

Modulo 1

Introduzione all'uso di Linux

Laboratorio di Sistemi Operativi I
Anno Accademico 2008-2009

Copyright © 2005-2007 Francesco Pedullà, Massimo Verola

Copyright © 2001-2005 Renzo Davoli, Alberto Montresor (Università di Bologna)

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation;

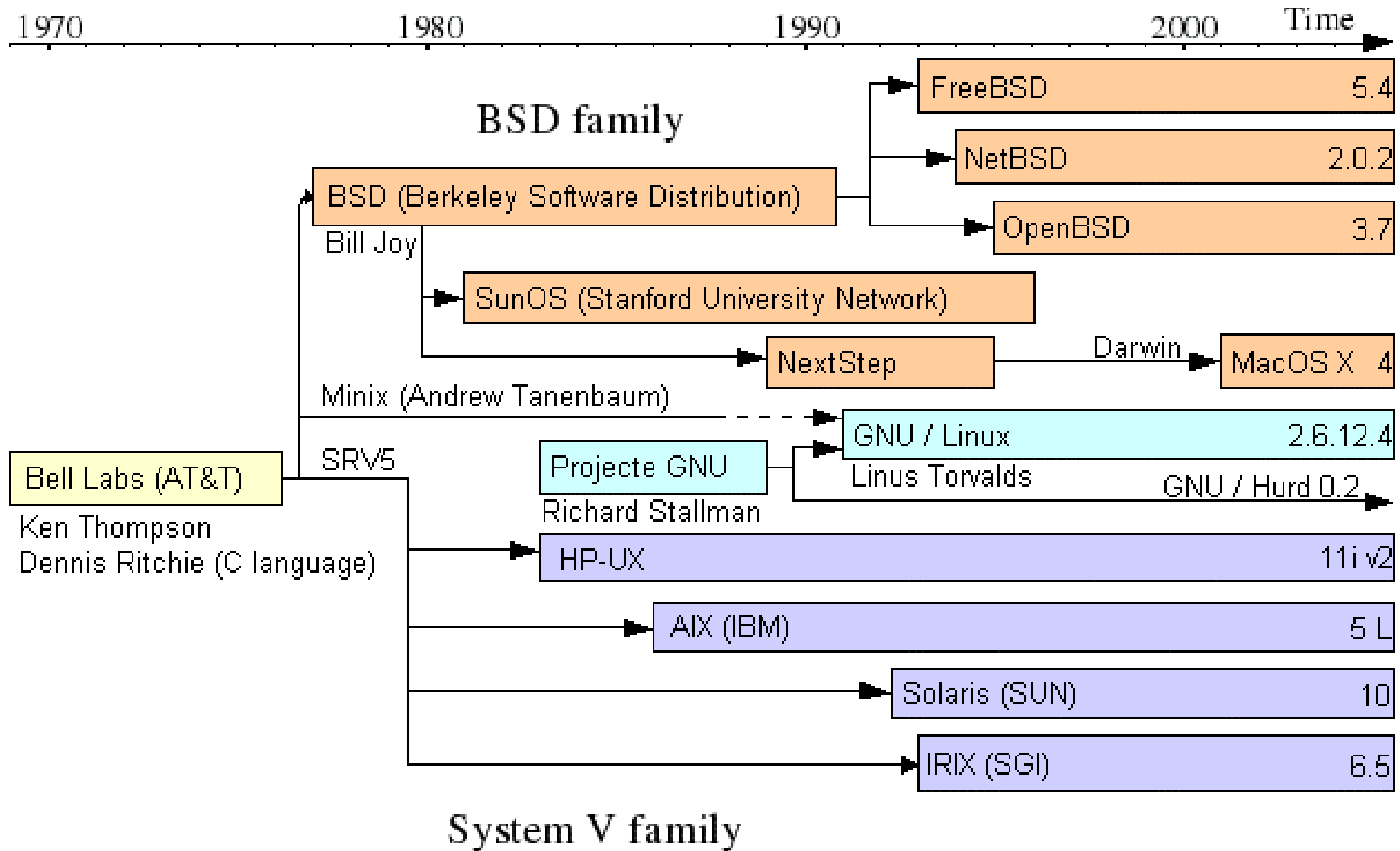
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license can be found at: <http://www.gnu.org/licenses/fdl.html#TOC1>

Introduzione

- Linux è un sistema operativo “free” ispirato a Unix, creato originariamente da Linus Torvalds e cresciuto con il supporto di una moltitudine di sviluppatori in tutto il mondo
- Strettamente connesso alla suite GNU (compilatore, linker, debugger, etc.), il sistema può essere correttamente chiamato GNU/Linux
- Sviluppato sotto la *GNU General Public License* (GPL), il codice sorgente è gratuitamente e liberamente disponibile per chiunque
- Inizialmente sviluppato per microprocessori Intel 386, adesso è disponibile su tutte le architetture di calcolo più diffuse
- E' utilizzato in una molteplicità di sistemi: personal computers, supercomputers, sistemi *embedded* come router/firewall, telefoni cellulari e videoregistratori

