

# **Il sistema operativo UNIX**

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

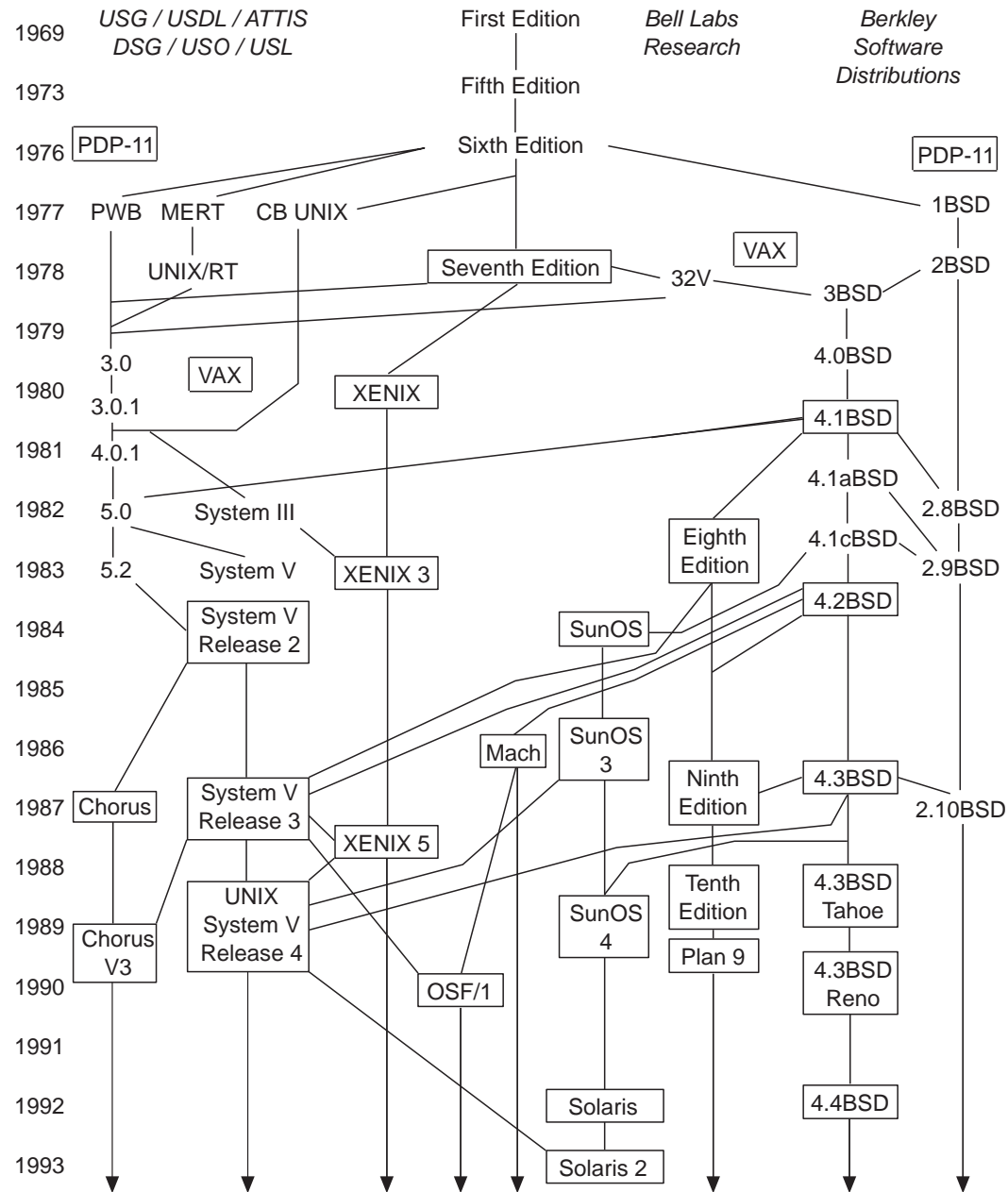
# Storia

- 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)

- 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.

