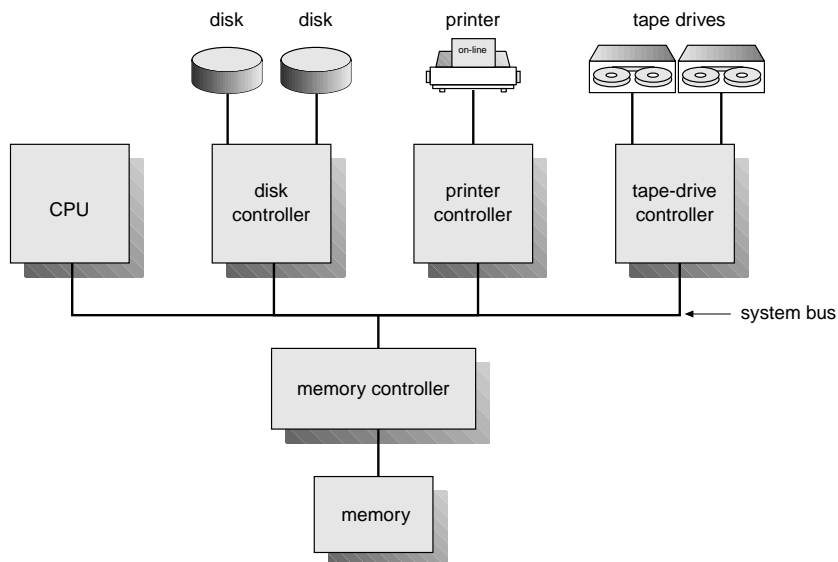
33

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

34

Architettura dei calcolatori



35

Operazioni dei sistemi di calcolo

- I dispositivi di I/O e la CPU possono funzionare concorrentemente
- Ogni controller di dispositivo 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*.

36

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*

37

Gestione degli Interrupt

- Il sistema operativo preserva lo stato della CPU salvando registri e program counter.
- Determinazione di quale tipo di interrupt è avvenuto:
 - *polling*
 - vettore di interrupt
- Per ogni tipo di interrupt, uno specifico segmento di codice determina cosa deve essere fatto.

38

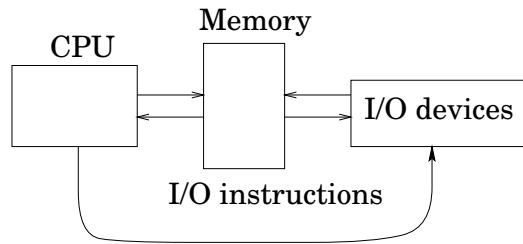
Struttura dell'I/O

- 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 **wait** blocca la CPU fino alla prossima interruzione
 - oppure, un tramite un ciclo di attesa (*busy wait*)
 - al più una richiesta di I/O è eseguita alla volta; non ci sono I/O paralleli

39

- 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.
- Viene generato un solo interrupt per blocco, invece di uno per ogni byte trasferito.

40

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)

41

Gerarchia della Memoria

I sistemi di memorizzazione sono organizzati gerarchicamente, secondo

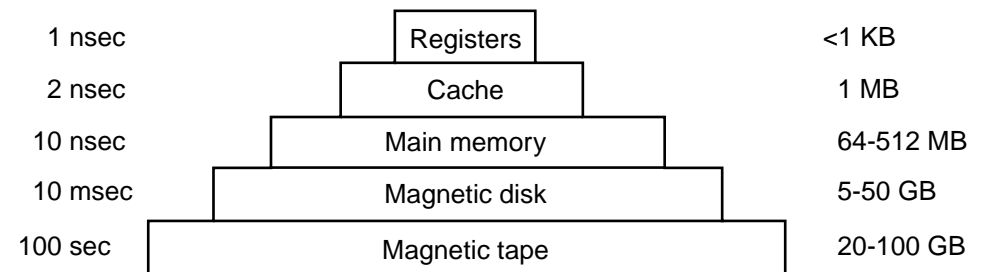
- velocità
- costo
- volatilità

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.

42

Typical access time

Typical capacity



Protezione hardware

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

43

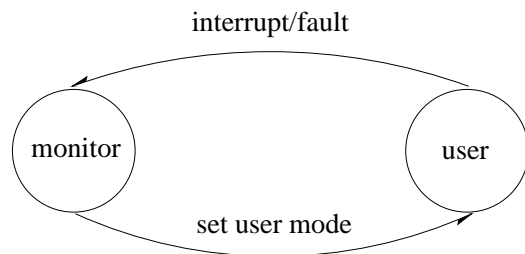
Funzionamento Dual-Mode

- 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

44

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

45

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)

46

Protezione della Memoria

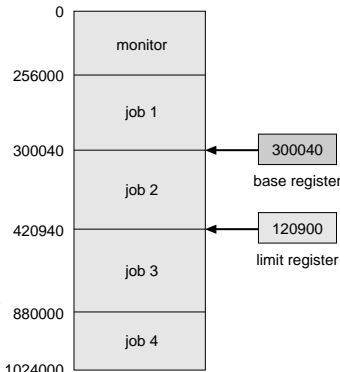
Si deve proteggere almeno il vettore delle interruzioni e le routine di gestione degli interrupt