Storia di UNIX

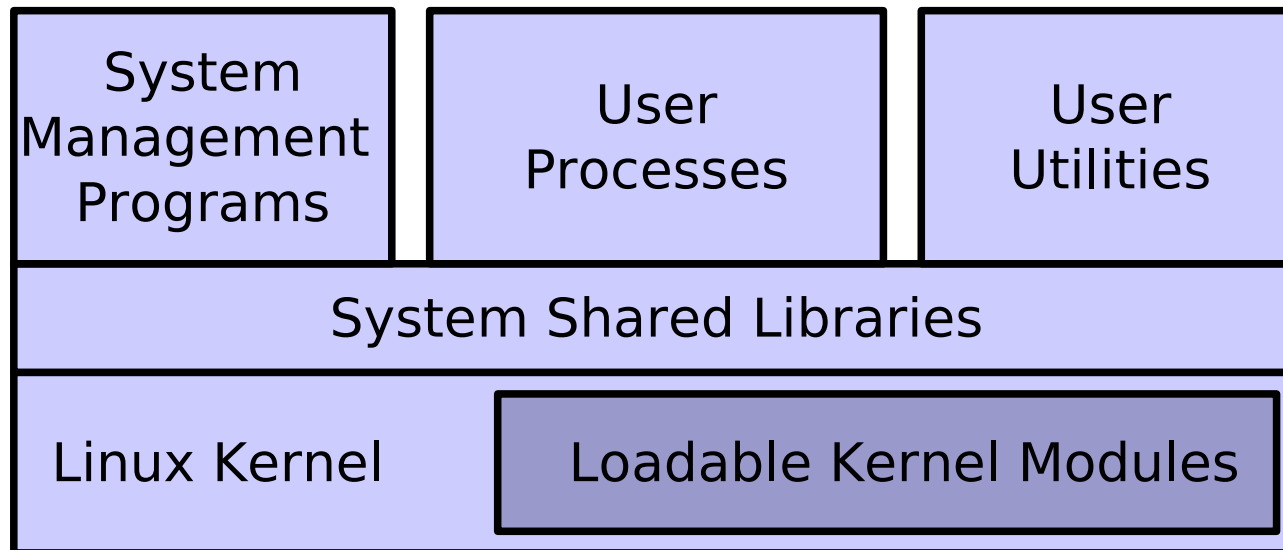


Caratteristiche di Unix/Linux

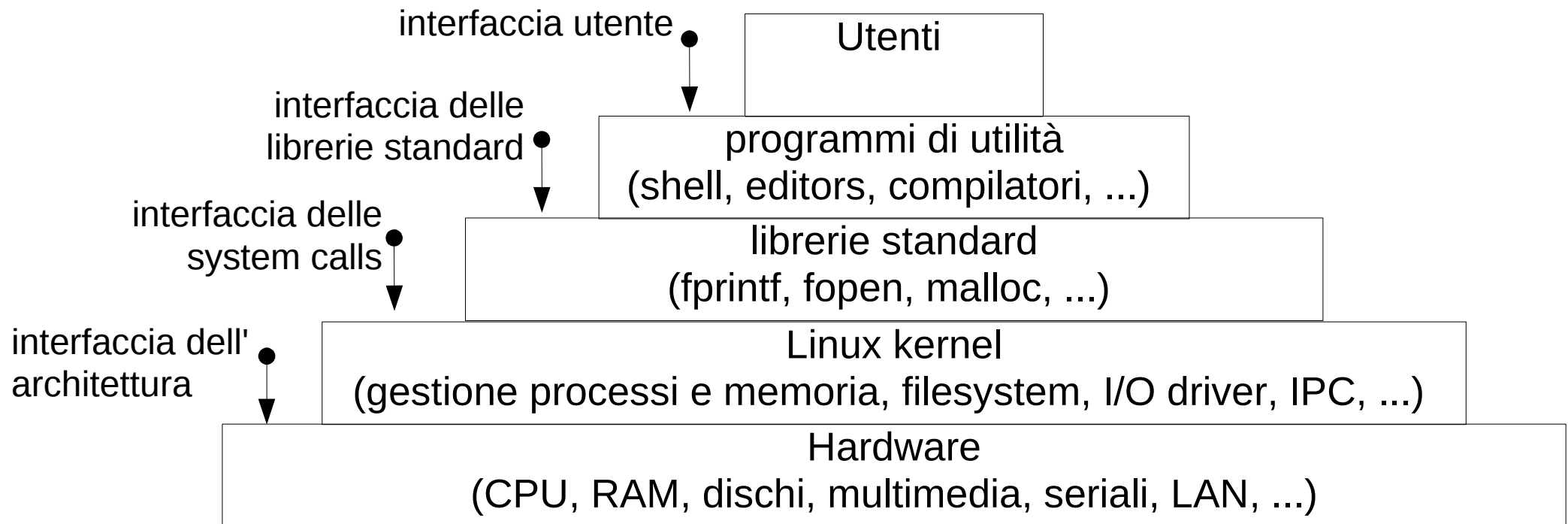
- ♦ Multiutente / multitasking
- ♦ Composizione di tool (filtri) a livello utente
- ♦ Progettato da programmatori per programmatori
- ♦ “Everything is a file”
- ♦ Kernel monolitico
- ♦ Modularità (e.g., file system)
- ♦ Scritto quasi integralmente in C (come Unix)

Principi generali di progettazione

- ◆ **Principali principi di progettazione:**
 - ◆ velocità, efficienza e standardizzazione
- ◆ **Rapporti con altri UNIX**
 - ◆ "compliant" con le specifiche POSIX
 - ◆ API fortemente basata su UNIX SVR4
 - ◆ molti tool e librerie derivanti da BSD
 - ◆ Integrazione con GNU



I layers di Linux



Il Kernel

- ♦ Il kernel supporta l'utilizzo di moduli dinamici (Loadable Kernel Module - LKM)
- ♦ i moduli sono sezioni del codice del kernel che possono essere compilati, caricati e scaricati in modo indipendente dal resto del kernel
- ♦ un modulo del kernel può implementare tipicamente un device driver, un file system, o un protocollo di networking
- ♦ l'interfaccia dei moduli permette a terze parti di scrivere o distribuire, in base ai propri termini, device driver o altro codice che non può essere distribuito sotto GPL

Login

- ♦ La procedura di login serve per:
 - ♦ autenticare l'utente
 - ♦ configurare un *environment* per l'utente
 - ♦ far partire la *shell*

```
Fedora Core release 3 (Heidelberg)
Kernel 2.6.9-1.667 on an i386

antarctic login: penguin
Password:
Last login: Thu Aug 18 17:13:26 on :0
[penguin@antarctic ~]$ echo $SHELL
/bin/bash
[penguin@antarctic ~]$
```

- ♦ Se lo *username* e la *password* sono corrette, l'utente ottiene il *prompt* (messaggio che il sistema è pronto) della shell

Logout

- ♦ La procedura di logout si usa per:
 - ♦ rilasciare le risorse allocate per l'utente
 - ♦ terminare i processi appartenenti all'utente
 - ♦ concludere la sessione di lavoro dell'utente

```
[penguin@antarctic ~]$ logout
```

```
Fedora Core release 3 (Heidelberg)  
Kernel 2.6.9-1.667 on an i386
```

```
antarctic login:
```

- ♦ In alternativa al comando `logout` si possono usare anche `exit` o `Ctrl-d`

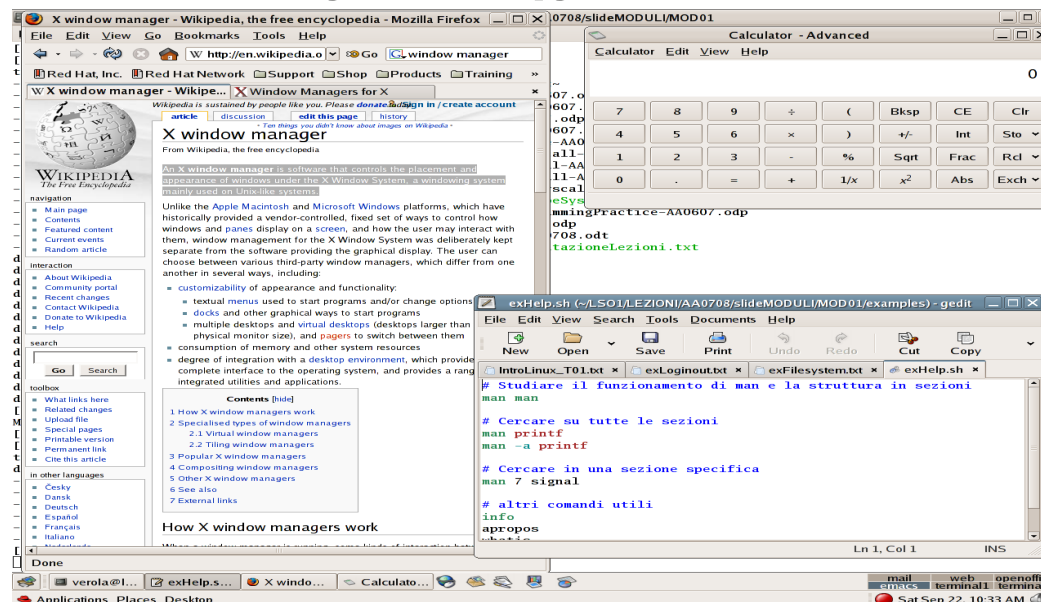
Interfaccia con l'utente

- Due modi diversi di interazione con l'utente:
 - interfaccia a comandi (gestita dalla shell)

```

verola@localhost:~/LSO1/DOCS
File Edit View Terminal Tabs Help
[verola@localhost LSO1]$ cd DOCS
[verola@localhost DOCS]$ ls -F
AdvLinuxProgramming/  FrontPageBooks/  Make/  VIM/
Bash/                 GCC/              MemMngmtTutorial/  Virus/
binutils/             GDB/              Partition+Filesystem/  Wikipedia/
diagrams/             kernel/            processControl/
ELF/                  Library/           SecureProgramming/
Filesystem/           Linux/             stackOverflow/
[verola@localhost DOCS]$
  
```

- interfaccia grafica (gestita dal server X Windows)