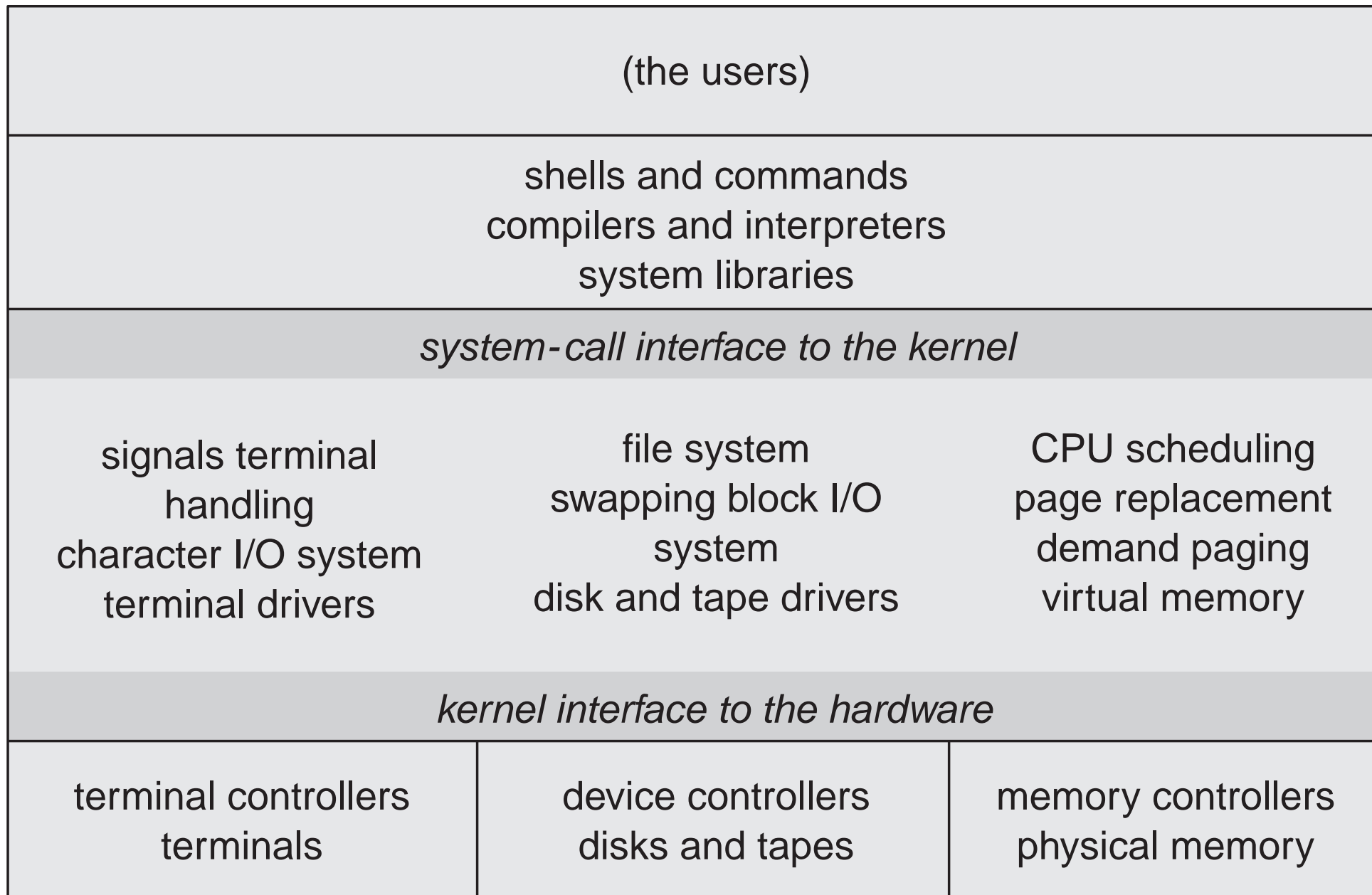
- Supporto per più processi; un processo può facilmente creare nuovi processi.
- L'ambiente è interattivo, e fornisce diverse utilità per lo sviluppo di programmi.