- 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

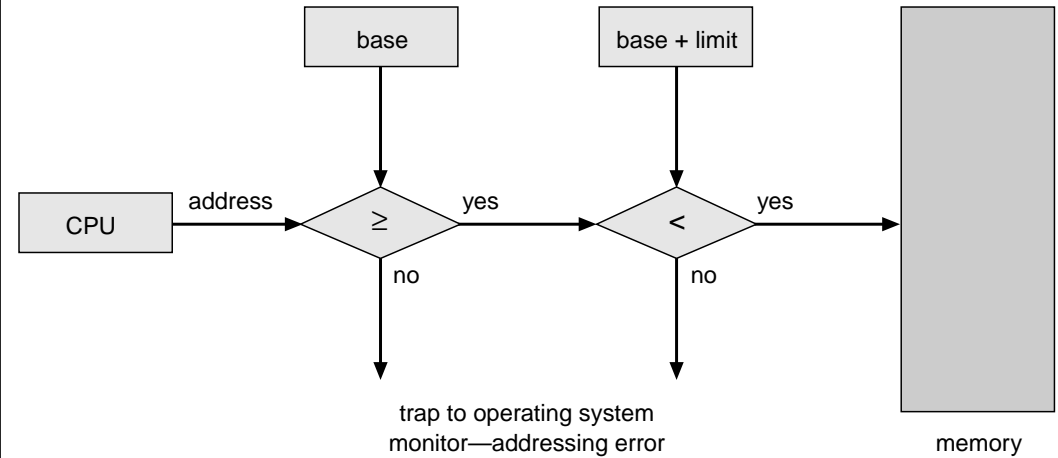
registro limite contiene la dimensione del range di memoria accessibile

- la memoria al di fuori di questo range è protetta



47

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

48

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

49

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"

50

Struttura dei Sistemi Operativi

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

51

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

52

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*

53

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.

54

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

55

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)

56

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

57

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

58

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

59

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

60

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.

61

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.

62

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

63

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
 - Linguaggi pensati per programmazione di sistema permettono di eseguire system call direttamente (e.g., C, Bliss, PL/360).
- Tre metodi generali per passare parametri tra il programma e il sistema operativo:
 - 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.

64

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,...

65

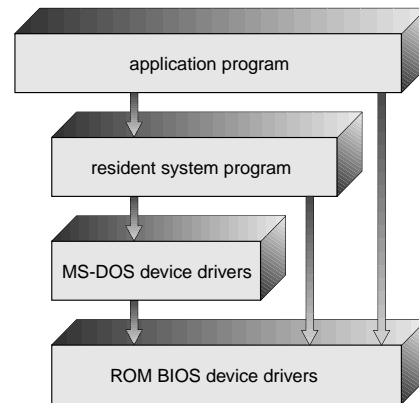
Programmi di sistema

- 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.

66

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.



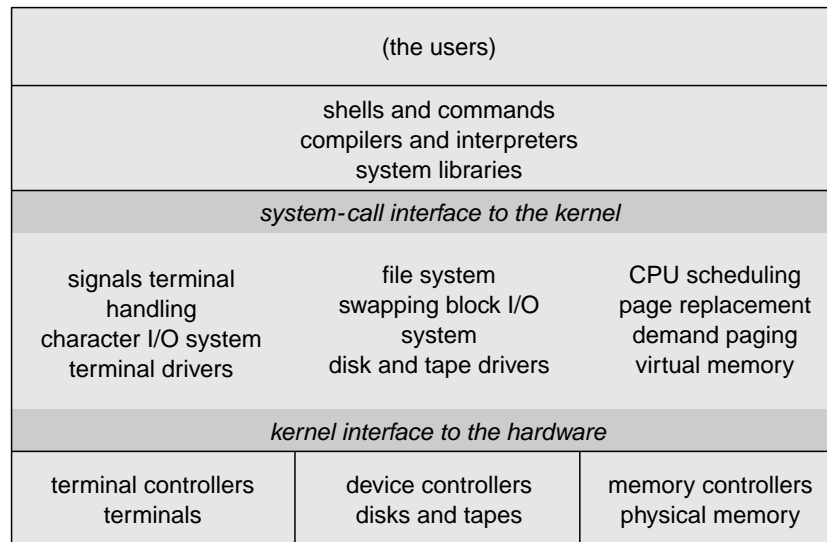
67

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.

68

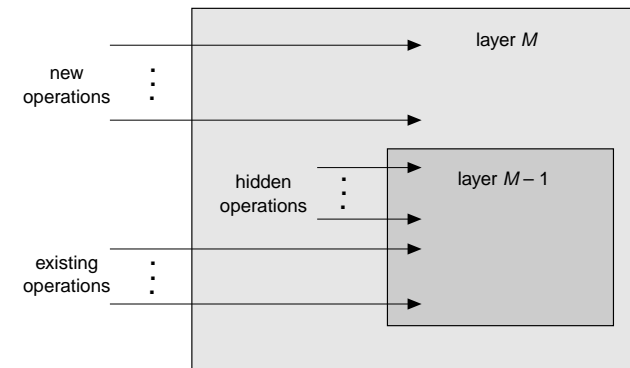
Struttura dei Sistemi Operativi – Unix originale



69

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.