Interfaccia a comandi

- l'utente interagisce con il sistema inviando ad esso dei comandi
- ogni comando è costituito da una sequenza di caratteri alfanumerici terminata dal tasto <INVIO> (<Enter> o <Return> o ↵)
- i comandi seguono una sintassi ben definita, che è analizzata da un *interprete dei comandi* (in Linux è la Shell ed anche l'utility di SO invocata dalla Shell per conto dell'utente)
- l'interprete dei comandi verifica la correttezza della stringa sulla linea di comando ed esegue le azioni richieste, eventualmente generando dei risultati o messaggi in output sul video

```
[penguin@antarctic ~]$ du -sh /home/penguin
507M    /home/penguin
[penguin@antarctic ~]$ di -sh /home/penguin
bash: di: command not found
[penguin@antarctic ~]$ du -sh (/home/verola
bash: syntax error near unexpected token `('
```

Interfaccia grafica

- si basa sull'utilizzo di icone e pannelli grafici per l'inserimento dei dati di input o la selezione di argomenti da menù predefiniti
- risponde all'esigenza di maggiore facilità d'uso e apprendimento del SO, richiede un terminale grafico e un mouse, oltre alla consueta tastiera
- l'interfaccia grafica in ambiente Linux è basata su **X Window System (X11** o anche solo **X**), che è un'applicazione client/server dedicata a fornire le primitive per realizzare GUI:
 - **X server:** gestisce mouse e tastiera, disegna all'interno delle finestre grafiche riservate agli *X Client* (processi che comunicano con l'X Server) i dati di output che tali client hanno inviato al server
 - **X client:** rimane in attesa dei dati di input provenienti dal server per elaborarli e invia al server i risultati dell'elaborazione
- le GUI sotto X Window System sono denotate come **X window manager**, cioè quei software che controllano il posizionamento, le funzionalità e lo stile grafico delle finestre
- l'utente può scegliere tra vari X window manager, i quali possono differire per vari fattori tra cui livello di personalizzazione ed integrazione con l'ambiente del desktop

Comandi principali

I comandi base e le utilities principali di Linux (derivati da Unix) sono:

- Creazione directory e file e spostamenti nel filesystem: `ls cd pwd mkdir rmdir tree touch cp rm find`
- Visualizzazione e editing di file: `more ed vi emacs ex`
- Elaborazione di testo: `echo cat grep sort uniq sed awk tail tee head cut tr split printf`
- Confronto tra file: `comm cmp diff patch`
- Programmi vari di utilità: `test xargs find`
- Amministrazione di sistema: `chmod chown ps su who`
- Comunicazione via rete: `mail telnet ftp finger ssh`
- Lancio di shell: `sh bash csh ksh tcsh`

Esercizi

- ◆ Dopo aver ottenuto un'utenza sul PC e una password, effettuare login, cambiare password e fare logout nei diversi modi:
 - ◆ chiudendo l'intera sessione grafica
 - ◆ chiudendo la terminal window
- ◆ Aprire una nuova terminal window, familiarizzare con le funzioni del window manager (*minimize, maximize, resize, move, close*), se non note!
- ◆ Analizzare il meccanismo dell'interfaccia testuale:
 - ◆ il prompt
 - ◆ l'inserimento di un comando
 - ◆ l'output del comando
- ◆ Lanciare il comando `man man` e spiegarne il significato

Filesystem - I

- ♦ Il filesystem è un modulo del SO dedicato all'archiviazione delle informazioni sulle memorie di massa
- ♦ Esistono diversi tipi di file system (ntfs, fat, ext2, ext3, jfs, ...)
- ♦ Diversi file system possono essere utilizzati contemporaneamente
- ♦ Le funzioni principali sono:
 - ♦ semplificare gli accessi alle memorie di massa: per leggere e scrivere dati non si devono specificare gli indirizzi fisici mediante *head*, *sector*, *cylinder*, ma basta riferirsi a nomi di file e directory
 - ♦ offrire all'utente una modalità strutturata ed efficiente per organizzare i propri dati
 - ♦ gestire i vari dischi collegati al sistema, assegnando o rilasciando aree di memoria secondaria in base alle richieste dei programmi che utilizzano il filesystem

Filesystem - II

- ◆ Altre funzioni importanti del filesystem sono:
 - ◆ Implementare degli schemi di protezione dei dati (diritti di accesso a file e directory) e politiche di limitazione dell'uso dello spazio disco (quotas)
 - ◆ Supportare l'accesso diretto a dispositivi periferici (**/dev**)
 - ◆ Permettere l'accesso ad informazioni sul sistema (**/proc**)
 - ◆ Permettere l'accesso a dati remoti (samba, NFS)
 - ◆ Permettere l'accesso a dati di altri sistemi operativi (NTFS, FAT)

Linux file

- “On a UNIX/Linux system, everything is a file; if something is not a file, it is a process”
- Infatti, oltre ai file “regolari” (programmi, testo, immagini, ...), ci sono i file “speciali” che possono rappresentare dispositivi di I/O, canali di comunicazione quali pipe e socket, aree di memoria del kernel, ...
- Un *file regolare* è una sequenza di byte (non esiste una struttura di record)
- Il SO non interpreta la struttura e il contenuto di un file; ciò è delegato alle varie applicazioni
- Una *directory* è un file contenente una lista di nomi di altri file e/o di altre directory (dette subdirectory) contenuti in essa
- Il file system di Linux è organizzato gerarchicamente ad albero:
 - la directory root (indicata con “/”) costituisce la radice dell'albero
 - le directory costituiscono i nodi dell'albero
 - i file costituiscono le foglie

Tipi di file (I)

- ♦ Un file Linux può essere dei seguenti tipi (il primo carattere è quello utilizzato dal comando `ls` per indicare il tipo di file):
 - **file regolare**: dati in formato testo o binario, eseguibili, programmi sorgente, immagini, ...
 - d directory**: file che contiene una lista di altri file
 - l link simbolico**: file che *punta* ad un altro file
 - c device a caratteri**: dispositivo di I/O organizzato a singoli byte
 - b device a blocchi**: dispositivo di I/O organizzato a blocchi di lunghezza fissa (bufferizzato)
 - p named pipe**: meccanismo per l'IPC all'interno del kernel
 - s socket**: canale per le comunicazioni su rete

Tipi di file (II)

- L'opzione `-l` del comando `ls` mostra il tipo di file, utilizzando il primo carattere di ciascuna linea di output:

```
[penguin@antarctic ~]$ ls -l
total 160
drwxr-xr-x   2 penguin birds 4096 Aug 13 23:07 Desktop
drwxrwxr-x   9 penguin birds 4096 Aug 17 23:11 LSO1
-rwxrwxr-x   1 penguin birds 6240 Aug 31 09:35 main
-rw-rw-r--   1 penguin birds  337 Aug 31 09:35 main.c
lrwxrwxrwx   1 penguin birds    6 Aug 31 00:04 prog.c -> main.c
prw-rw-r--   1 penguin birds    0 Aug 31 17:40 myfifo
```

Nomi di file (I)

- Un nome di file può essere di qualsiasi lunghezza e può utilizzare qualsiasi carattere stampabile, eccetto il *forward slash* '/'
- I nomi dei file devono essere unici all'interno di una stessa directory
- Un nome di file che inizia per '.' denota un file/directory nascosto/a (*hidden*) e non viene mostrato dal comando `ls` (a meno di non usare l'opzione `-a`)
- Un nome di file non dovrebbe mai iniziare con *hyphen* '-', in quanto molti comandi considerano una stringa che inizia per '-' come indicazione di un'opzione
- Un nome di file/directory non ha una struttura definita: eventuali estensioni (.c, .h, .txt, ...) hanno solo un valore convenzionale e non intrinseco

