Operazione Rif. PA 2022-17295/RER approvata con DGR 1379/2022 del 01/08/2022 finanziata con risorse del Programma Fondo sociale europeo Plus 2021-2027 della Regione Emilia –Romagna.

Progetto n. **1** - Edizione n. **1**

MODULO: N. 6

Titolo: SICUREZZA DEI SISTEMI INFORMATICI

DOCENTE: MARCO PRANDINI

Parte 4 - Sistemi operativi e controllo dell'accesso

Introduzione ai concetti di base di Unix

MATERIALE TRATTO DALLE DISPENSE DI

Sistemi Operativi L-A

Università di Bologna

CdS: Laurea Triennale in Ingegneria Informatica

A.A. 2007-2008

Autore: Prof. Paolo Bellavista

La versione completa è disponibile all'indirizzo:

`http://lia.deis.unibo.it/Courses/sola0708-info/`

Struttura dei SO

Quali sono le ***componenti*** di un SO?

gestione dei ***processi***

gestione della ***memoria centrale***

gestione di ***memoria secondaria*** e ***file system***

gestione dell'***I/O***

protezione e sicurezza

interfaccia utente/programmatore

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

Processi

Processo = programma in esecuzione

il ***programma*** è ***un'entità passiva*** (un insieme di byte contenente le istruzioni che dovranno essere eseguite)

il processo è un'entità attiva:

- è ***l'unità di lavoro/esecuzione*** all'interno del sistema. ***Ogni attività all'interno del SO è rappresentata da un processo***
- ***è l'istanza di un programma in esecuzione***

Processo = programma +
contesto di esecuzione (PC, registri, …)

Gestione dei processi

In un sistema multiprogrammato: più processi possono essere ***simultaneamente presenti*** nel sistema

Compito cruciale del SO

- ***creazione/terminazione*** dei processi
- ***sospensione/ripristino*** dei processi
- ***sincronizzazione/comunicazione*** dei processi
- ***gestione del blocco critico (deadlock)*** di processi

Gestione della memoria centrale

HW di sistema di elaborazione è equipaggiato con ***un unico spazio di memoria*** accessibile direttamente da CPU e dispositivi

Compito cruciale di SO

separare gli spazi di indirizzi associati ai processi

allocare/deallocare memoria ai processi

memoria virtuale **-** gestire ***spazi logici di indirizzi*** di dimensioni complessivamente ***superiori allo spazio fisico***

realizzare i collegamenti (***binding***) tra ***memoria logica e memoria fisica***

Gestione dei dispositivi di I/O

Gestione dell'I/O rappresenta una parte importante di SO:

interfaccia tra programmi e dispositivi

per ogni dispositivo: ***device driver***

routine per l'interazione con un particolare dispositivo

contiene ***conoscenza specifica*** sul dispositivo (ad es., routine di gestione delle interruzioni)

Gestione della memoria secondaria

Tra tutti i dispositivi, la ***memoria secondaria*** riveste un ruolo particolarmente importante**:**

allocazione/deallocazione di spazio

gestione dello ***spazio libero***

scheduling delle operazioni sul disco

Di solito:

la **gestione dei file** usa i meccanismi di gestione della memoria secondaria

la **gestione della memoria secondaria** è indipendente dalla gestione dei file

Gestione del file system

Ogni sistema di elaborazione dispone di uno o più dispositivi per la memorizzazione persistente delle informazioni (***memoria secondaria***)

Compito di SO

fornire una ***visione logica uniforme della memoria secondaria*** (indipendente dal tipo e dal numero dei dispositivi):

realizzare il ***concetto astratto di file***, come unità di memorizzazione logica

fornire una struttura astratta per ***l'organizzazione*** dei file (***direttorio***)

Gestione del file system

Inoltre, SO si deve occupare di:

creazione/cancellazione di file e direttori

manipolazione di file/direttori

associazione tra file e dispositivi di memorizzazione secondaria

Spesso file, direttori e dispositivi di I/O vengono ***presentati*** a utenti/programmi ***in modo uniforme***

Protezione e sicurezza

In un sistema multiprogrammato, più entità (processi o utenti) possono utilizzare le risorse del sistema contemporaneamente: ***necessità di protezione***

Protezione: controllo dell'accesso alle risorse del sistema da parte di processi (e utenti) mediante

autorizzazioni

modalità di accesso

Risorse da proteggere:

memoria

processi

file

dispositivi

Protezione e sicurezza

Sicurezza:

se il sistema appartiene a una rete, la ***sicurezza misura l'affidabilità del sistema nei confronti di accessi (attacchi) dal mondo esterno***

Non ce ne occuperemo all'interno di questo corso…

Interfaccia utente

SO presenta un'interfaccia che consente l'interazione con l'utente

- **interprete comandi** (***shell***): l'interazione avviene mediante una linea di comando
- **interfaccia grafica** (graphical user interface, ***GUI***): l'interazione avviene mediante ***interazione*** mouse-elementi grafici su desktop; di solito è organizzata a finestre

Interfaccia programmatore

L'interfaccia del SO verso i processi è rappresentato dalle ***system call***:

- mediante la system call il ***processo richiede a SO*** l'esecuzione di un servizio
- la system call esegue ***istruzioni privilegiate***: passaggio da modo ***user*** a modo ***kernel***

Classi di system call:

- gestione dei processi
- gestione di file e di dispositivi (spesso trattati in modo omogeneo)
- gestione informazioni di sistema
- comunicazione/sincronizzazione tra processi

Programma di sistema **= programma che chiama system call**

Struttura e organizzazione di SO

Sistema operativo = insieme di componenti

- **gestione dei processi**
- **gestione della memoria centrale**
- **gestione dei file**
- **gestione dell'I/O**
- **gestione della memoria secondaria**
- **protezione e sicurezza**
- **interfaccia utente/programmatore**

Le componenti non sono indipendenti tra loro, ma interagiscono

Struttura e organizzazione di SO

Visto che le varie componenti di un SO sono interagenti, come sono organizzate nella struttura di un SO?

Vari approcci

struttura *monolitica*

struttura *modulare: stratificazione*

microkernel

Struttura monolitica

SO è costituito da un ***unico modulo*** contenente un **insieme di procedure**, che realizzano le varie componenti:

- l'interazione tra le componenti avviene mediante il meccanismo di chiamata a procedura

Ad esempio:

MS-DOS

prime versioni di UNIX

SO monolitici

Principale vantaggio: basso costo di interazione tra le componenti -> efficienza

Svantaggio: SO è un sistema complesso e presenta gli stessi requisiti delle applicazioni ***in-the-large***

estendibilità
manutenibilità
riutilizzo
portabilità
affidabilità
...

Soluzione: organizzazione **modulare**

Struttura modulare

Le varie componenti del SO vengono organizzate in ***moduli caratterizzati da interfacce ben definite***

Sistemi stratificati (a livelli)

(THE, Dijkstra1968)

SO è costituito da ***livelli sovrapposti***, ognuno dei quali realizza un insieme di funzionalità:

- ogni livello realizza un insieme di funzionalità che vengono ***offerte al livello superiore*** mediante un'interfaccia
- ogni livello ***utilizza le funzionalità offerte dal livello sottostante***, per realizzare altre funzionalità

Struttura a livelli

Ad esempio: **THE (5 livelli)**

livello 5: programmi di utente
livello 4: buffering dei dispositivi di I/O
livello 3: driver della console
livello 2: gestione della memoria
livello 1: scheduling CPU
livello 0: hardware

Struttura a livelli

Vantaggi

Astrazione: ogni livello è un oggetto astratto, che fornisce ai livelli superiori una visione astratta del sistema (***macchina virtuale***), limitata alle astrazioni presentate nell'interfaccia

Modularità: relazioni tra livelli sono chiaramente esplicitate dalle interfacce possibilità di sviluppo, verifica, modifica ***in modo indipendente*** dagli altri livelli

Svantaggi

organizzazione gerarchica tra le componenti: non sempre è possibile -> difficoltà di realizzazione

scarsa efficienza (costo di attraversamento dei livelli)

Soluzione: limitare il numero dei livelli

Nucleo (kernel) di SO

È la parte di SO che esegue ***in modo privilegiato*** (modo ***kernel***)

È la parte ***più interna*** di SO che si interfaccia direttamente con l'hardware della macchina

Le funzioni realizzate all'interno del nucleo variano a seconda del particolare SO

Nucleo (kernel) di SO

Per un sistema multiprogrammato a divisione di tempo, il nucleo deve, almeno:

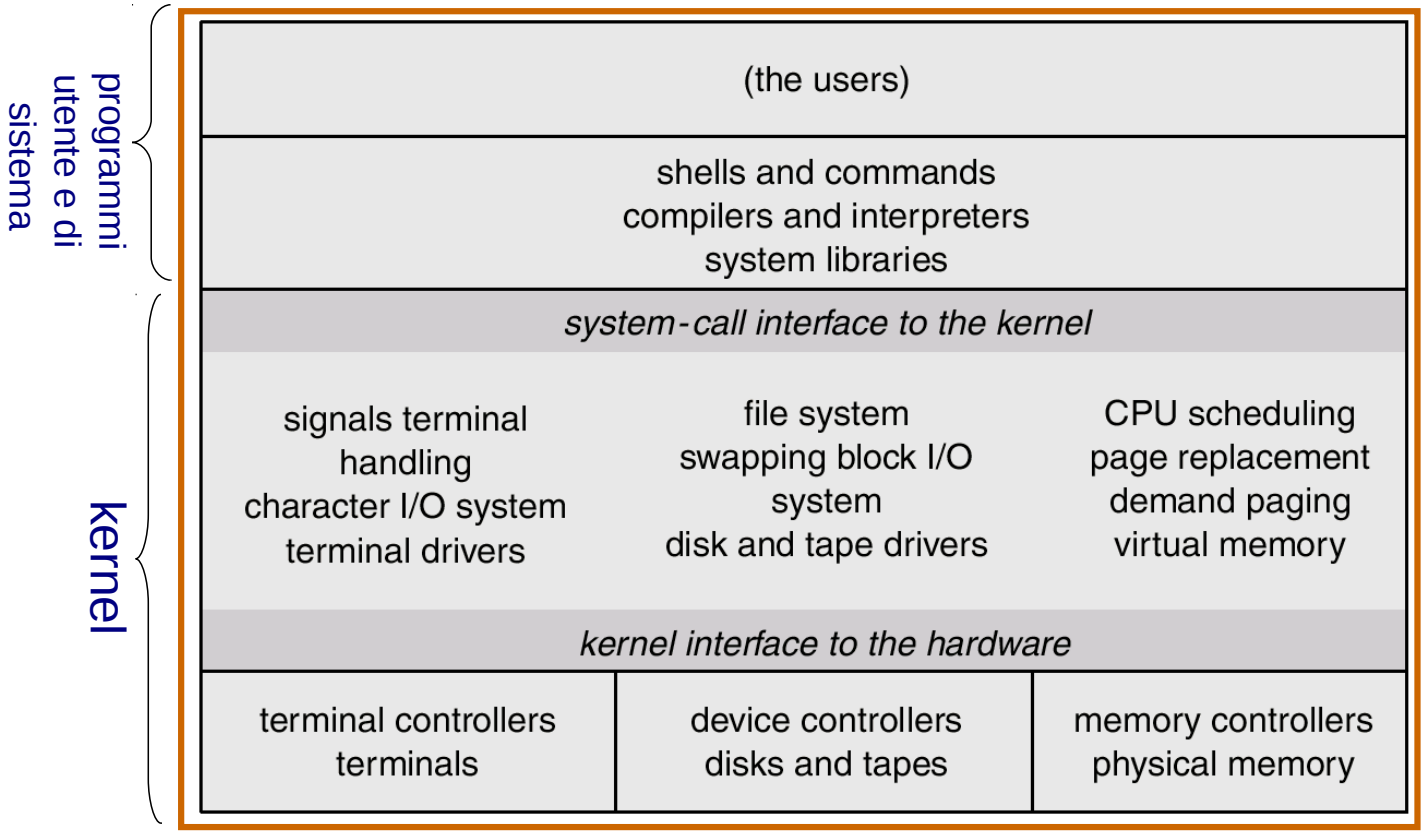
- gestire il ***salvataggio/ripristino dei contesti (context-switching)***
- realizzare lo ***scheduling della CPU***
- gestire le ***interruzioni***
- realizzare il meccanismo di ***chiamata a system call***

Una piccola panoramica: organizzazione di UNIX

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

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

Organizzazione di UNIX



UNIX: principi di progettazione e vantaggi

Progetto ***snello, pulito e modulare***

Scritto in ***linguaggio di alto livello*** (linguaggio C)

Disponibilità codice sorgente

Potenti primitive di SO su una piattaforma a ***basso prezzo***

Progettato per essere ***time-sharing***

User interface semplice (shell), anche sostituibile

File system con ***direttori organizzati ad albero***

Concetto unificante di file, come ***sequenza non strutturata*** di byte

Supporto semplice a ***processi multipli e concorrenza***

Supporto ampio allo ***sviluppo di programmi*** applicativi e/o di ***sistema***

Modularità

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

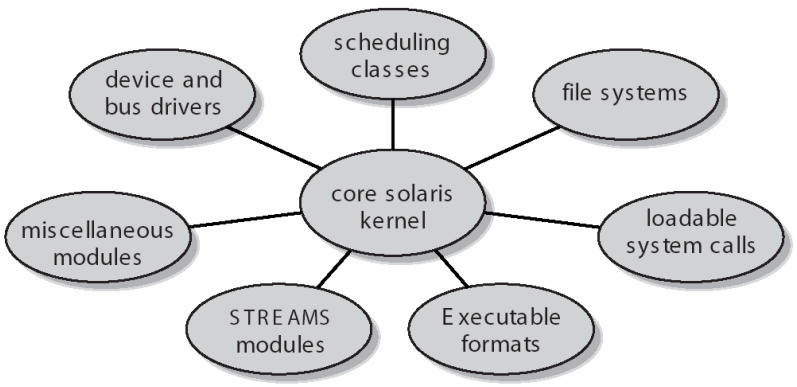
ogni modulo core è ***separato***

ogni modulo interagisce con gli altri tramite ***interfacce note***

ogni modulo può essere ***caricato nel kernel quando*** e ove ***necessario***

possono usare tecniche object-oriented

Strutturazione simile ai livelli, ma con ***maggiore flessibilità***



Esempio di SO Solaris di SUN

Processi e thread

Concetto di processo

Il processo è un programma in esecuzione

È l'**unità di esecuzione** all'interno del SO

Solitamente, esecuzione **sequenziale** (istruzioni vengono eseguite in sequenza, secondo l'ordine specificato nel testo del programma)

SO multiprogrammato consente ***l'esecuzione concorrente di più processi***

D'ora in poi faremo implicitamente riferimento sempre al caso di SO multiprogrammati

Concetto di processo

Programma = entità passiva
Processo = entità attiva

Il processo è rappresentato da:

codice (*text*) del programma eseguito
dati: variabili globali
program counter
alcuni **registri** di CPU
stack: parametri, variabili locali a funzioni/procedure

Concetto di processo

Processo = {PC, registri, stack, text, dati}

Inoltre, a un processo possono essere associate delle ***risorse di SO***. Ad esempio:

file aperti
connessioni di rete
altri dispositivi di I/O in uso
...

Stati di un processo

Un processo, durante la sua esistenza può trovarsi in vari **stati**:

- Init: ***stato transitorio*** durante il quale il processo viene caricato in memoria e SO inizializza i dati che lo rappresentano
- Ready: processo è ***pronto per acquisire la CPU***
- Running: processo ***sta utilizzando la CPU***
- Waiting: processo è ***sospeso in attesa di un evento***
- Terminated: ***stato transitorio*** relativo alla fase di terminazione e deallocazione del processo dalla memoria

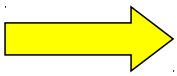
Stati di un processo

Transizioni di stato:

Stati di un processo

In un sistema ***monoprocessore e multiprogrammato***:

- ***un solo processo*** (al massimo) si trova nello ***stato running***
- più processi possono trovarsi negli ***stati ready e waiting***
  - necessità di ***strutture dati per mantenere in memoria le informazioni su processi in attesa***
    - di acquisire la CPU (ready)
    - di eventi (waiting)

 ***Descrittore di processo***

Operazioni sui processi

Ogni SO multiprogrammato prevede dei meccanismi per la gestione dei processi

Meccanismi necessari:

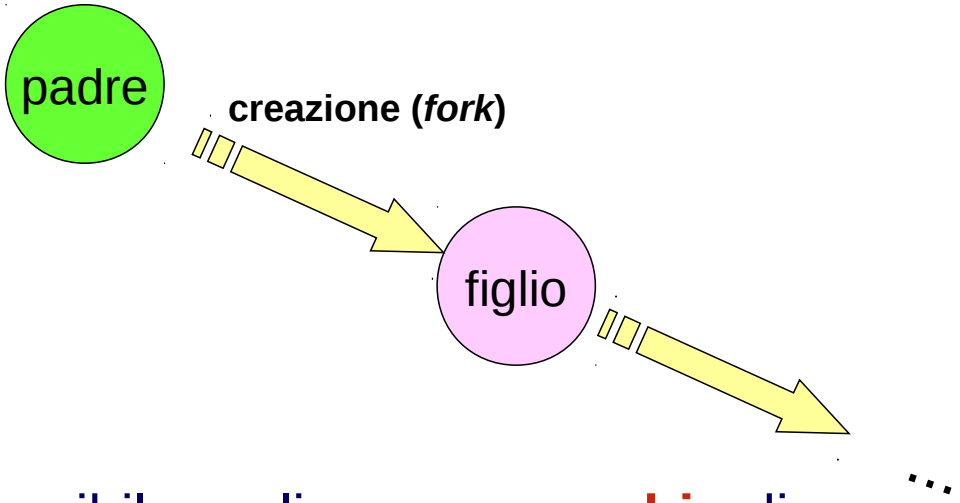
- creazione
- terminazione
- interazione tra processi

Sono operazioni privilegiate (esecuzione in modo kernel)

definizione di **system call**

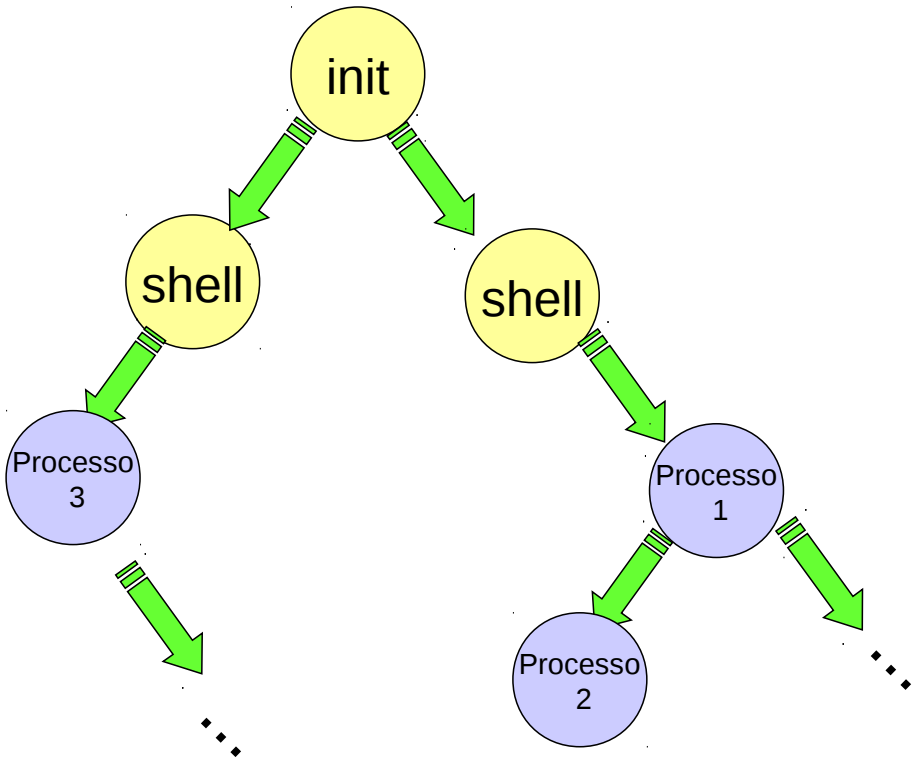
Creazione di processi

Un processo (***padre***) può richiedere la creazione di un nuovo processo (***figlio***)



È possibile realizzare **gerarchie** di processi

Gerarchie di processi (es. UNIX)



Relazione padre-figlio

Vari aspetti/soluzioni:

concorrenza

padre e figlio procedono in ***parallelo*** (es. UNIX), oppure

il padre ***si sospende*** in attesa della terminazione del figlio

condivisione di risorse

le risorse del padre (ad esempio, i file aperti) sono ***condivise*** con i figli (es. UNIX), oppure

il figlio utilizza risorse soltanto se esplicitamente richieste da se stesso

spazio degli indirizzi

duplicato: lo spazio degli indirizzi del figlio è una copia di quello del padre (es. fork() in UNIX), oppure

differenziato: spazi degli indirizzi di padre e figlio con codice e dati diversi (es. VMS, stesso processo dopo exec() in UNIX)

Terminazione

Ogni processo:

è **figlio** di un altro processo

può essere a sua volta **padre** di processi

SO deve mantenere le informazioni relative alle relazioni di *parentela*

➽ **nel descrittore: riferimento al padre**

Se un processo termina:

il padre può rilevare il suo ***stato di terminazione***

tutti i figli terminano

Interazione tra processi

I processi, pesanti o leggeri, *possono* interagire

Classificazione

processi indipendenti: due processi P1 e P2 sono indipendenti se l'esecuzione di P1 non è influenzata da P2, e viceversa

processi interagenti: P1 e P2 sono interagenti se l'esecuzione di P1 è influenzata dall'esecuzione di P2, e/o viceversa

Processi interagenti

Tipi di interazione

Cooperazione: l'interazione consiste nello ***scambio di informazioni*** al fine di eseguire un'attività comune

Competizione: i processi interagiscono per ***sincronizzarsi nell'accesso a risorse comuni***

Interferenza: ***interazione*** non desiderata e potenzialmente ***deleteria*** tra processi

Processi interagenti

Supporto all'interazione

L'interazione può avvenire mediante

- **memoria condivisa** (modello ad *ambiente globale*): SO consente ai processi (***thread***) di condividere variabili; l'interazione avviene tramite l'accesso a ***variabili condivise***
- **scambio di messaggi** (modello ad *ambiente locale*): i processi non condividono variabili e interagiscono mediante meccanismi di trasmissione/ricezione di messaggi; SO prevede dei meccanismi a supporto dello ***scambio di messaggi***

Processi interagenti

Aspetti

- ***concorrenza -> velocità***
- suddivisione dei compiti tra processi -> ***modularità***
- ***condivisione di informazioni***
  - assenza di replicazione: ogni processo accede alle stesse istanze di dati
  - necessità di ***sincronizzare i processi*** nell'accesso a dati condivisi

I Processi nel SO UNIX

Processi UNIX

UNIX è un sistema operativo *multiprogrammato a divisione di tempo*:

unità di computazione è il processo

Caratteristiche del processo UNIX:

processo pesante con codice *rientrante*

- ***dati non condivisi***
- ***codice condivisibile* con altri processi**

funzionamento dual mode

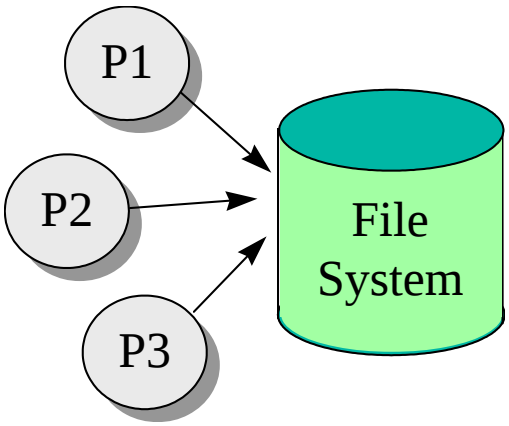
- processi di utente **(modo *user*)**
- processi di sistema **(modo *kernel*)**

diverse potenzialità e, in particolare, diversa visibilità della memoria

Modello di processo in UNIX

Ogni processo ha un proprio spazio di indirizzamento ***completamente locale e non condiviso***