70

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

71

Stratificazione

- Il sistema MULTICS era organizzato ad anelli concentrici (livelli)
- Per accedere ad un livello più 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)

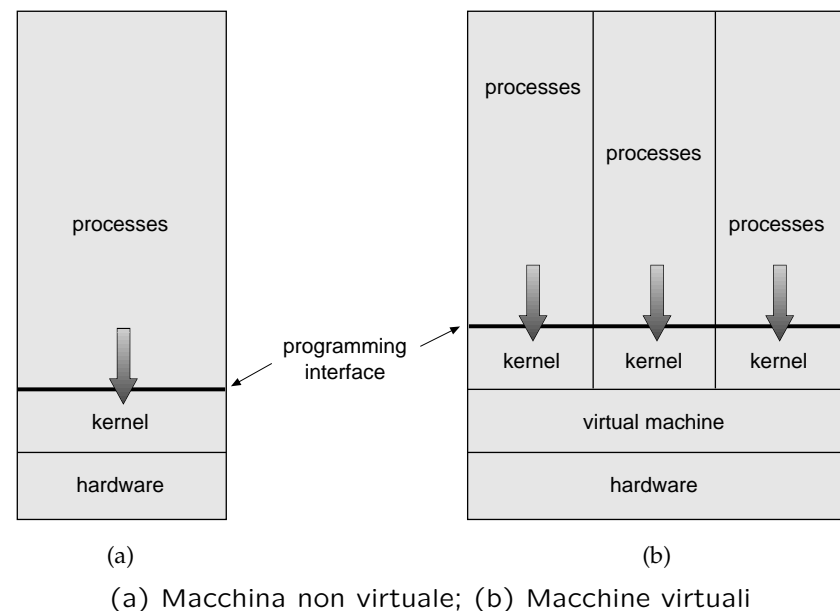
72

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"

73

Macchine Virtuali (Cont.)



74

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, . . .

75

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

76

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.

77

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 . . .)

78

Il sistema operativo UNIX

- Storia
- Principi di progetto
- Interfaccia per il programmatore
- Interfaccia utente

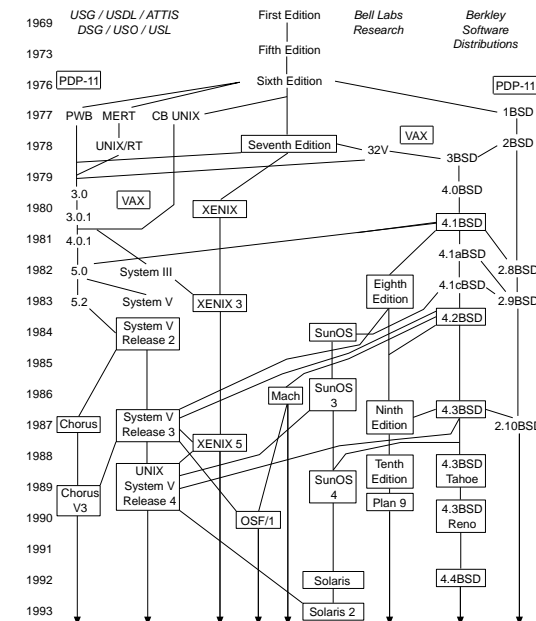
Storia

- Sviluppo originale nel 1969, di Ken Thompson e Dennis Ritchie del Research Group ai Bell Laboratories; incorpora caratteristiche di altri sistemi operativi, specialmente MULTICS.
- La terza versione fu scritta in C, sviluppato ai Bell Labs appositamente per supportare UNIX.
- Il più importante centro di ricerca su UNIX, non dell'AT&T: Università della California a Berkeley (*Berkeley Software Distributions*).
 - 4BSD UNIX risultò da fondi DARPA per lo sviluppo di un UNIX standard per uso governativo.
 - Sviluppato sul VAX, 4.3BSD fu una delle versioni più influenti sullo sviluppo dei seguenti S.O. Fu portata a molte altre piattaforme.

Storia (Cont.)

- Diversi progetti di standardizzazione cercano di consolidare le varianti di UNIX, per raggiungere una interfaccia di programmazione uniforme: ISO ha rilasciato POSIX; X/Open (ora Open Group), ha rilasciato XPG3 e XPG4, e le specifiche UNIX95 e UNIX98.
- Attualmente, la maggior parte degli UNIX commerciali rientra in XPG3 o XPG4. Tutti sono conformi a POSIX.
- Recentemente, c'è stato un ritorno al metodo di sviluppo originario, con il movimento Open Source (GNU/Linux, FreeBSD, OpenBSD, ...). Open Source risponde alla necessità naturale dei programmatori di riusare il più possibile il codice ed il lavoro già fatto—anche (e specialmente) dagli altri.