## Interfaccia per il programmatore

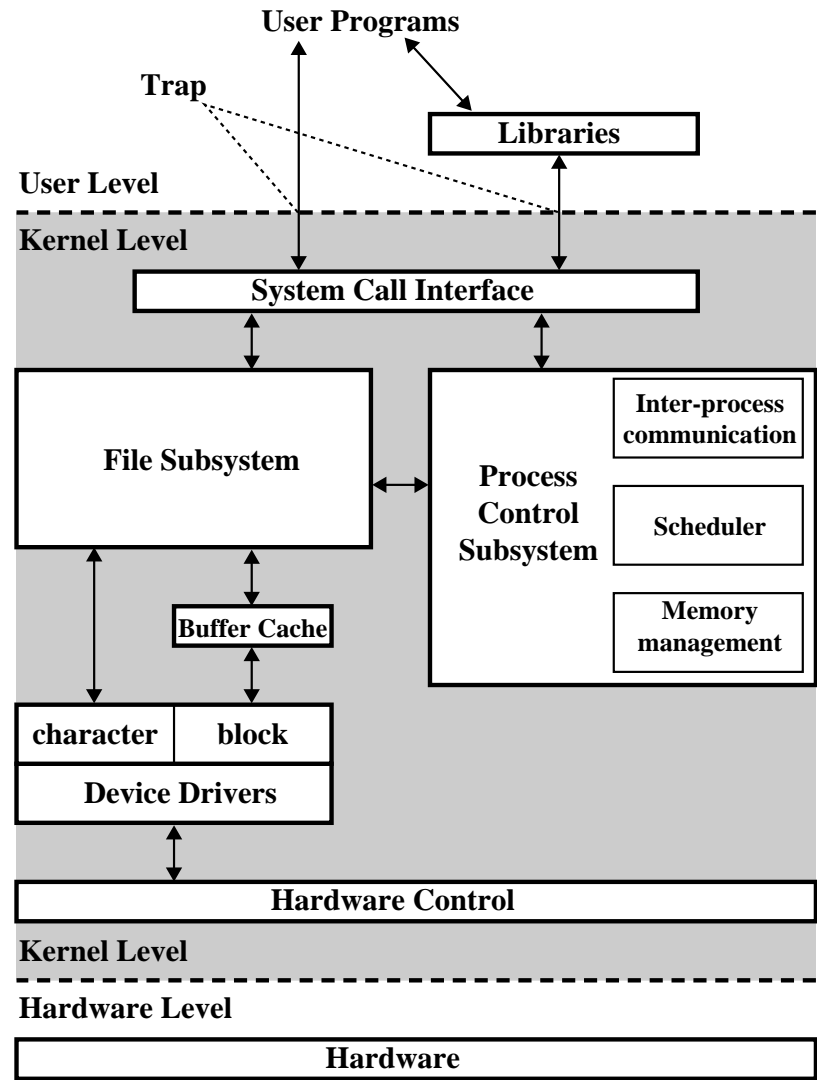
Come la maggior parte dei sistemi, UNIX consiste di due parti separate:

- 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, ...