➔ ***Modello ad Ambiente Locale***

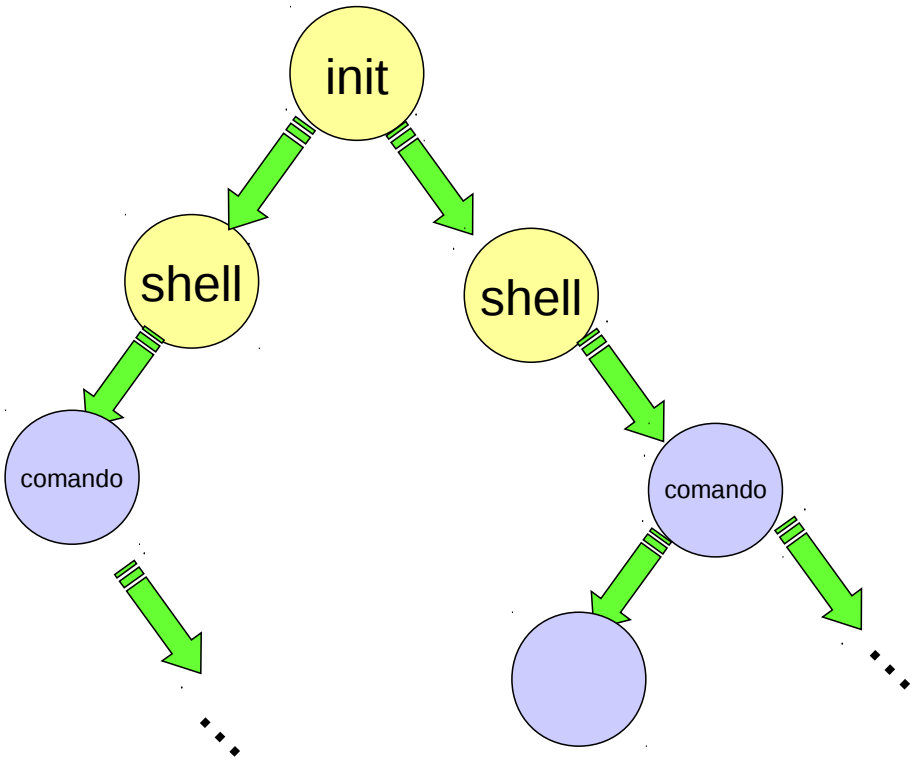


Eccezioni:

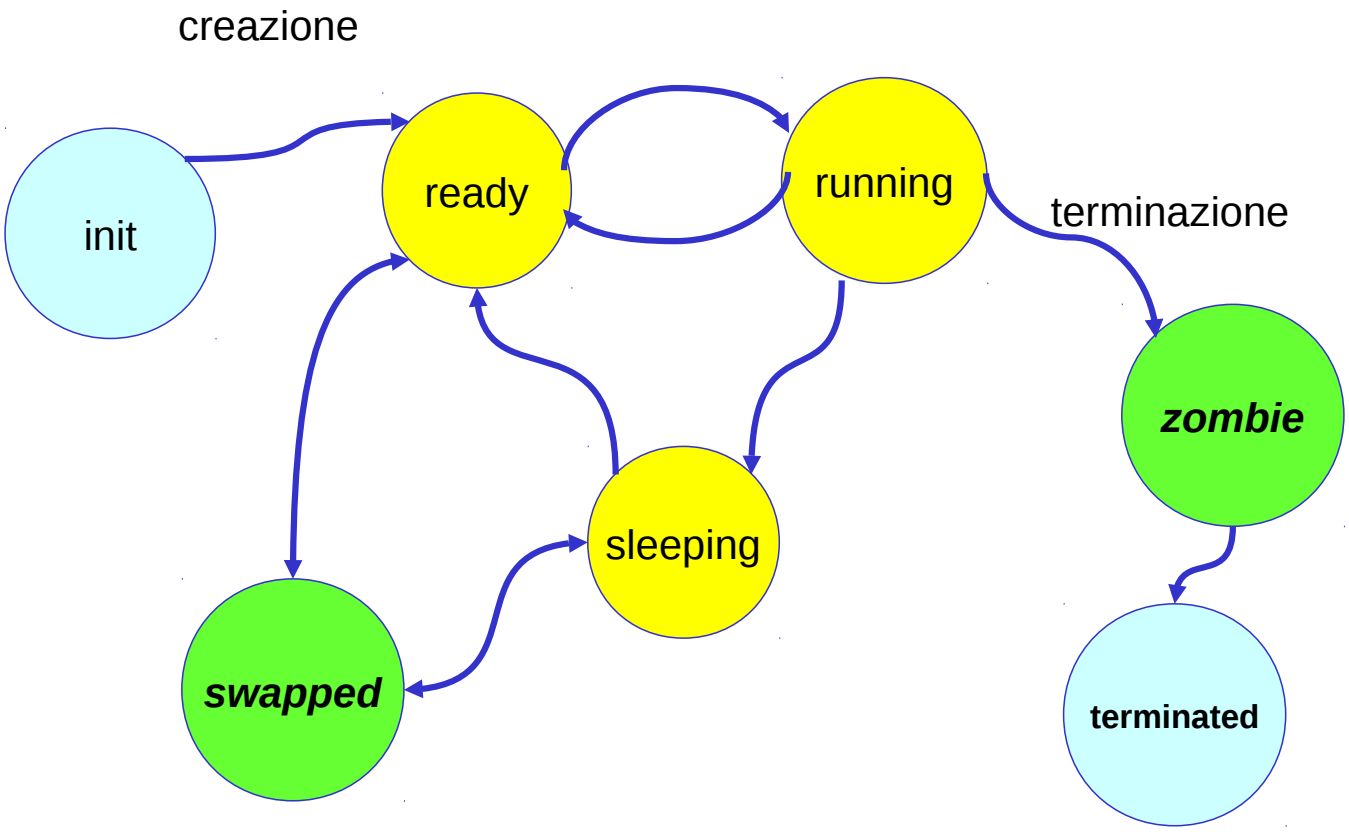
il codice può essere condiviso

il file system rappresenta un ambiente condiviso

Gerarchie di processi UNIX



Stati di un processo UNIX



Stati di un processo UNIX

Come nel caso generale

Init: ***caricamento in memoria*** del processo e inizializzazione delle strutture dati del SO

Ready: processo ***pronto***

Running: processo ***usa la CPU***

Sleeping: processo è ***sospeso in attesa di un evento***

Terminated: ***deallocazione*** del processo dalla memoria

In aggiunta

Zombie: processo è terminato, ma è ***in attesa che il padre ne rilevi lo stato di terminazione***

Swapped: processo (o parte di esso) è ***temporaneamente trasferito in memoria secondaria***

Processi swapped

Lo scheduler a medio termine (swapper) gestisce i trasferimenti dei processi

da memoria centrale a secondaria (dispositivo di swap): *swap out*

si applica preferibilmente ai ***processi bloccati (sleeping),*** prendendo in considerazione tempo di attesa, di permanenza in memoria e dimensione del processo (preferibilmente i ***processi più lunghi***)

da memoria secondaria a centrale: *swap in*

si applica preferibilmente ai ***processi più corti***

Rappresentazione dei processi UNIX

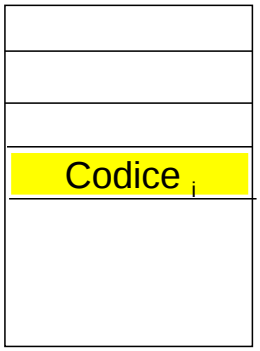
Il codice dei processi è rientrante: più processi possono condividere lo stesso codice (*text*)

codice e dati sono separati (modello a *codice puro*)

SO gestisce una ***struttura dati globale*** in cui sono contenuti i ***puntatori ai codici utilizzati***, eventualmente condivisi) dai processi: *text table*

L'elemento della text table si chiama *text structure* e contiene:

- **puntatore al codice** (se il processo è swapped, riferimento a memoria secondaria)
- numero dei processi che lo condividono...



Text table:

1 elemento ∀ segmento di codice utilizzato

PCB = process structure + user structure

Process structure (residente): mantiene le informazioni necessarie per la gestione del processo, anche se questo è ***swapped*** in memoria secondaria

User structure: il suo contenuto è necessario **solo in caso di esecuzione** del processo (***stato running***); se il processo è soggetto a swapping, anche ***la user structure può essere trasferita in memoria secondaria***

Process structure: contiene il riferimento a **user structure** (in memoria centrale o secondaria se swapped)

System call per la gestione di processi

Chiamate di sistema per

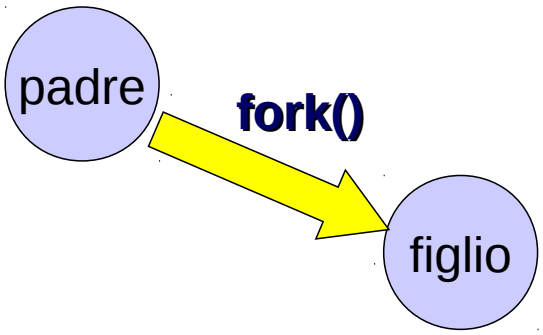
- ***creazione di processi***: **`fork()`**
- ***sostituzione di codice e dati***: **`exec...()`**
- ***terminazione***: **`exit()`**
- ***sospensione*** in attesa della ***terminazione di figli***: **`wait()`**

N.B. System call di UNIX sono attivabili attraverso chiamate a funzioni di librerie C standard: **`fork(), exec(), ...`** sono quindi funzioni di libreria che chiamano le system call corrispondenti

Creazione di processi: fork()