Storia (schematica) delle versioni di UNIX



Vantaggi dei primi UNIX

- Scritto in un linguaggio ad alto livello: portabile
- Distribuito in sorgente: modificabile
- Forniva un insieme di primitive potenti su piattaforme poco costose
- Piccolo, modulare, progettazione pulita.

UNIX è facilmente estendibile, senza snaturarlo: nel corso degli anni, aggiunto supporto per

- rete e ambienti distribuiti (TCP/IP, DCE, NFS, CORBA)
- architetture parallele (SMP, NUMA, NORMA)

83

- interfacce grafiche
- multithreading
- microkernel, ...

Principi di progetto di UNIX

“Il coltello svizzero del software” (Dennis Ritchie)

- Originalmente sviluppato da programmatori, per programmatori.
- Progettato per essere un sistema time-sharing multiutente.
- Separazione tra interfaccia e kernel; l'interfaccia (shell) può essere facilmente rimpiazzata.
- Il file system è ad albero; il meccanismo di controllo di accesso permette di controllare ogni file.
- Il kernel supporta i file come una sequenza non strutturata di byte.

84

- Supporto per più processi; un processo può facilmente creare nuovi processi.
- L'ambiente è interattivo, e fornisce diverse utilità per lo sviluppo di programmi.

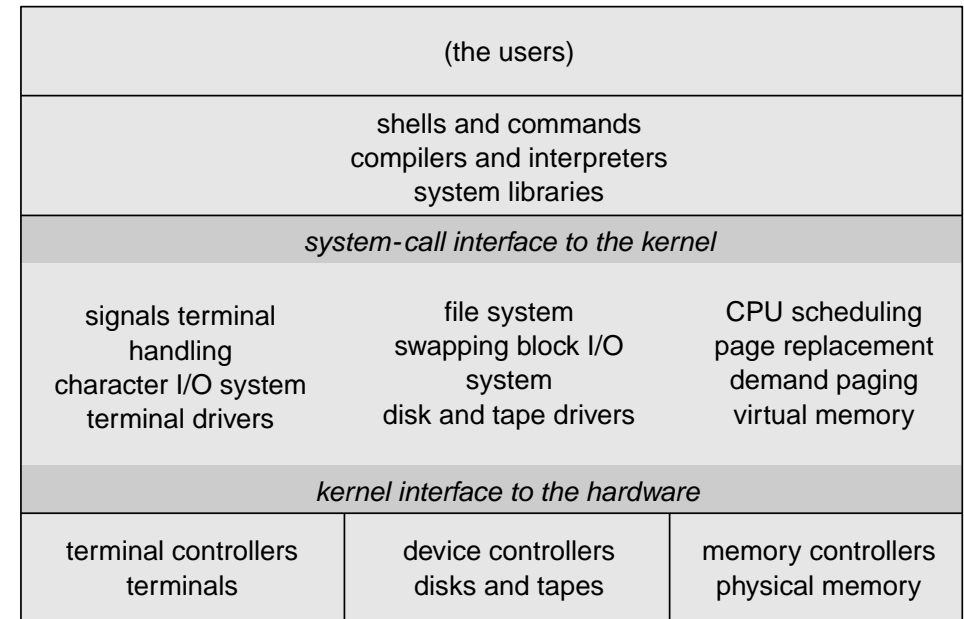
Interfaccia per il programmatore

Come la maggior parte dei sistemi, UNIX consiste di due parti separate:

- Kernel: tutto ciò che sta tra l'interfaccia delle system call e l'hardware
 - Implementa il file system, scheduling della CPU, gestione della memoria, protezione e altre funzionalità attraverso le chiamate di sistema
- Programmi di sistema: usano le chiamate di sistema per fornire funzioni di utilità, e.g., compilatori, gestione file, ...