Nomi di file (II)

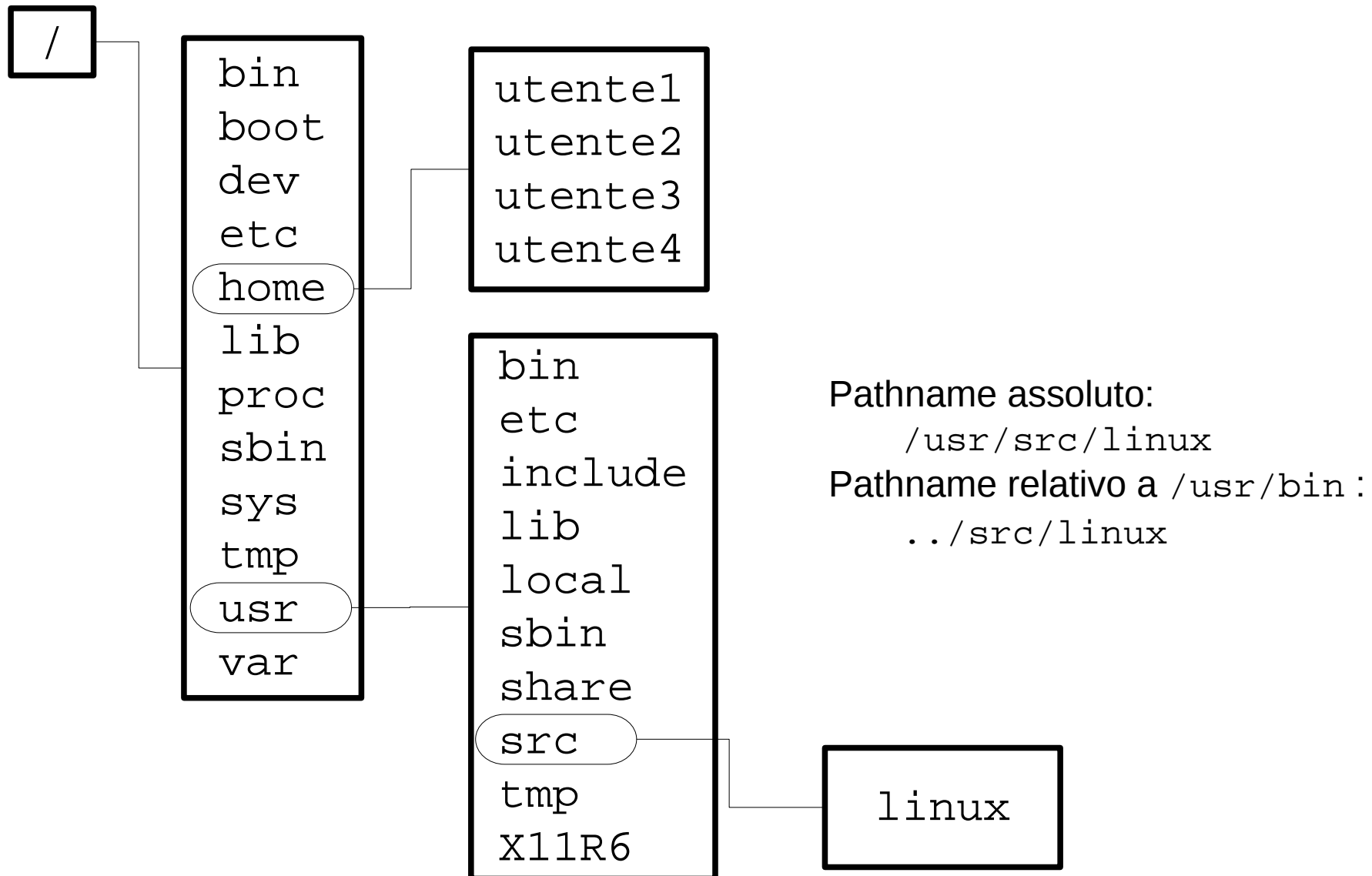
- Se un nome di file contiene caratteri speciali quali `&`, `*`, `\`, `$`, `?`, `<spazio>`, e deve essere usato su linea di comando della shell, bisogna usare degli accorgimenti mediante l'utilizzo dei caratteri di escape `\` e il quoting `'` per evitare che la shell interpreti ed espanda i caratteri speciali

```
[penguin@antarctic DUMMY]$ ls -l
total 24
-rw-rw-r-- 1 verola verola 337 Sep  1 11:14 filename with spaces
-rw-rw-r-- 1 verola verola   4 Sep  1 11:18 fileWith$Dollar
-rw-rw-r-- 1 verola verola  38 Sep  1 11:17 fileWithWildcard*.c
[penguin@antarctic DUMMY]$ file filename with spaces
filename: ERROR: cannot open `filename' (No such file or directory)
with:     ERROR: cannot open `with' (No such file or directory)
spaces:   ERROR: cannot open `spaces' (No such file or directory)
[penguin@antarctic DUMMY]$ file 'filename with spaces'
filename with spaces: ASCII C program text
[verola@localhost DUMMY]$ file fileWith$Dollar
fileWith: ERROR: cannot open `fileWith' (No such file or directory)
[verola@localhost DUMMY]$ file fileWith\$Dollar
fileWith$Dollar: ASCII text
[verola@localhost DUMMY]$
```

Pathname

- ♦ Per identificare un file o una directory si utilizza un nome di percorso (*pathname*) composto da nomi di directory separati dal carattere '/'; l'ultimo nome è quello della directory o del file in questione
- ♦ Il pathname può essere:
 - ♦ *assoluto*: inizia per '/' e quindi si riferisce alla root del filesystem
 - ♦ *relativo*: inizia con un nome di directory o è composto dal solo nome del file da indirizzare e la ricerca inizia a partire dalla *directory attiva o di lavoro (working directory)* del processo in esecuzione
- ♦ Il simbolo '.' indica la directory attiva, mentre '..' indica la directory che contiene la directory attiva (*parent directory*)

Esempio di filesystem Linux



Directory principali sotto Linux

| | |
|--------------|--|
| /bin | Comandi base (<i>utilities</i>) per gli utenti |
| /boot | File immagine del kernel |
| /dev | Device file per accedere a periferiche o sistemi di memorizzazione |
| /etc | File di configurazione del sistema |
| /home | "Home directory" degli utenti |
| /lib | Librerie condivise dai programmi e utili per il loro funzionamento |
| /proc | File system virtuale senza reale allocazione su disco. Viene utilizzato per fornire informazioni di sistema, in particolare relative al kernel |
| /sbin | Comandi per la gestione del sistema, destinati al <i>system administrator</i> |
| /sys | File relativi ad informazioni sull'HW di sistema |
| /tmp | Directory dei file temporanei |
| /usr | Directory contenente gran parte dei programmi esistenti nel sistema |
| /var | Dati variabili, code di stampa |

Comandi relativi al filesystem

- ♦ **pwd**
 - ♦ Print Working Directory
- ♦ **cd *directory***
 - ♦ Change working Directory (“go to” directory)
- ♦ **mkdir *directory***
 - ♦ MaKe a directory
- ♦ **rmdir *directory***
 - ♦ ReMove directory
- ♦ **ls *directory***
 - ♦ LiSt directory content
 - ♦ **-a** all files
 - ♦ **-l** long listing
 - ♦ **-g** group information
- ♦ **tree *directory***
 - ♦ display directory tree