La funzione **fork()** consente a un processo di ***generare un processo figlio***:

- padre e figlio ***condividono lo STESSO codice***
- il figlio ***EREDITA una copia dei dati*** (***di utente e di kernel***) del padre



fork()

```
int fork(void);
```

`fork()` non richiede parametri

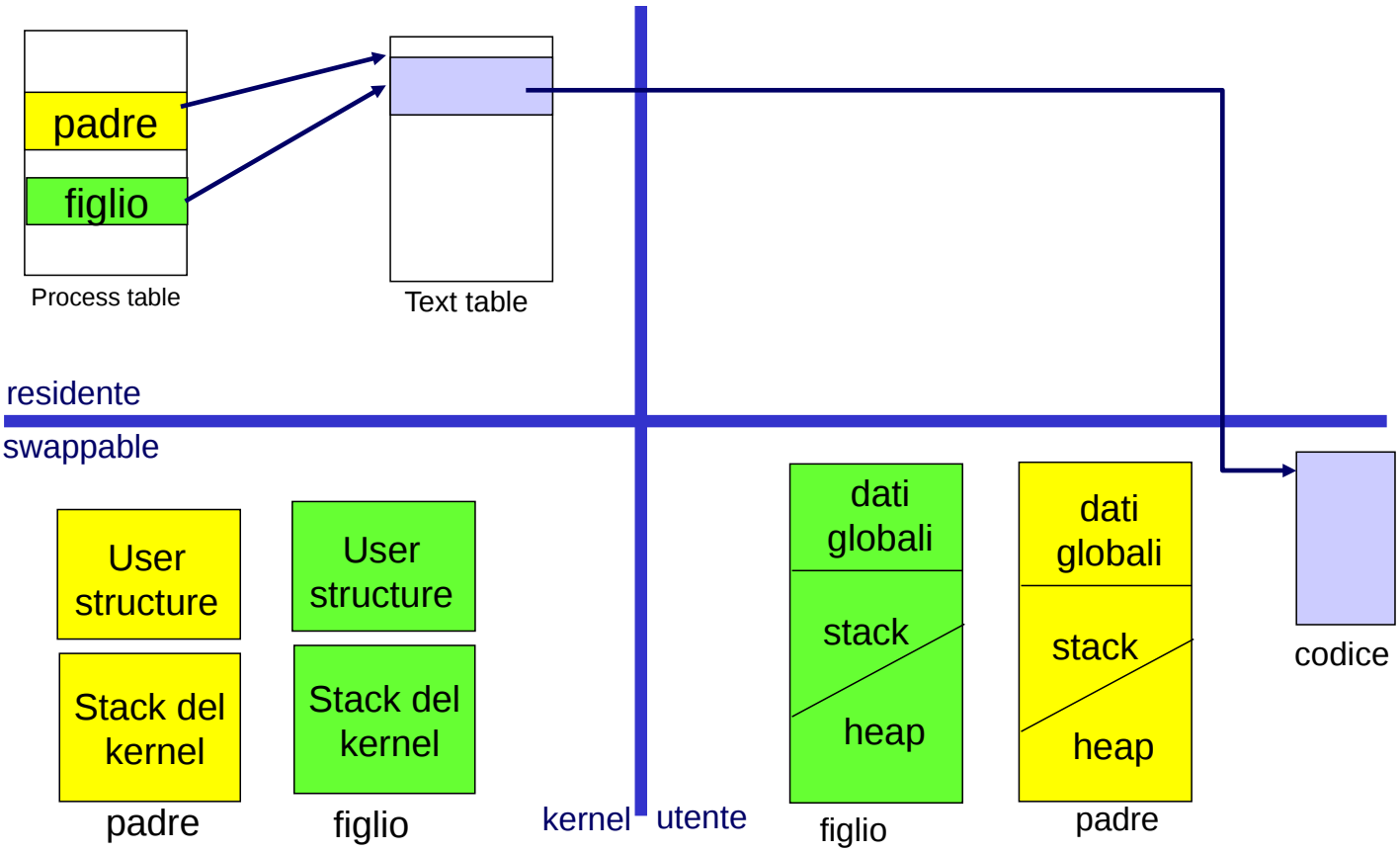
restituisce un intero che:

- ***per il processo creato vale 0***
- ***per il processo padre è un valore positivo che rappresenta il PID del processo figlio***
- è un valore **negativo** in caso di errore (la creazione non è andata a buon fine)

Effetti della fork()

- Allocazione di una ***nuova process structure*** nella process table associata al processo figlio e sua inizializzazione
- Allocazione di una ***nuova user structure*** nella quale viene ***copiata la user structure del padre***
- Allocazione dei ***segmenti di dati e stack*** del figlio nei quali vengono ***copiati dati e stack del padre***
- Aggiornamento del riferimento ***text*** al codice eseguito (condiviso col padre): incremento del contatore dei processi, ...

Effetti della fork()



Esecuzioni differenziate del padre e del figlio

```
...
if (fork()==0) {
  ...      /* codice eseguito dal figlio */
     ...
} else {
  .../* codice eseguito dal padre */
     ...
}
```

Dopo la generazione del figlio ***il padre può decidere***

se operare ***contemporaneamente*** ad esso

oppure

se ***attendere la sua terminazione*** (system call **`wait()`**)

fork(): esempio

```
#include <stdio.h>
main()
{ int pid;
    pid=fork();
    if (pid==0)
    { /* codice figlio */
   printf("Sono il figlio ! (pid: %d)\n", getpid());
   }
    else if (pid>0)
      { /* codice padre */
     printf("Sono il padre: pid di mio figlio: %d\n", pid);
    ....
      }
   else printf("Creazione fallita!");
 }
```

NB: system call **`getpid()`** ritorna il pid del processo che la chiama

Relazione padre-figlio in UNIX

Dopo una `fork()`:

concorrenza

padre e figlio procedono in parallelo

lo spazio degli indirizzi è duplicato

ogni ***variabile del figlio è inizializzata con il valore assegnatole dal padre*** prima della **`fork()`**

la **user structure** è duplicata

le ***risorse allocate al padre*** (ad esempio, i file aperti) prima della generazione sono ***condivise con i figli***

le informazioni per la gestione dei segnali sono le stesse per padre e figlio (associazioni segnali-handler)

il figlio nasce con lo ***stesso program counter del padre***: la prima istruzione eseguita dal figlio è quella che segue immediatamente **`fork()`**

Terminazione di processi

Un processo può terminare:

involontariamente

tentativi di azioni illegali

interruzione mediante segnale

salvataggio dell'immagine nel file **`core`**

volontariamente

chiamata alla funzione **`exit()`**

esecuzione dell'ultima istruzione

exit( )

```
void exit(int status);
```

la funzione **`exit()`** prevede un parametro (**`status`**) mediante il quale il processo che termina può comunicare al padre ***informazioni sul suo stato di terminazione*** (ad esempio esito dell'esecuzione)

è ***sempre una chiamata senza ritorno***

exit()

Effetti di una exit():

chiusura dei file aperti non condivisi

terminazione del processo

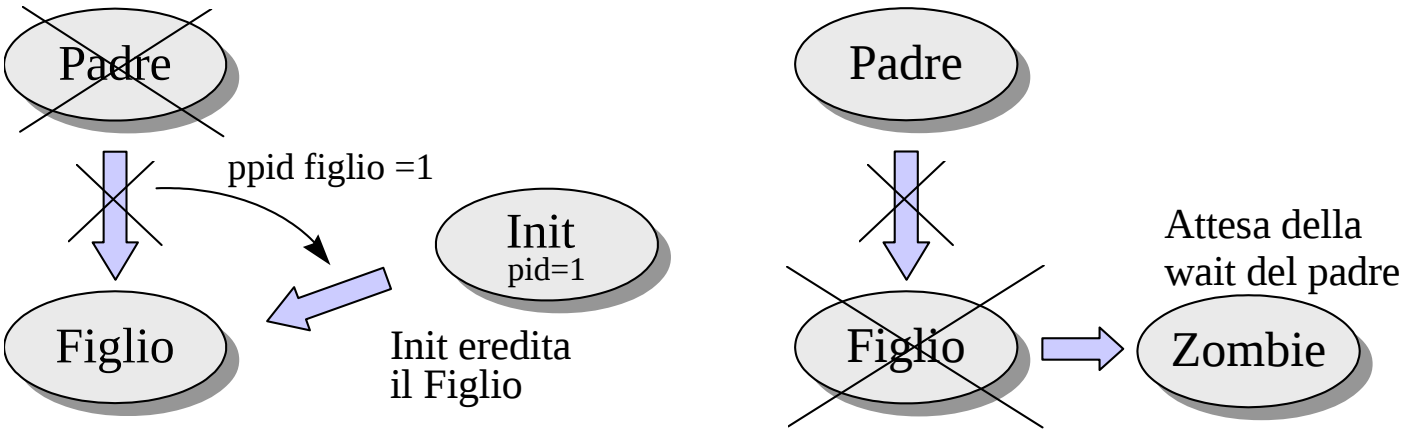
- se il processo che termina ha ***figli in esecuzione***, il processo ***init adotta i figli dopo la terminazione del padre*** (nella process structure di ogni figlio al pid del processo padre viene assegnato il valore 1)
- se il processo ***termina prima che il padre ne rilevi lo stato di terminazione*** con la system call **`wait(),`** il processo passa nello stato ***zombie***