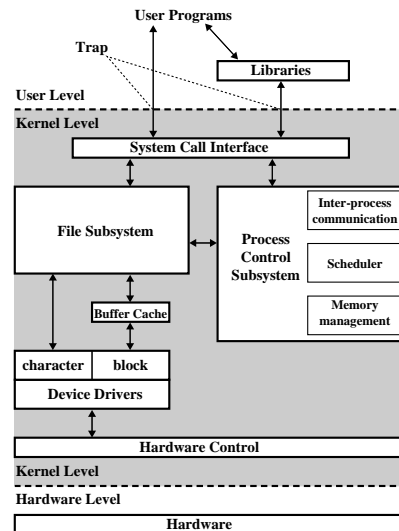
85

Struttura stratificata di 4.3BSD



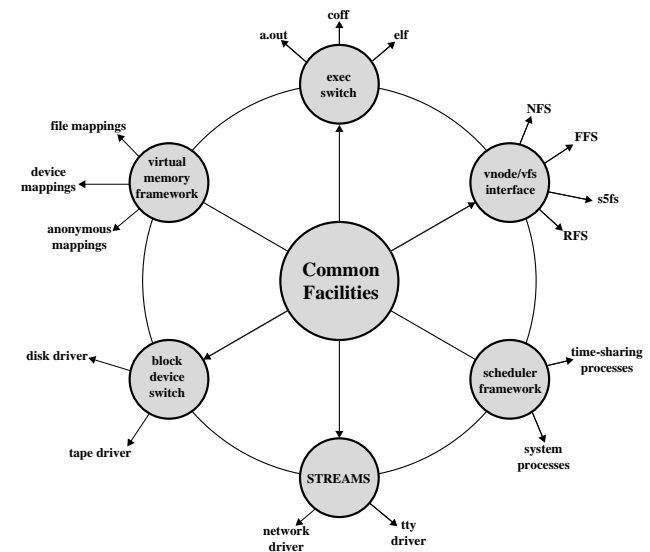
86

Kernel UNIX tradizionale (fino a 4.3BSD)



87

Kernel UNIX moderno (modulare)



88

Interfacce al programmatore e all'utente

- Le chiamate di sistema definiscono l'*interfaccia al programmatore (API)* di UNIX. Affiancate da molte librerie standard.
 - L'insieme dei programmi di sistema definiscono l'*interfaccia utente*; le stesse funzionalità sono disponibili anche attraverso *interfacce utente grafiche (GUI)*.
 - All'incirca, tre categorie di chiamate di sistema in UNIX
 - Gestione file
 - Controllo dei processi
 - Gestione delle informazioni
- I dispositivi vengono gestiti come i file (stesse system call).

89

Come funziona Unix?

- In fase di inizializzazione il processo *init* crea l'interprete dei comandi, chiamato *shell*
- Il processo *shell* legge un comando da tastiera (o dal file indicato dall'utente), lo interpreta, e provvede ad eseguirlo
- La shell accetta dall'utente dei comandi che devono essere scritti utilizzando un linguaggio specifico
- Esistono diverse versioni del programma shell: *csh*, *tcsk*, *ksh*, *bash* ecc. Volendo, si può scrivere la propria shell.
- Comando=programma che gira come utente superuser (con privilegi speciali)

90

Esempi di comandi

- Consultazione del manuale: `man`

```
elios> man passwd
```

```
PASSWD(1)          Unix Programmer's Manual          PASSWD(1)
```

```
NAME
```

```
passwd - change password
```

```
SYNOPSIS
```

```
passwd [ name ]
```

```
DESCRIPTION
```

```
passwd will change the specified user's password. Only the superuser is allowed to change other user's passwords. If the user is not root, then the old password is prompted for and verified.
```

```
...
```

91

File in Unix

- Come in qualunque sistema operativo, anche in Unix dati e codice sono memorizzati su file (sequenza di byte).
- Ogni file ha un nome seguito eventualmente da un'estensione, che viene normalmente utilizzata per indicare il tipo del file.
- I file sono raggruppati in directory.
- Esistono 3 tipi di file:
 - File normali (o flat) che contengono dati o codice,
 - File directory, che contengono altri file,
 - File speciali, dispositivi di I/O, trattati a tutti gli effetti come file.

92

Directory

- Sono utilizzati per raggruppare i file e sono organizzate ad albero.
- Un file e' individuato dal suo nome specifico, e dal cammino (path name) che bisogna fare nell'albero delle directory per arrivare ad esso.
- Il cammino e' indicato a partire dalla radice (indicata con il solo carattere /) con i vari nomi delle directory attraversate separate fra loro dallo stesso carattere /.
- Il path name specifica quindi la posizione del file nell'albero delle directory (attenzione: Unix distingue fra lettere minuscole e maiuscole).

93

Esempio di Directory

- Il path-name

/user/SysOp/Esame

- specifica la posizione del file Esame a partire dalla radice (pathname assoluto).
- Per evitare di usare cammino completo si utilizza il concetto di Working Directory (WD):
- E' possibile posizionarsi ad un certo punto (la WD) dell'albero delle directory (con il comando "cd"), e da quel momento in poi i nomi dei file possono essere dati in modo relativo a quella directory.

94

- Il comando di shell "pwd" provoca la scrittura della working directory corrente
- Al momento del login viene automaticamente aperta una WD detta "home directory", che per il superuser e' la root /. La home directory e' decisa dall'amministratore del sistema per ogni utente.
- La shell riconosce alcuni caratteri a cui da' un significato preciso. Fra gli altri qui ricordiamo:
 - . indica la directory corrente
 - .. indica la directory genitore
 - ~ indica la home directory
- Con il comando "cd nomedirectory" si puo' cambiare working directory.