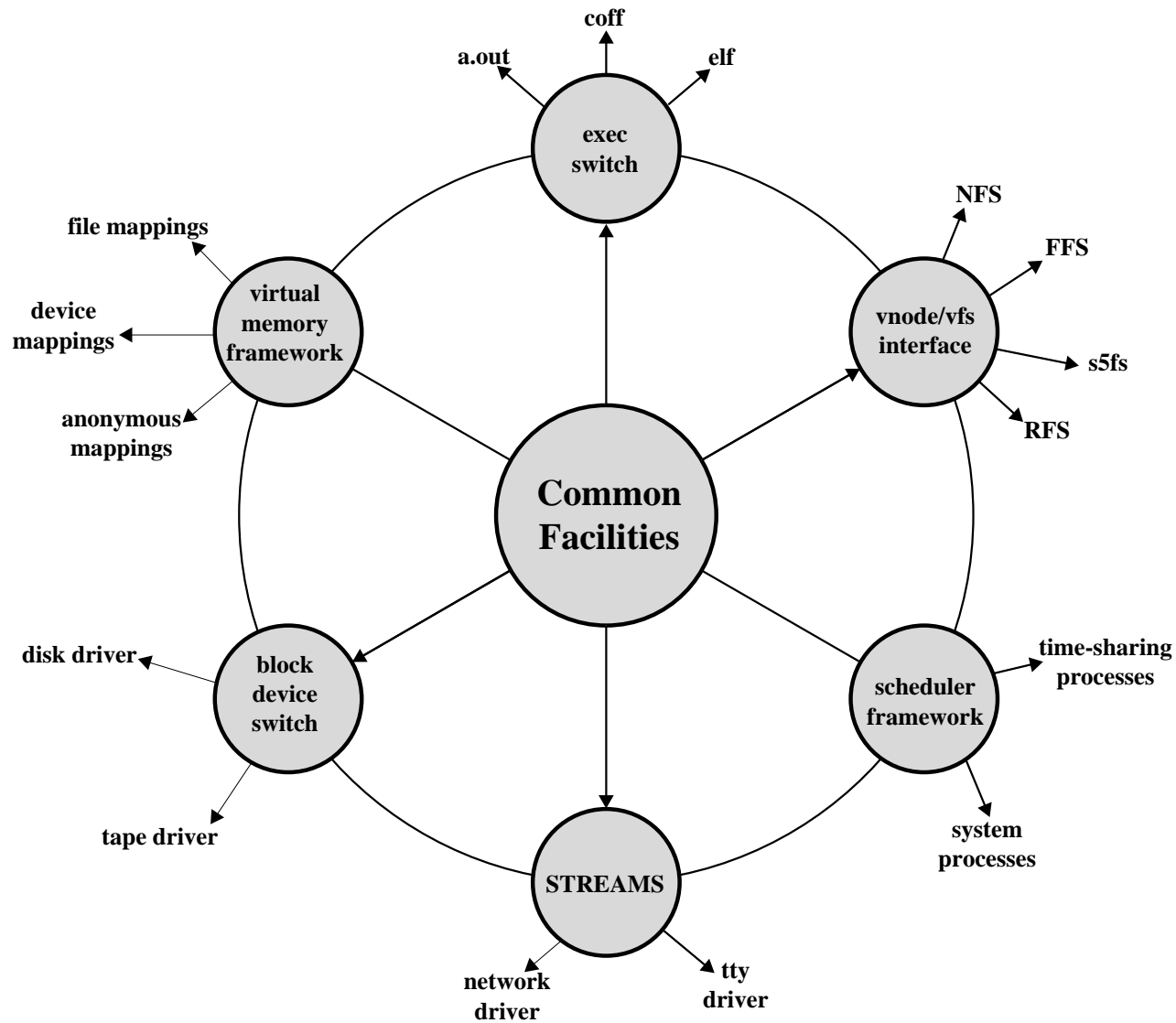
# Struttura stratificata di 4.3BSD



# Kernel UNIX tradizionale (fino a 4.3BSD)



# Kernel UNIX moderno (modulare)



# 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).

## 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: *cs*, *tcsh*, *ksh*, *bash* ecc. Volendo, si può scrivere la propria shell.
- Comando=*program* programma che gira come utente *superuser* (con privilegi speciali)

## 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.
```

```
...
```

# 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.

# Directory

- Sono utilizzati per raggruppare i file e sono organizzate ad albero.
- Un file è individuato dal suo nome specifico, e dal cammino (path name) che bisogna fare nell'albero delle directory per arrivare ad esso.
- Il cammino è 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).

## 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):
- È 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.

- 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 è la root /. La home directory è 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
```

# 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                                   |

## 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, cioè quelli che cominciano  
con il carattere `."`

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

## 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   |

# 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
```

# 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

## File speciali

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

# Protezioni

- In Unix si possono definire gruppi di utenti (i.e. definiti tramite l'insieme di login di ogni singolo utente appartenente al gruppo)
- L'amministratore di sistema (superuser, root) ha diritti speciali (può accedere ad ogni file, ecc.)
- Ad ogni file (e ad ogni processo) e' associato
  - un proprietario, individuato dallo uid (user identifier) ed il gruppo di appartenenza (gid, group identifier) del proprietario,
  - un insieme di permessi per tre classi di utenti (il proprietario, il gruppo, e gli altri utenti), 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.
- 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:

|       |       |       |
|-------|-------|-------|
| R W X | R W X | R W X |
| owner | group | world |

## 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.
- Solo il proprietario (u) 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.

# Comando chmod

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