NB: Quando termina un processo adottato dal processo **init**, **init** rileva automaticamente il suo stato di terminazione -> i processi figli di **init** non permangono nello stato di zombie

Parentela processi e terminazione

Terminazione del padre

Terminazione del figlio: processi zombie



System call exec()

Mediante **`fork()`** i processi ***padre e figlio condividono il codice e lavorano su aree dati duplicate***. In UNIX è possibile ***differenziare il codice dei due processi*** mediante una system call della famiglia **exec**

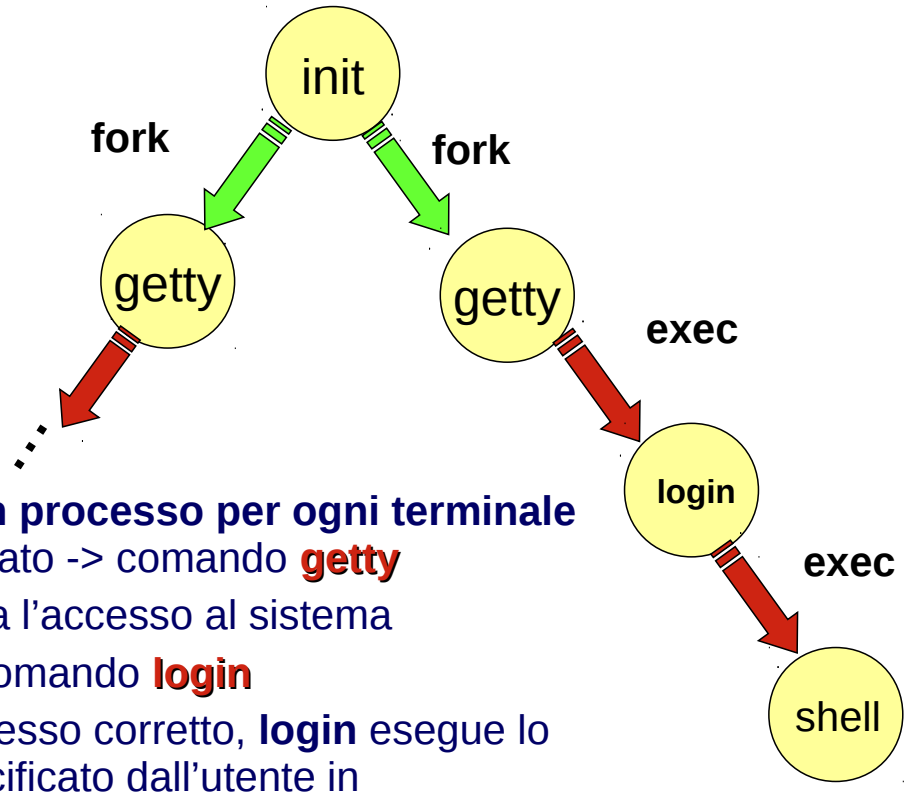
`execl(), execle(), execlp(), execv(), execve(), execvp()...`

Effetto principale di system call famiglia exec:

vengono ***sostituiti codice ed eventuali argomenti di invocazione*** del processo che chiama la system call, ***con codice e argomenti di un programma specificato come parametro*** della system call

NO generazione di nuovi processi

Inizializzazione dei processi UNIX



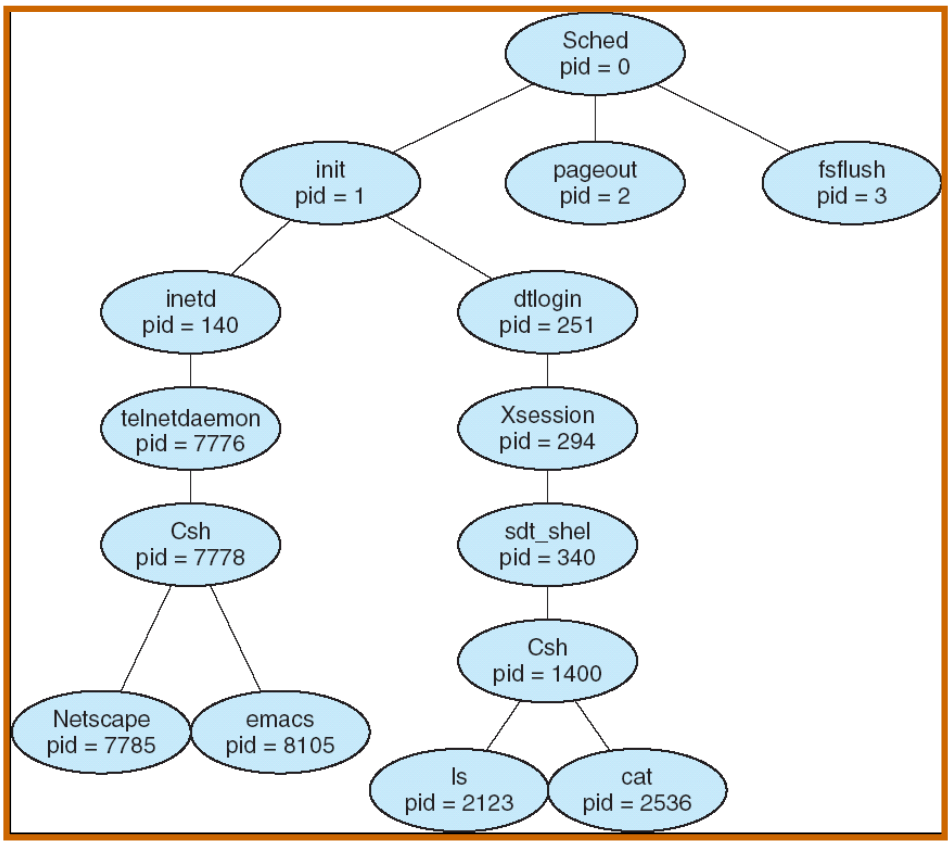
init genera **un processo per ogni terminale** (**tty**) collegato -> comando **getty**

getty controlla l'accesso al sistema

exec del comando **login**

in caso di accesso corretto, **login** esegue lo **shell** (specificato dall'utente in **/etc/passwd**)

Tipico albero di generazione di processi in Solaris

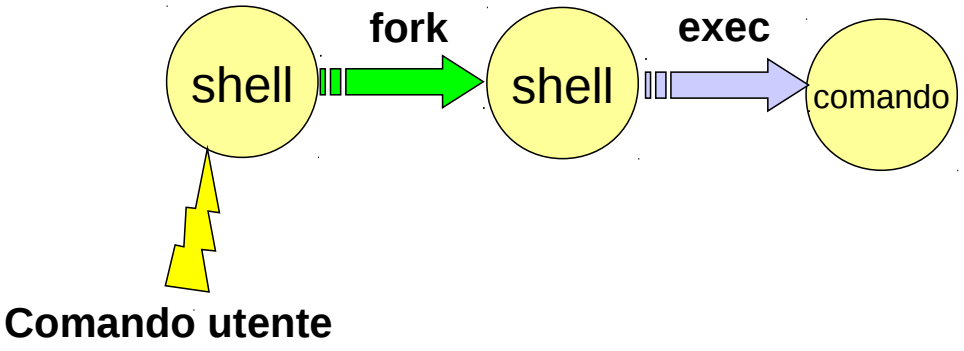


Interazione con l'utente tramite shell

Ogni utente può interagire con lo ***shell*** mediante la ***specifica di comandi***

Ogni ***comando*** è presente nel file system come ***file eseguibile*** (direttorio ***/*bin**)

Per ogni comando, ***shell genera un processo figlio*** dedicato all'esecuzione del comando:



Relazione shell padre-shell figlio

Per ogni comando, shell genera un figlio; possibilità di **due diversi comportamenti**:

- il padre si pone in attesa della terminazione del figlio (esecuzione in ***foreground***); es:

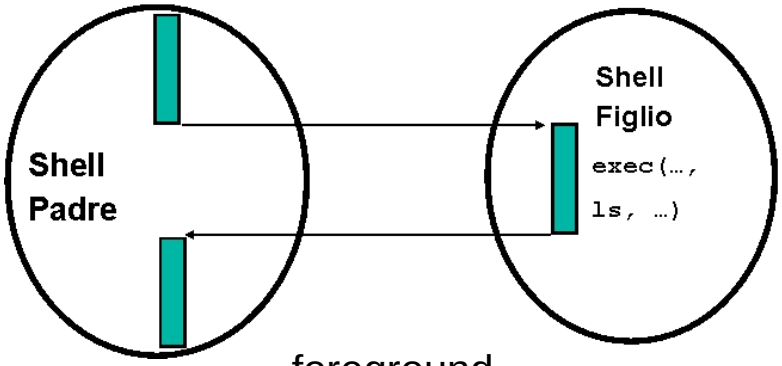
```
ls -l pippo
```

- il padre continua l'esecuzione concorrentemente con il figlio (esecuzione in ***background***):

```
ls -l pippo &
```

foreground vs background

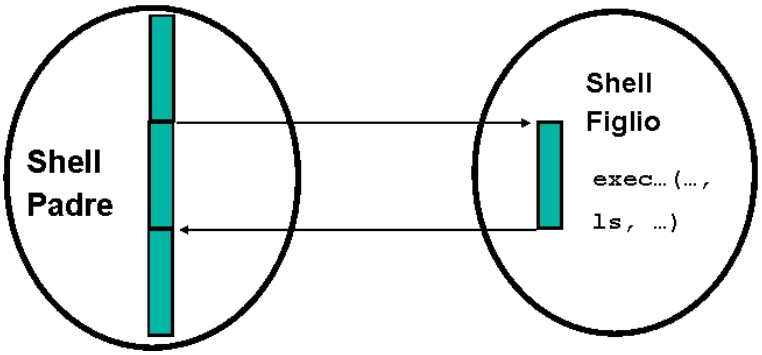
```
$ ls
```



foreground

```
$ ls&
```

background



Sincronizzazione tra processi UNIX: i segnali

Sincronizzazione tra processi

Processi interagenti possono avere bisogno di meccanismi di **sincronizzazione**

Ad esempio, abbiamo visto e rivedremo diffusamente il caso di ***processi pesanti UNIX*** che vogliano accedere allo ***stesso file in lettura/scrittura*** (***sincronizzazione di produttore e consumatore***)

UNIX: ***non c'è condivisione alcuna di spazio di indirizzamento tra processi***. Serve un meccanismo di sincronizzazione per modello ad ambiente locale

segnali

Sincronizzazione tra processi

Segnali

sono ***interruzioni software*** a un processo, che notifica un evento asincrono. Ad esempio segnali:

- generati da ***terminale*** (es. CTRL+C)
- generati da ***altri processi***
- generati dal ***kernel SO*** in seguito ad ***eccezioni HW*** (violazione dei limiti di memoria, divisioni per 0, …)
- generati dal ***kernel SO*** in seguito a ***condizioni SW*** (time-out, scrittura su pipe chiusa come vedremo in seguito, …)

Segnali UNIX

Un segnale può essere inviato

- dal kernel SO a un processo
- da un processo utente ad altri processi utente

(es. comando **`kill)`**

Quando un processo riceve un segnale, può comportarsi in ***tre modi diversi***

- ***gestire il segnale*** con una funzione ***handler*** definita dal programmatore
- ***eseguire un'azione predefinita*** dal SO (azione di ***default***)
- ***ignorare il segnale*** (nessuna reazione)

Nei primi due casi, il processo ***reagisce in modo asincrono*** al segnale

1. interruzione dell'esecuzione
2. esecuzione dell'azione associata (***handler*** o ***default***)
3. ritorno alla prossima istruzione del codice del processo interrotto

Segnali UNIX

Per ogni versione di UNIX esistono vari tipi di segnale (in Linux, 32 segnali), ognuno identificato da un intero

Ogni segnale è ***associato a un particolare evento*** e prevede una ***specifica azione di default***

È possibile riferire i segnali con identificatori simbolici (**`SIGxxxx`**):

`SIGKILL, SIGSTOP, SIGUSR1`, ...

L'associazione tra nome simbolico e intero corrispondente (che dipende dalla versione di UNIX) è specificata nell'header file **`<signal.h>`**

Segnali UNIX (Linux): signal.h

```
#define SIGHUP   1   /* Hangup (POSIX). Action: exit */
#define SIGINT   2   /* Interrupt (ANSI). CTRL+C. Action: exit */
#define SIGQUIT  3   /* Quit (POSIX). Action: exit, core dump */
#define SIGILL   4   /* Illegal instr (ANSI). Action: exit,
                                core dump */
...
#define SIGKILL  9   /* Kill, unblockable (POSIX). Action: exit */
#define SIGUSR1  10  /* User-def sig1 (POSIX). Action: exit */

#define SIGSEGV  11  /* Segm. violation (ANSI). Action: exit, core
                        dump */
#define SIGUSR2  12  /* User-def sig2 (POSIX). Action: exit */
#define SIGPIPE  13  /* Broken pipe (POSIX). Action: exit  */
#define SIGALRM  14  /* Alarm clock (POSIX). Action: exit */
#define SIGTERM  15  /* Termination (ANSI). Action: exit */
...
#define SIGCHLD  17  /* Chld stat changed (POSIX). Action: ignore  */
#define SIGCONT  18  /* Continue (POSIX). Action ignore  */
#define SIGSTOP  19  /* Stop, unblockable (POSIX). Action: stop */
...
```


Gestione dei segnali UNIX

Quando un processo riceve un segnale, può gestirlo in 3 modi diversi:

- gestire il segnale con una ***funzione handler definita dal programmatore***
- eseguire ***un'azione predefinita*** dal SO (azione di ***default***)
- ***ignorare*** il segnale

NB: non tutti i segnali possono essere gestiti in modalità scelta esplicitamente dai processi: **`SIGKILL`** e **`SIGSTOP`** non sono ***né intercettabili, né ignorabili***

- qualunque processo, alla ricezione di SIGKILL o SIGSTOP esegue sempre ***l'azione di default***

Astrazione di File System

Gestione del file system

Parte di SO che fornisce i ***meccanismi di accesso e memorizzazione*** delle informazioni (programmi e dati) ***allocate in memoria di massa***

Realizza i ***concetti astratti***

di ***file***: ***unità logica*** di memorizzazione

di *direttorio*: ***insieme di file*** (e direttori)

di *partizione*: insieme di file associato ad un ***particolare dispositivo fisico*** (o porzione di esso)

Le ***caratteristiche*** di file, direttorio e partizione sono ***del tutto indipendenti*** da natura e tipo di dispositivo utilizzato

File

È un insieme di informazioni:

- programmi
- dati (in rappresentazione binaria)
- dati (in rappresentazione testuale)
- …

rappresentati come ***insieme di record logici***

Ogni file è ***individuato da (almeno) un nome simbolico*** mediante il quale può essere riferito (ad esempio, nell'invocazione di comandi o system call)

Ogni file è ***caratterizzato da un insieme di attributi***

Attributi del file

A seconda del SO, i file possono avere attributi diversi. Solitamente

- ***tipo***: stabilisce l'appartenenza a una classe (eseguibili, testo, musica, non modificabili, …)
- ***indirizzo***: puntatore/i a memoria secondaria
- ***dimensione***: numero di byte contenuti nel file
- ***data e ora*** (di creazione e/o di modifica)

In SO multiutente anche

- ***utente proprietario***
- ***protezione***: ***diritti di accesso*** al file per gli utenti del sistema

Attributi del file

***Descrittore* del file:**
è la struttura dati che contiene gli attributi di un file

Ogni ***descrittore*** di file deve essere ***memorizzato in modo persistente***:

SO mantiene ***l'insieme dei descrittori di tutti i file presenti nel file system in apposite strutture*** in memoria secondaria (ad es. UNIX: ***i-list***, lo vedremo diffusamente)

Tipi di file: nomi ed estensioni

In alcuni SO, ***l'estensione*** inclusa nel nome di un file ***rappresenta il suo tipo***

NON è il caso di UNIX

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Operazioni sui file

Compito del SO è consentire ***l'accesso on-line ai file*** (ogni volta che un processo ***modifica un file***, tale cambiamento è ***immediatamente visibile*** per tutti gli altri processi)