Esempio di Directory

```
[giorgio:etabeta:303:/usr] cd bin
[giorgio:etabeta:304:/usr/bin] pwd
/usr/bin
[giorgio:etabeta:305:/usr/bin] cd ..
[giorgio:etabeta:306:/usr] cd java
[giorgio:etabeta:307:/usr/java] pwd
/usr/java
[giorgio:etabeta:312:/usr/java] cd ../local/share
[giorgio:etabeta:313:/usr/local/share] pwd
/usr/local/share
```

95

Organizzazione del File System

/ Directory generale del sistema, detta "root"
/bin Contiene i comandi piu' importanti per l'utente
/dev Contiene i file di accesso ai dispositivi fisici del calcolatore (dischi, memoria, porte seriali e parallele, ...)
/lib Contiene le librerie dinamiche necessarie al funzionamento dei programmi
/etc Contiene dei file e le sottodirectory per l'amministrazione del sistema
/tmp Contiene i file temporanei del sistema e degli utenti
/var Contiene sottodirectory con file che tendono a crescere di dimensioni.
/var/spool Contiene i file di spool temporanei di vari programmi: stampa, mail, ...
/var/adm Contiene i file con messaggi del sistema
/home Contiene le directory assegnate agli utenti
/sbin Contiene i programmi di partenza del sistema
/usr Contiene il grosso del sistema operativo. E' divisa a sua volta in sottodirectory
/usr/bin Contiene i comandi di base
/usr/sbin Contiene i comandi di amministrazione del sistema
/usr/include Contiene gli header file per la programmazione C e quindi per la creazione del kernel
/usr/man Contiene i manuali
/usr/lib Contiene le librerie per la programmazione e file di supporto per molti programmi

96

Contenuto di una directory

Il comando "ls" lista i nomi dei file della WD

"ls -l" da' una lista lunga dei file della WD, con indicazioni sul tipo e la lunghezza di ogni file

"ls -a" lista anche i file nascosti, cioe' quelli che cominciano con il carattere "."

"ls -R" elenca ricorsivamente i file della WD e le sue sottodirectory

97

Bit di protezione

- Utilizzando "ls -a" compare una lettera prima dei permessi:

d	directory
l	link simbolico
c	file speciale a caratteri
b	file speciale a blocchi

98

File speciali

- L'accesso ai dispositivi hardware avviene attraverso i device file.
- Essi sono quindi visibili attraverso le system call per la lettura e scrittura di file.
- Sono elencati nella directory /dev.

```
[giorgio:etabeta:298:~] cd /dev
[giorgio:etabeta:299:/dev] ls -al | more
total 284
drwxr-xr-x 18 root  root    86016 Sep 11 18:30 ./
drwxr-xr-x 20 root  root    4096 Sep 11 18:30 ../
crw----- 1 root  root    10, 10 Apr 11 2002 adbmouse
crw-r--r-- 1 root  root    10, 175 Apr 11 2002 agpgart
crw----- 1 root  root    10,  4 Apr 11 2002 amigamouse
crw----- 1 root  root    10,  7 Apr 11 2002 amigamouse1
```

99

File speciali

- *Block file*: associati a dispositivi organizzati a blocchi ed accessibili in modo diretto (es. dischi)
- *Character file*: associati a dispositivi organizzati come sequenze di caratteri (es. stampanti)
- Tutti i file speciali hanno un
 - Major Device Number: specifica la classe del device (floppy, terminale)
 - Minor Device Number: identifica il numero dell'unita'
- Una tabella Unix associa ad ogni file speciale un codice che identifica il device driver del dispositivo in questione.
- I device con lo stesso Major D.N. condividono il codice dell'unico driver di quel tipo di dispositivo

100

File speciali

<code>hda</code>	Primo disco fisso IDE
<code>hda1, hda2 ..</code>	Partizioni disco fisso IDE
<code>hdb</code>	Secondo disco IDE
<code>ttyS0, ttyS1 ...</code>	Porte seriali input
<code>cua0, cua1, ...</code>	Porte seriali output (modem)
<code>lp0, lp1, ...</code>	Porte parallele
<code>fd0, fd1, ...</code>	Unita' dischetti
<code>fd0H1440</code>	Unita' dischetti formattata 1.44 MB
<code>null</code>	Nulla

101

Protezioni

- Ad ogni file (e ad ogni processo) e' associato
 - un proprietario, individuato dallo uid (user identifier) e il gruppo di appartenenza (gid, group identifier) del proprietario,
 - un insieme di permessi, ognuno rappresentato con un bit, che ne definiscono l'utilizzo.
- I permessi sono di tre tipi: lettura (R), scrittura (W), esecuzione (X).
- Con il permesso di lettura si puo' listare il file, con quello di scrittura si puo' modificarlo, anche azzerarlo, ma non cancellarlo come file.
- Il permesso di esecuzione permette di eseguirlo, purché sia un binario eseguibile o uno script.

102

- Per una directory il permesso di lettura consente di listarne il contenuto, in scrittura indica la possibilita' di modificare una directory, infine il permesso di esecuzione indica la possibilita' di attraversarla per accedere a sue sottodirectory.
- I possibili utilizzatori sono di tre tipi: il possessore, il gruppo a cui appartiene, tutti gli altri utenti (other).