```
[giorgio:etabeta:132:~] touch prova
[giorgio:etabeta:132:~] ls -al prova
-rw-r--r--    1 giorgio  collab                0 Sep 29 10:51 prova
[giorgio:etabeta:133:~] chmod u+x prova
```

- assegna la possibilita' di esecuzione al proprietario (u)

```
[giorgio:etabeta:137:~] chmod g-r prova
[giorgio:etabeta:138:~] ls -al prova
-rwx---r--    1 giorgio  collab                0 Sep 29 10:51 prova*
```

- elimina la possibilita' di lettura per il gruppo (g)

```
[giorgio:etabeta:143:~] chmod o+wx prova
```

```
[giorgio:etabeta:144:~] ls -al prova
-rwx---rwx    1 giorgio  collab          0 Sep 29 10:51 prova*
```

- assegna il permesso di scrittura ed esecuzione a tutti gli altri utenti (o)

```
[giorgio:etabeta:145:~] chmod a-w prova
[giorgio:etabeta:146:~] ls -al prova
-r-x---r-x    1 giorgio  collab          0 Sep 29 10:51 prova*
```

- elimina il permesso di scrittura per tutti gli utenti (a)

## 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
```

- L'opzione -s specifica che si tratta di un link simbolico, cioè 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 -l README PROVA

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

```
krypton > rm PROVA1
```

```
krypton > ls -l PROVA*
```

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

# 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
```

- 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 è più necessario riferirlo per accedere ai file.
- Ad esempio, per accedere al file "PROVA" nel floppy disc montato è 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.

# Monitoraggio processi: il comando ps

```
[giorgio:etabeta:112:~] ps -aux | more
```

| USER    | PID   | %CPU | %MEM | VSZ  | RSS  | TTY  | STAT | START | TIME | COMMAND           |
|---------|-------|------|------|------|------|------|------|-------|------|-------------------|
| giorgio | 13114 | 0.0  | 0.4  | 2968 | 608  | tty1 | S    | Sep27 | 0:00 | -tcsh             |
| giorgio | 13133 | 0.0  | 0.6  | 2244 | 876  | tty1 | S    | Sep27 | 0:00 | /bin/sh /usr/bin/ |
| giorgio | 13144 | 0.0  | 0.4  | 2276 | 512  | tty1 | S    | Sep27 | 0:00 | xinit /etc/X11/xi |
| giorgio | 13148 | 0.0  | 2.0  | 7424 | 2564 | tty1 | S    | Sep27 | 0:00 | /usr/bin/gnome-se |
| giorgio | 13163 | 0.0  | 1.2  | 6140 | 1572 | ?    | S    | Sep27 | 0:02 | gnome-smproxy --s |
| giorgio | 13177 | 0.0  | 2.3  | 5720 | 2944 | ?    | S    | Sep27 | 0:17 | sawfish --sm-clie |
| giorgio | 13188 | 0.0  | 1.5  | 3760 | 1980 | ?    | S    | Sep27 | 0:04 | xscreensaver -no- |

Dove:

|            |  |
|------------|--|
| USER       | proprietario del processo                                    |
| PID        | identificatore del processo                                  |
| %CPU, %MEM | Percentuali utilizzo CPU e memoria nell'ultimo minuto        |
| VSZ        | Dimensione totale del processo in memoria virtuale           |
| RSS        | KB di memoria principale 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 cumulativo di esecuzione in CPU                        |
| COMMAND    | Comando con cui e' stato lanciato il processo                |

## Uccisione di processi: comando kill

- Il comando per interrompere un processo in background e' kill pid dove pid e' il process identifier.
- Ad esempio kill 13114 causa, se si hanno i necessari privilegi, la morte del processo 13114.

## Generalita' sulla Shell

- È 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 è quella normalmente usata in Linux, la shell bash.

# Funzionamento della Shell

- 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 ">2" seguiti da un nome di file, *ridirigono* l'output, l'input e l'error al file specificato

# 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

# 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?

# Pipe

- In Unix c'è la possibilità di definire un pipe (tubo) di comandi, in cui l'output di un comando è 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 è un altro dei caratteri gestiti dalla shell) separa due comandi: l'output del primo è l'input del secondo.
- Si può fare una sequenza di più di due comandi.

# 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 "..." è presa così com'è

## 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

## 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 possibilità, 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.