Gestione dei file

- **rm**
 - ReMove (delete) files
- **cp**
 - CoPy files
- **mv**
 - MoVe (or rename) file
- **ln**
 - LiNk creation (symbolic or not)
- **more, less**
 - page through a text file
- **df [options] [directory]**
 - mostra lo spazio libero nei dischi
 - > **df -Tm**
- **du [options] [directory]**
 - > **du**
 - > **du directory**
 - > **du -s directory**
 - > **du -k directory**

Esercizi

1. Cambiare directory di lavoro, verificare che il comando abbia avuto effetto e poi tornare alla propria home directory.
2. Verificare i file presenti nella directory di lavoro e in `/etc`.
3. Trovare le opzioni del comando `ls` per:
 - a) visualizzare i *file nascosti*
 - b) output ricorsivo (lista contenuto tutte subdirectory)
 - c) listare i file in ordine alfabetico inverso
 - d) stampare l'inode dei file
4. Creare nella propria home directory una nuova directory `test`.
5. Trovare opzione di `cp` per copiare ricorsivamente un albero di directory.
6. Trovare opzione di `rm` che richiede la conferma della cancellazione del file.
7. Rimuovere un file che inizia per '-' (as es. `-file`).
8. Trovare opzione di `mkdir` che permette di creare la directory e le eventuali parent directory, se non esistenti.

Il filesystem nella realtà

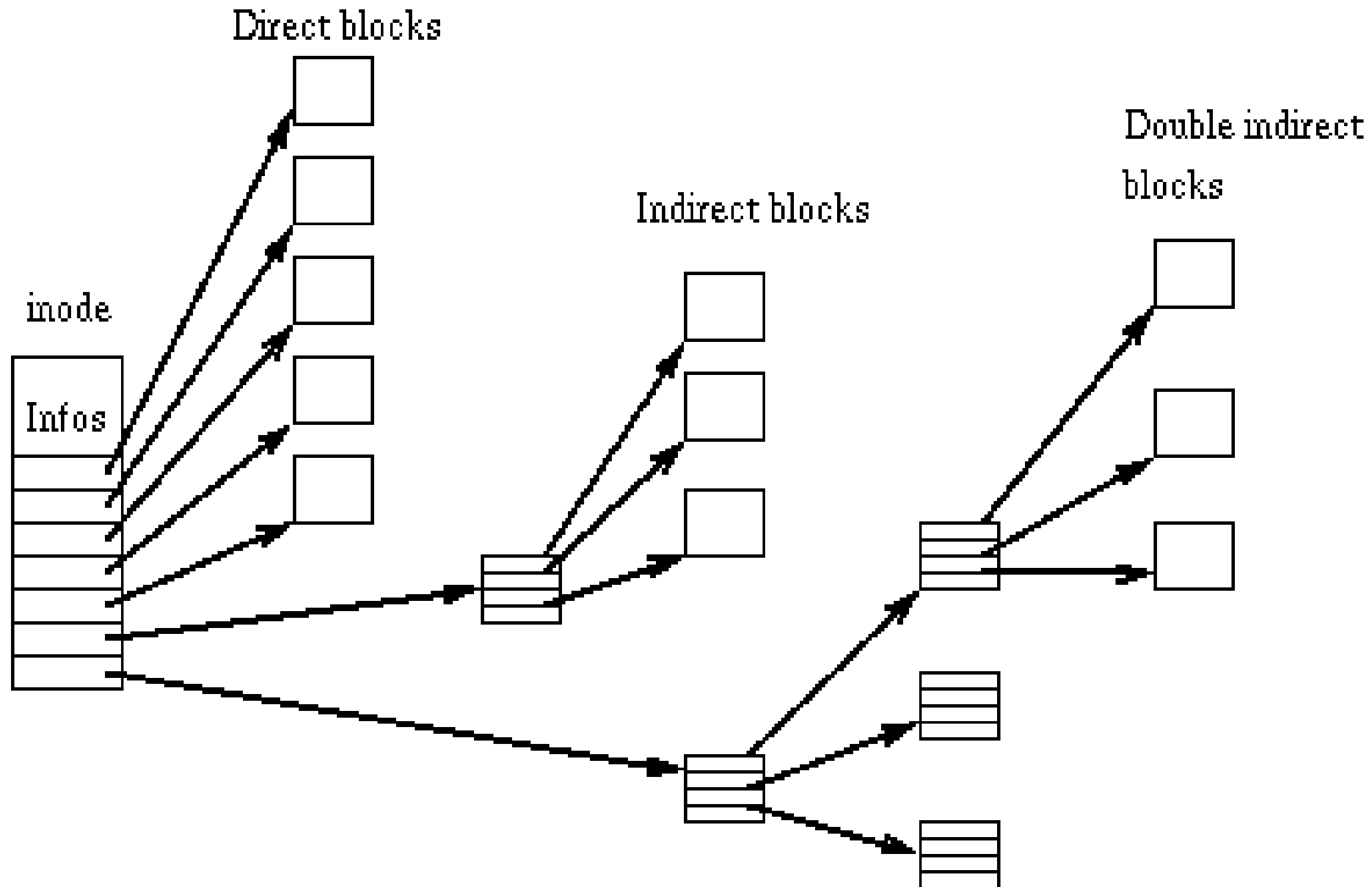
- ♦ Anche se per l'utente i file e le directory sono organizzate secondo una struttura ordinata ad albero, il computer in realtà non conosce il significato di tale rappresentazione
- ♦ Il disco è diviso in partizioni ed ogni partizione ha un suo filesystem; l'insieme dei filesystem forma la struttura ad albero con cui l'utente interagisce
- ♦ I nomi dei file sono utili per l'utente, ma il SO identifica i file mediante i loro *inode number*
- ♦ Un *inode number* è un identificatore univoco associato ad una struttura di dati (*inode*) contenente informazioni sulle proprietà del file e sulla locazione fisica dei dati che costituiscono il file stesso
- ♦ Ogni partizione ha il suo insieme di *inode*: lo spazio per essi viene generalmente creato durante il processo di installazione del sistema

L'inode

- ♦ Ad ogni file creato viene assegnato un *inode* con le seguenti informazioni:
 - ♦ lunghezza del file in bytes
 - ♦ ID del dispositivo contenente il file
 - ♦ user ID del proprietario del file e group ID del file
 - ♦ *inode number*
 - ♦ *file mode*, che determina il tipo e i permessi di accesso e esecuzione del file
 - ♦ data di ultima modifica (*mtime*), di ultimo accesso (*atime*) e di ultima modifica dell'*inode* (*ctime*)
 - ♦ *link number*, che indica il numero di *hard-link* collegati all'*inode*

inode e blocchi su disco

Esempio di struttura di *inode* (ext2 filesystem)



Attributi dei file

- ♦ Per ottenere informazioni complete su un file:

```
% ls -lis prova.txt
```

```
72670 8 -rwxr--r-- 1 penguin birds 213 Oct 2 00:12  
prova.txt
```

- ♦ Descrizione dei campi:
 - ♦ Indice dell'*inode* del file
 - ♦ Numero di blocchi utilizzati dal file (1 blocco = 1024 bytes)
 - ♦ Tipo e permessi del file
 - ♦ Il conteggio di hard-link
 - ♦ Username e groupname del possessore del file
 - ♦ Dimensione in byte
 - ♦ Data di ultima modifica
 - ♦ Nome del file
- ♦ Si noti che questi sono gran parte dei dati contenuti in un *inode*