- Esempio:

<code>R W X</code>	<code>R W X</code>	<code>R W X</code>
<code>owner</code>	<code>group</code>	<code>world</code>

Esempio di permessi

- Una directory da' il diritto X e non quello W a other.
- Uno dei file listati nella directory da' il permesso W.
- Allora un qualunque utente puo' modificare, anche azzerare totalmente il file, ma non puo' cancellarne il nome dalla directory.
- Il permesso di scrittura per group (g) e per other (o) e' comunque protetto: si puo' cioe' scrivere sul file, ma non cambiarne gli attributi
- Solo il proprietario di un file puo' modificare gli attributi del file (ad esempio tramite il comando "chmod").
- I bit di protezione si applicano anche ai file speciali.

103

Comando chmod

- Si possono cambiare i permessi dei file con il comando chmod.

```
krypton> chmod o-w README
```

- elimina la possibilita' di scrittura all'owner.

104

Link tra file

- E' possibile indicare lo stesso file fisico con piu' di un nome, attraverso un link simbolico
- Questi file sono listati con il carattere l prima dei permessi, inoltre sulla destra compare il nome del file con il simbolo -> seguito dal nome del file a cui e' linkato

Esempio: pvm3 -> /home/elios/pvm/pvm3

- Il comando usato per creare il link fra pvm3 e /home/elios/pvm/pvm3 (la directory esistente) e'

```
ln -s /home/elios/pvm/pvm3 pvm3
```

105

- L'opzione -s specifica che si tratta di un link simbolico, cioe' solo fra i nomi dei file.
- I due file (la directory su cui si inserisce il link pvm3, e /home/elios/pvm/pvm3) possono anche essere su supporti fisici diversi

Cancellazione file

- Per la cancellazione di un file si usa il comando "rm nomefile".
- Le directory si cancellano con il comando "rmdir nomedir", solo se sono già vuote, oppure con il comando ricorsivo "rm -r nomedir"

- krypton > ls -li README PROVA

```
102 -rw-r--r--  2  giorgio  prof      1267 Apr 22 14:56 PROVA
102 -rw-r--r--  2  giorgio  prof      1267 Apr 22 14:56 PROVA1
```

```
krypton > rm PROVA1
krypton > ls -li PROVA*
```

```
102 -rw-r--r--  1  giorgio  prof      1267 Apr 22 14:56 PROVA
```

106

Gestione del file system

- Unix vede i file in modo indipendente dal supporto fisico su cui sono scritti.
- La struttura ad albero delle directory e' presente anche sul supporto su cui i file si trovano.
- Occorre allora agganciare la radice di questa struttura ad una foglia dell'albero che origina da root.
- Supponiamo di voler *montare* il file system contenuto in un floppy disc come sottodirectory di /user/SysOp, di nome AA1999
- Possiamo usare il comando "mount" come segue

```
mount    /dev/fd0  /user/SysOp/AA1999
```

107

- Il comando mount potrebbe essere disabilitato, cioè utilizzabile solo dal superuser.
- Al momento dell'inizializzazione del sistema, sono eseguiti comandi di mount che provvedono a montare eventuali supporti necessari.
- Dopo aver montato un supporto, non e' piu' necessario riferirlo per accedere ai file.
- Ad esempio, per accedere al file "PROVA" nel floppy disc montato e' sufficiente il suo nome: /user/SysOp/AA1999/PROVA
- Il comando "umount" consente di *smontare* un FS, ma solo quando non ci sono attività ancora da completare che accedono a quei file
- Si deve solo specificare il device, non la directory:

```
krypton > umount /dev/fd0
```

Demoni e processi

- I demoni sono processi, lanciati per lo piu' all'avvio del sistema, che aspettano le richieste dell'utente (stile client-server).
- Ad esempio il demone lpd aspetta le richieste di stampa date con il comando lpr.
- Il comando per avere la lista dei processi e' ps. Senza opzioni vengono listati solo i processi di cui il richiedente e' proprietario, e che sono stati attivati durante la stessa sessione (fra lo stesso login - logout). L'opzione aux consente di avere il listato completo.

108

Monitoraggio processi: il comando ps(1)

disi > ps

PID	TTY	STAT	TIME	COMMAND
16200	p2	s	0:04.92	-tcsh
16330	p2	R	0:01.87	ps

krypton > ps -aux

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	S	STARTED	TIME	COMMAND
root	315	0,0	0,2	1,39M	48K	??	I	gui 19	0:29.26	/usr/sbin/inetd
root	320	0,0	0,0	1,34M	8K	??	I	gui 19	0:05.74	/usr/sbin/cron
root	351	0,0	0,4	1,38M	112K	??	I	gui 19	0:07.55	/usr/sbin/lpd
root	274	0,0	0,4	1,55M	128K	??	I	gui 19	3:22.66	/usr/sbin/snmpd
gianuzzi	5843	20,0	1,7	2,03M	520K	ttyp4	S	14.42.18	0:01.48	- (tcsh)
root	376	7,0	10,2	14,1M	3,1M	??	S <	gui 19	04:21:18	/usr/bin/X11/X -nice -2 -auth /usr/var/
gianuzzi	4389	3,0	2,3	9,49M	696K	??	S	09.38.53	0:08.86	/usr/bin/X11/mwm

Dove (con varie opzioni):

User	proprietario del processo
PID	identificatore del processo
%CPU, %MEM	Percentuali utilizzo CPU e memoria nell'ultimo minuto
SIZE	Dimensione del processo
RSS	KB di memoria occupata
TTY	Porta seriale associata al programma
STAT	Stato del processo
	S=sleeping, R=running, I=idle, Z=zombie
START	Orario di avvio del processo
TIME	Tempo utilizzato effettivamente
COMMAND	Comando con cui e' stato lanciato il processo

109

Alcuni Processi

- init e' l'unico processo lanciato dal kernel, essenziale in quanto il sistema operativo esegue solo funzioni e non comandi.
- I processi della lista il cui nome finisce con d sono demoni, mentre getty e' il programma di gestione di accesso al sistema
- Il comando per interrompere un processo in background e' kill pid dove pid e' il process identifier.
- Ad esempio kill 315 causa, se si hanno i necessari privilegi, la morte del processo 315 (demone gestione rete).

110

Generalita' sulla Shell

- E' un programma che fa da intermediario fra l'utente e il kernel.
- Legge una linea di caratteri e la interpreta attivando i processi come richiesti dall'utente.
- Esistono diverse shell (sh, csh, tcsh, bash ecc.). La bash (FSF) e' quella normalmente usata in Linux, la shell bash.

Funzionamento della Shell

111

- La shell accetta comandi per Unix dall'utente e li manda in esecuzione, secondo le direttive date.
- Assegna automaticamente ad ogni programma mandato in esecuzione: lo standard input, cioè da dove riceve i dati (tastiera), lo standard output, dove deve stampare i risultati (schermo), lo standard error, dove scrive i messaggi d'errore (schermo).
- Un comando normalmente invia l'output su monitor (ad esempio ls). Come fare se invece si vuole la directory listata su file per poterla ad esempio stampare?
- Si usa il linguaggio delle shell: ">", "<" e ">>" seguiti da un nome di file, *ridirigono* l'output, l'input e l'error al file specificato

112

Ridirezionamento

- krypton > cat README
..... < listato del file README >
- krypton > cat > file2
linea1 --> scritti dall'utente
linea2
^D
- krypton > cat README file2 > file1
- Con il primo comando si lista su video il contenuto del file README
- Con il secondo si crea il file file2, che conterra' le due righe scritte da tastiera dall'utente
- Con il terzo i file README e file2 sono listati in sequenza sul file file1

113

Problema

- Vogliamo sapere quanti processi attivi ci sono appartenenti all'utente gianuzzi.
- Il comando "ps -aux" ci consente di conoscere tutti i processi (la lista potrebbe essere lunga).
- Il comando "grep stringa-caratteri lista-file" ci consente di cercare le occorrenze della stringa all'interno dei file indicati.

- Posso usare i comandi

```
krypton > ps aux > proc
krypton > grep gianuzzi proc
gianuzzi 5843 520K ttyp4 S 14.42.18 0:01.48 - (tcsh)
gianuzzi 4389 696K ?? S 09.38.53 0:08.86 /usr/bin/X11/mwm
```

- Il file "proc" va tuttavia cancellato. Si può fare in altri modi?

114

Pipe

- In Unix c'e' la possibilita' di definire un pipe (tubo) di comandi, in cui l'output di un comando e' usato come input del successivo.
- krypton > ps aux | grep gianuzzi

gianuzzi 5843 520K ttyp4 S 14.42.18 0:01.48 - (tcsh)
gianuzzi 4389 696K ?? S 09.38.53 0:08.86 /usr/bin/X11/mwm
- Il carattere | (che e' un altro dei caratteri gestiti dalla shell) separa due comandi: l'output del primo e' l'input del secondo.
- Si puo' fare una sequenza di piu' di due comandi.

115

Espressioni regolari

- Le espressioni regolari sono espressioni costruite su operatori interpretati su insiemi di stringhe.
- Tutte le shell supportano linguaggio per comandi con espressioni regolari
- Ad esempio
 - l'operatore "*" sta per qualsiasi numero (anche zero) di caratteri diversi da "/"
 - "?" puo' essere sostituito da un singolo carattere.
 - la stringa racchiusa fra doppi apici "..." e' presa cosi' com'e'

116

Interpretazione espressioni regolari

- La shell legge il comando
- Se trova caratteri speciali li trasforma nella lista di nomi di file presenti nel sistema che possono essere costruiti da quell'argomento
- Manda in esecuzione il comando sulla lista di file

117

Esempio di espressioni regolari

- ```
krypton > ls
f1.c f2.c f3.txt m.tex

krypton > ls *
f1.c f2.c f3.txt m.tex

krypton > ls f*
f1.c f2.c f3.txt

krypton > ls *.c
f1.c f2.c

krypton > ls f*.*
f1.c f2.c
```

- Ci sono altre possibilita', ad esempio /usr/[a-f]\* indica tutti i file della directory /usr il cui nome inizia con una lettera minuscola fra a e f comprese.

118