Concetto di owner e gruppo

- ◆ **Ogni file è associato a:**
 - ◆ un utente proprietario del file
 - ◆ un gruppo (i.e., insieme di utenti) con speciali diritti sul file
- ◆ **Come identificare utenti e gruppi:**
 - ◆ *user id* (valore intero, Unix internals); *username* (stringa)
 - ◆ *group id* (valore intero, Unix internals); *groupname* (stringa)
- ◆ **Come associare un utente ad un file:**
 - ◆ quando create un file, viene associato al vostro user id
 - ◆ potete modificare il proprietario tramite **chown newUserId file(s)**
 - ◆ normalmente non disponibile in un sistema in cui vengono gestite system quotas

Gestione dei gruppi

- ♦ **Come ottenere la lista dei gruppi a cui appartenete**
groups [username]
invocata senza argomenti, elenca i gruppi a cui appartenete;
indicando **username**, ritorna i gruppi associati a **username**
- ♦ **Come associare un gruppo ad un file**
 - ♦ quando create un file, viene associato al vostro gruppo corrente
 - ♦ il vostro gruppo corrente iniziale è scelto dall'amministratore
 - ♦ potete cambiare il vostro gruppo corrente (aprendo una nuova shell) tramite **newgrp <groupname>**
 - ♦ potete modificare il gruppo associato ad un file tramite il comando **chgrp <groupname> <file(s)>**

Permessi dei file (I)

Ogni file è associato a 9 flag chiamati “Permission bits”

| User | | | Group | | | Others | | |
|------|---|---|-------|---|---|--------|---|---|
| R | W | X | R | W | X | R | W | X |

- ♦ *Read:*
 - ♦ file regolari: possibilità di leggere il contenuto
 - ♦ directory: possibilità di leggere l'elenco dei file contenuti in una directory
- ♦ *Write:*
 - ♦ file regolari: possibilità di modificare il contenuto
 - ♦ directory: possibilità di aggiungere, rimuovere, rinominare file
- ♦ *Execute:*
 - ♦ file regolari: possibilità di eseguire il file (se ha senso)
 - ♦ directory: possibilità di fare `cd` nella directory o accedervi tramite path

Permessi dei file (II)

- ♦ **In realtà:**
 - ♦ la gestione dei permessi è leggermente più complessa di quanto presentato
- ♦ **Quando un processo è in esecuzione, possiede:**
 - ♦ Un user ID / group ID reale (usato per accounting)
 - ♦ Un user ID / group ID effettivo (usato per accesso)
- ♦ **Quali permission vengono utilizzati?**
 - ♦ Se l'user ID effettivo corrisponde a quello del possessore del file, si applicano le User permission
 - ♦ Altrimenti, se il group ID effettivo corrisponde a quello del file, si applicano le Group permission
 - ♦ Altrimenti, si applicano le Others permission

Come cambiare i permessi (I)

Relativo: `chmod [ugoa][+ -=][rwxXstugo] file(s)`

- ♦ Esempi:
 - ♦ `chmod u+x script.sh`
Aggiunge il diritto di esecuzione per il proprietario per il file `script.sh`
 - ♦ `chmod -R ug+rwX src/*`
Aggiunge il diritto di scrittura, lettura per il proprietario e il gruppo per i file e contenuti in `src/`, ricorsivamente. Inoltre aggiunge il diritto di esecuzione per le directory
 - ♦ `chmod -R o-rwx $HOME`
Toglie tutti i diritti a tutti gli utenti che non sono il proprietario e non appartengono al gruppo, ricorsivamente
- ♦ Nota:
Consultate `info chmod` per maggiori dettagli

Come cambiare i permessi (II)

Assoluto: `chmod octal-number file(s)`

| User | | | Group | | | Others | | |
|------|---|---|-------|---|---|--------|---|---|
| R | W | X | R | W | X | R | W | X |
| 4 | 2 | 1 | 4 | 2 | 1 | 4 | 2 | 1 |

Esempi:

- **`chmod 755 public_html`**

Assegna diritti di scrittura, lettura e esecuzione all'utente, diritti di lettura e esecuzione al gruppo e agli utenti

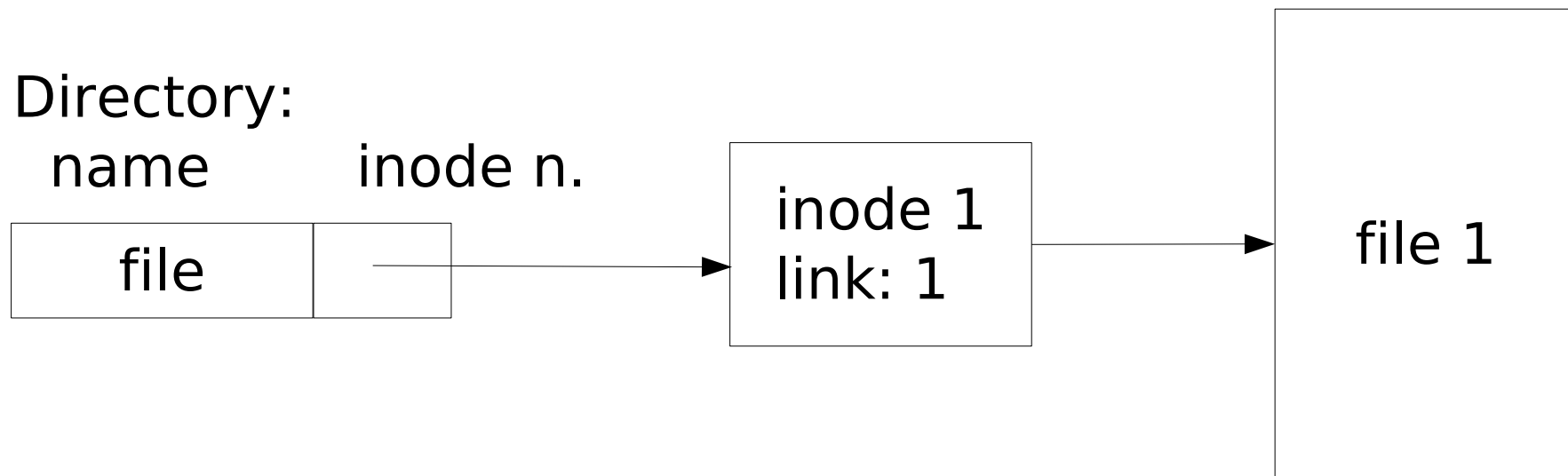
- **`chmod 644 .procmailrc`**

Assegna diritti di scrittura, lettura all'utente, diritti di lettura al gruppo e agli altri

Link

- ♦ **ln *file* *hlink***
 - ♦ E' un hard-link, cioè crea una entry (nella directory corrente) chiamata *hlink* con lo stesso *inode number* di *file*
 - ♦ Il *link number* dell'inode di *file* viene incrementato di 1
- ♦ **ln -s *file* *slink***
 - ♦ E' un link simbolico, cioè crea un file speciale (nella directory corrente) chiamato *slink* che "punta" alla entry nella directory di nome *file*
 - ♦ Il link number dell'inode di *file* non viene incrementato
- ♦ Se cancello *file*:
 - ♦ **hard-link**: il *link number* dell'inode viene decrementato, ma i dati del file non vengono rimossi dal disco, fintanto che il *link number* non diventa uguale a 0.
 - ♦ **link simbolico**: il link diviene "*stale*", ovvero punta ad un file inesistente.

Esempio - Situazione iniziale

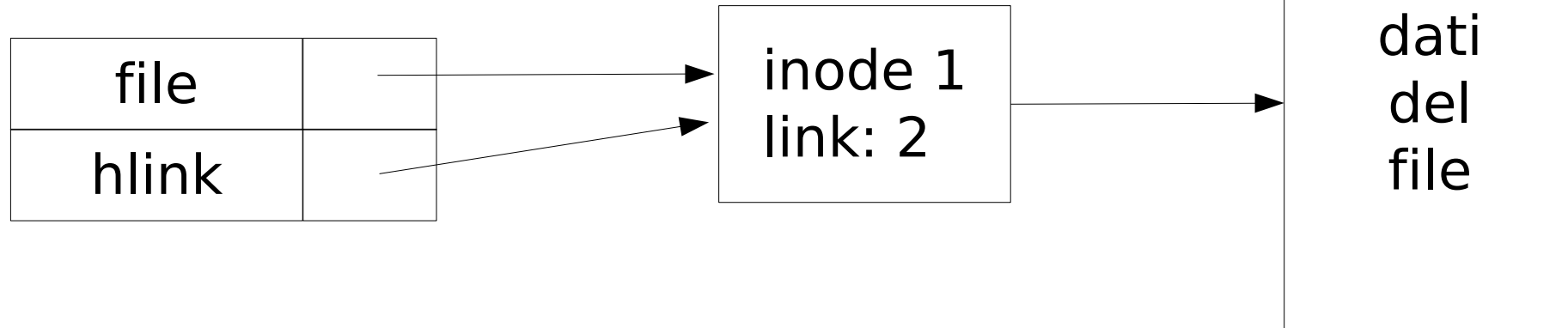


Esiste un file di nome `file` con *inode number*=1 e *link number*=1
(cioè una sola entry nella directory si riferisce all'*inode* del file)

Esempio - Creazione di un hard-link

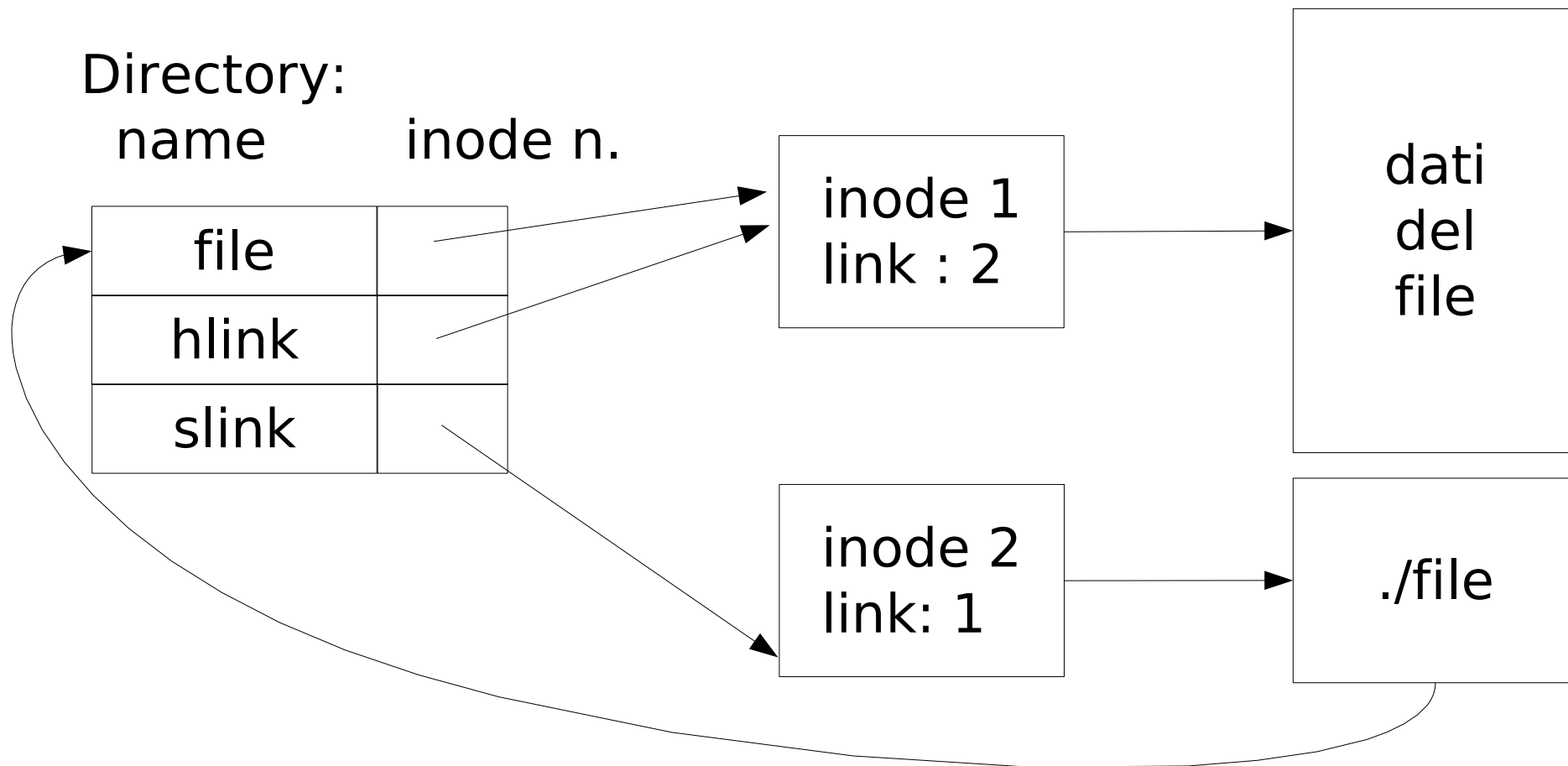
Directory:

name inode n.



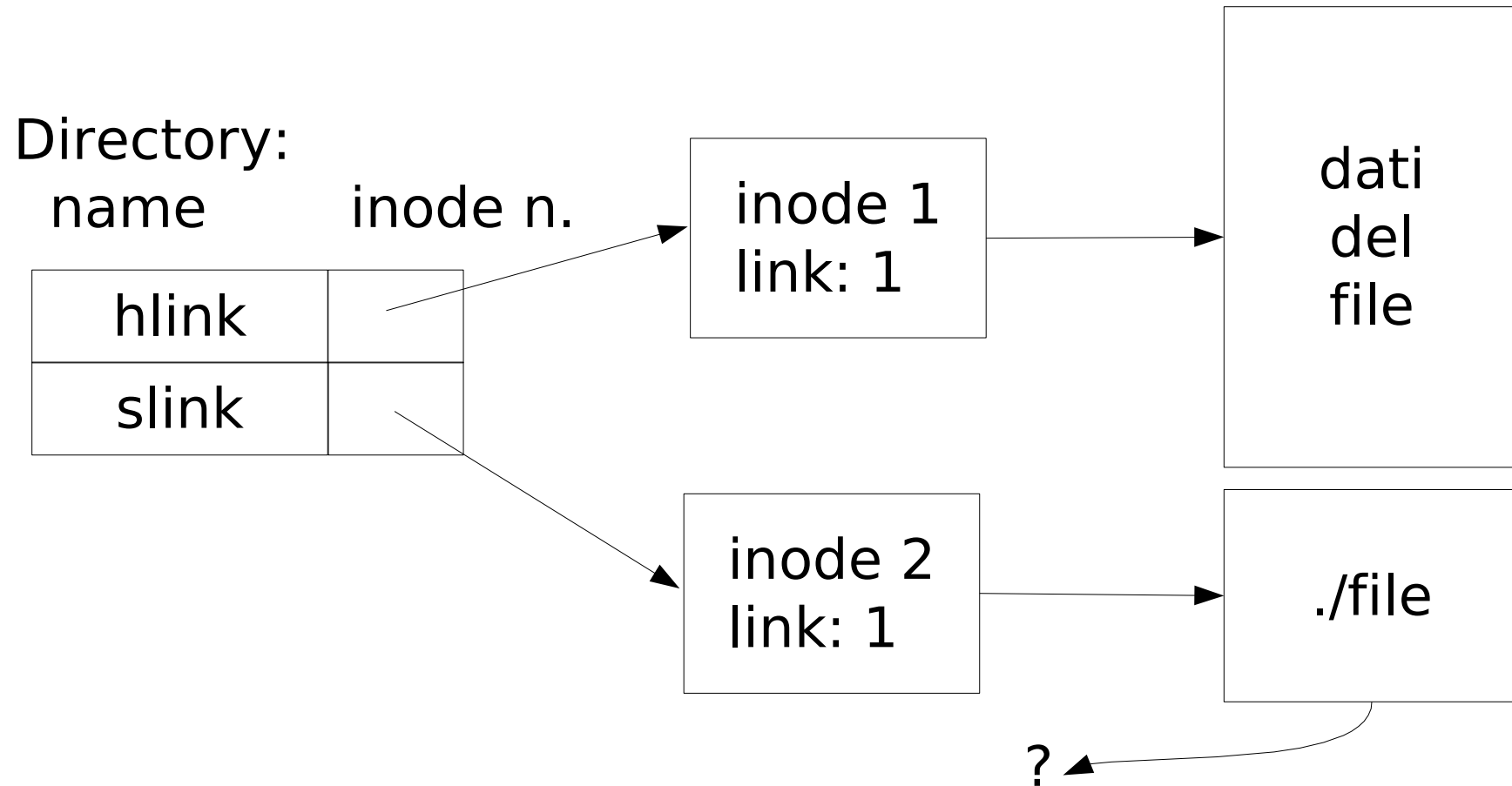
Viene creato un (secondo) *hard link* di nome **hlink** che si riferisce all'*inode* 1, quindi il *link number* dell'*inode* diventa 2 (cioè 2 entry nella directory si riferiscono all'*inode* del file)

Esempio - Creazione di un link simbolico



Viene creato un *link simbolico* di nome **slink** che **punta** al file con *inode* 1, quindi **slink** si riferisce a un nuovo inode 2 con *link number*=1; il contenuto del file **slink** è il pathname del file puntato

Esempio - Rimozione del file originale



Se viene rimossa la entry `file` dalla directory, il link simbolico `slink` pur rimanendo un file esistente, diventa *stale*

Esercizi

1. Utilizzare `ls` per visualizzare gli attributi del file, inode incluso.
2. Qual è la rappresentazione ottale della maschere di permessi `-rwxr-----` ? E la rappresentazione simbolica della maschere di permessi ottale `755` ?
3. Rendere una directory non eseguibile, ma leggibile, e fare `cd` nella dir. Cosa si ottiene? E con `ls dir`?
4. Rendere una directory non leggibile, ma eseguibile, e fare `cd` nella dir. Cosa si ottiene? E con `ls dir`?
5. Come si può listare i file in ordine cronologico? Cosa fanno le opzioni `-u` e `-c` di `ls`?
6. Analizzare le info fornite dal comando `stat`.
7. Creare un link simbolico `symlink1` che punta a un link simbolico `symlink2` che a sua volta punta a `symlink1` (dipendenza circolare). Cosa succede se si tenta di leggere il contenuto di uno dei 2 file?
8. Creare una sottodirectory `bin` all'interno della propria home directory in cui mettere file eseguibili e script. Fare un hard-link al file del comando `compress` ed un link simbolico al file del comando `date`.

Attributi dei processi

- ♦ **pid**: identificatore del processo
- ♦ **ppid**: parent pid (identificatore del processo padre)
- ♦ **nice number**: priorità statica del processo; può essere cambiata (diminuita) con il comando **nice**
- ♦ **TTY**: terminal device associato al processo
- ♦ **real, effective user id**
real, effective group id
Identificatori dell'owner e del group owner del processo
- ♦ altro: memoria utilizzata, cpu utilizzata, etc.

Monitoraggio dei processi

Il comando `ps` riporta lo stato dei processi attivi nel sistema (vedi `man ps` per il significato delle varie colonne)

```
$ ps
PID TTY TIME CMD
648 pts/2 00:00:00 bash

$ ps alx
 F  UID  PID  PPID  PRI  NI   VSZ  RSS  WCHAN  STAT  TTY      TIME  COMMAND
 0    0    1    0   15   0   500  244  1207b7  S     ?        0:05  init
 0    0    2    1   15   0    0    0  124a05  SW    ?        0:00  [keventd]
 0    0    3    1   34  19    0    0  11d0be  SWN   ?        0:00  [ksoftirqd_CPU0]
 0    0    4    1   25   0    0    0  135409  SW    ?        0:00  [kswapd]
 0    0    5    1   25   0    0    0  140f23  SW    ?        0:00  [bdflush]
 0    0    6    1   15   0    0    0  1207b7  SW    ?        0:00  [kupdated]
 0    0    7    1   25   0    0    0  15115f  SW    ?        0:00  [kinoded]
 0    0    9    1   19   0    0    0  23469f  SW    ?        0:00  [mdrecoveryd]
 0    0   12    1   15   0    0    0  1207b7  SW    ?        0:00  [kreiserfsd]
 0    0  150    1    0  -20    0    0  107713  SW<   ?        0:00  [lvm-mpd]
```

Priorità e terminazione dei processi

- ♦ Comando **nice**
 - ♦ esegue un comando con una priorità statica diversa da quella di default (nel range da -20=massima priorità a 19=minima priorità, 0=default, per dettagli vedi **info nice**)

```
nice -n 19 command
```

- ♦ Comando **renice**
 - ♦ cambia la priorità di un processo mentre è in esecuzione

```
renice [+ -]value -p pid
```

- ♦ Comando **kill**
 - ♦ manda un segnale a un processo; alcuni segnali possono provocare la terminazione del processo

```
kill -9 pid
```

Lancio e controllo dei processi

- ◆ **Processi in foreground**
 - ◆ Processi che "controllano" il terminale da cui sono stati lanciati
 - ◆ In ogni istante, un solo processo è in foreground
- ◆ **Processi in background**
 - ◆ Vengono eseguiti senza "controllare" il terminale a cui sono "attaccati"
- ◆ **Job control**
 - ◆ Permette di portare i processi da background a foreground e viceversa
- ◆ **&** Lancia un processo direttamente in background
Esempio: **long_cmd &**
- ◆ **Ctrl+z** Ferma (stop) il processo in foreground
- ◆ **jobs** Lista i processi in background
- ◆ **%n** Si riferisce al processo n-esimo in background
Esempio: **kill %1**
- ◆ **fg** Porta un processo da background a foreground
Esempio: **fg %1**
- ◆ **bg** Fa ripartire in background i processi fermati

Una lista più completa di tasti di controllo `Ctrl+<x>` e caratteri speciali con il loro significato/azione verrà presentata nel modulo sulla Shell.

Esercizi

1. Esaminare e provare le varie opzioni di `ps`.
2. Utilizzare l'utility per un ps periodico: `top`.
3. Fare test di lancio processi in foreground e background.
4. Verificare l'effetto dei vari interrupt `Ctrl+c`, `Ctrl+\`, `Ctrl+z` su processi in foreground e in background.
5. Trovare l'opzione di `kill` per listare i segnali supportati.
6. Trovare il modo di lanciare un processo (ad es. `sleep 30`) in foreground, sospenderlo, listarlo nella lista dei job, riattivarlo in background, sospenderlo di nuovo, riattivarlo in foreground.
7. Trovare il modo di verificare l'esistenza di un processo di cui e' noto il PID, utilizzando `kill`, ma senza terminarlo.
8. Abbassare la priorit  di un processo alla minima possibile.