Tipiche Operazioni

- ***Creazione***: allocazione di un file in memoria secondaria e inizializzazione dei suoi attributi
- ***Lettura*** di record logici dal file
- ***Scrittura***: inserimento di nuovi record logici all'interno di file
- ***Cancellazione***: eliminazione del file dal file system

Ogni operazione richiederebbe la localizzazione di informazioni su disco, come:

- indirizzi dei record logici a cui accedere
- altri attributi del file
- record logici

-> costo elevato

Operazioni sui file

Per migliorare l'efficienza:

- SO mantiene ***in memoria una struttura che registra i file attualmente in uso (file aperti) - tabella dei file aperti***
  - per ogni file aperto **{puntatore al file, posizione su disco, ...}**
- Spesso viene fatto il ***memory mapping dei file aperti***:
  - i file aperti (o porzioni di essi) vengono temporaneamente copiati in memoria centrale ➔ accessi più veloci

Operazioni necessarie

- **Apertura**: introduzione di un ***nuovo elemento nella tabella dei file aperti*** e eventuale memory mapping del file
- **Chiusura**: ***salvataggio*** del file in memoria secondaria ed ***eliminazione*** dell'elemento corrispondente dalla ***tabella dei file aperti***

Struttura interna dei file

Ogni dispositivo di memorizzazione secondaria viene ***partizionato in blocchi (o record fisici):***

Blocco: unità di ***trasferimento fisico*** nelle operazioni di I/O da/verso il dispositivo. Sempre di ***dimensione fissa***

L'utente vede il file come un ***insieme di record logici:***

Record logico: unità di ***trasferimento logico*** nelle operazioni di accesso al file (es. lettura, scrittura di blocchi). Di ***dimensione variabile***

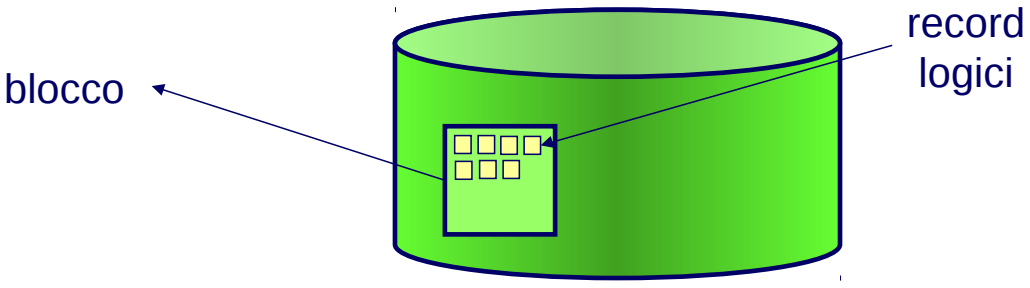
Blocchi & record logici

Uno dei compiti di SO (parte di gestione del file system) è stabilire una ***corrispondenza tra record logici e blocchi***

Usualmente:

Dimensione(blocco) >> Dimensione(record logico)

impaccamento di record logici all'interno di blocchi



Metodi di accesso

L'accesso a file può avvenire secondo varie modalità:

- ***accesso sequenziale***
- ***accesso diretto***
- ***accesso a indice***

Il metodo di accesso è ***indipendente***:

- ***dal tipo di dispositivo*** utilizzato
- ***dalla tecnica di allocazione*** dei blocchi in memoria secondaria

Accesso sequenziale

Il file è una **sequenza** [R_1, R_2, ..., R_N] di record logici:

per accedere ad un particolare record logico R_i , è necessario accedere ***prima agli (i-1) record che lo precedono*** nella sequenza:

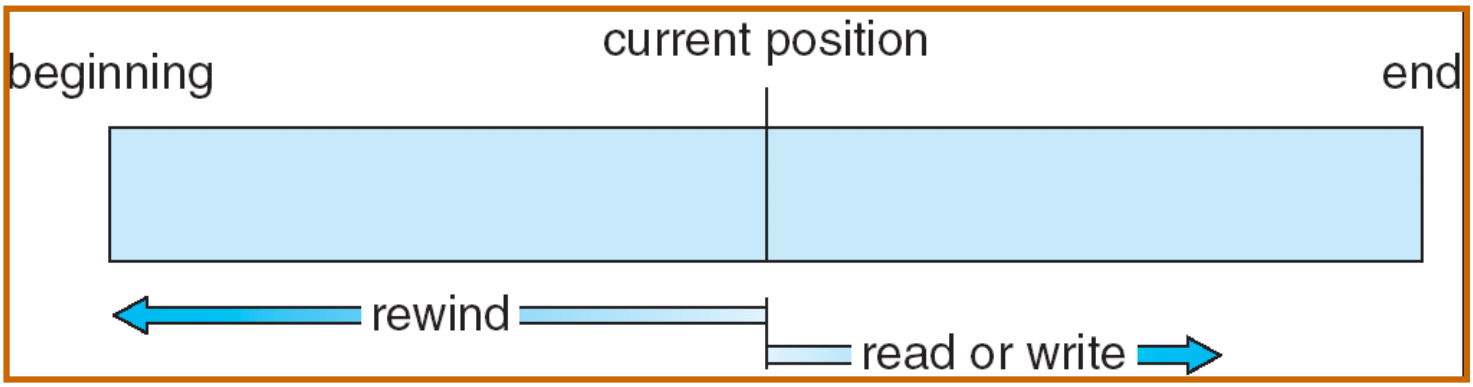
le operazioni di accesso sono del tipo:

- ***readnext***: lettura del prossimo record logico della sequenza
- ***writenext***: scrittura del prossimo record logico

ogni operazione di accesso (lettura/scrittura) posiziona il ***puntatore al file*** sull'elemento successivo a quello letto/scritto

UNIX prevede questo tipo di accesso

Accesso sequenziale



ogni operazione di accesso (lettura/scrittura) posiziona il ***puntatore al file*** sull'elemento successivo a quello letto/scritto

UNIX prevede questo tipo di accesso

Accesso diretto

Il file è un ***insieme*** $\{R_1, R_2, ..., R_N\}$ di ***record logici numerati*** con associata la nozione di ***posizione***:

si può accedere direttamente a un particolare record logico specificandone il numero

operazioni di accesso sono del tipo

read i: lettura del record logico i

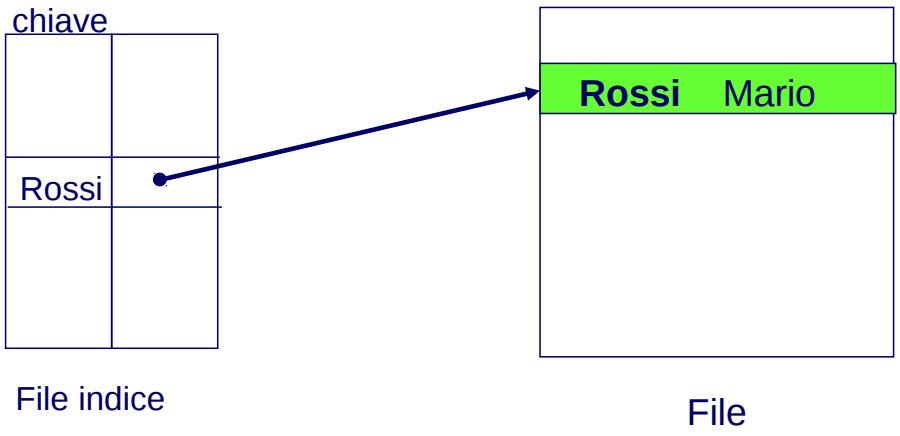
write i: scrittura del record logico i

Utile quando si vuole accedere a grossi file per estrarre/aggiornare poche informazioni (ad esempio nell'accesso a database)

Accesso a indice

Ad ogni file viene associata una ***struttura dati*** contenente l'***indice*** delle informazioni contenute

per accedere a un record logico, si esegue ***una ricerca nell'indice (utilizzando una chiave)***



Directory (o direttorio)

Strumento per ***organizzare i file all'interno del file system***:

una directory può contenere più file

è realizzata mediante una ***struttura dati*** che associa al nome di ogni file come e/o dove esso è allocato in memoria di massa

Operazioni sui direttori:

Creazione/cancellazione di directory

Aggiunta/cancellazione di file

Listing: elenco di tutti i file contenuti nella directory

Attraversamento della directory

Ricerca di file in directory

Tipi di directory

La ***struttura logica delle directory*** può variare a seconda del SO

Schemi più comuni:

- ***a un livello***
- ***a due livelli***
- ***ad albero***
- ***a grafo aciclico***

Tipi di directory

Struttura a un livello: una sola directory per ogni file system

Problemi

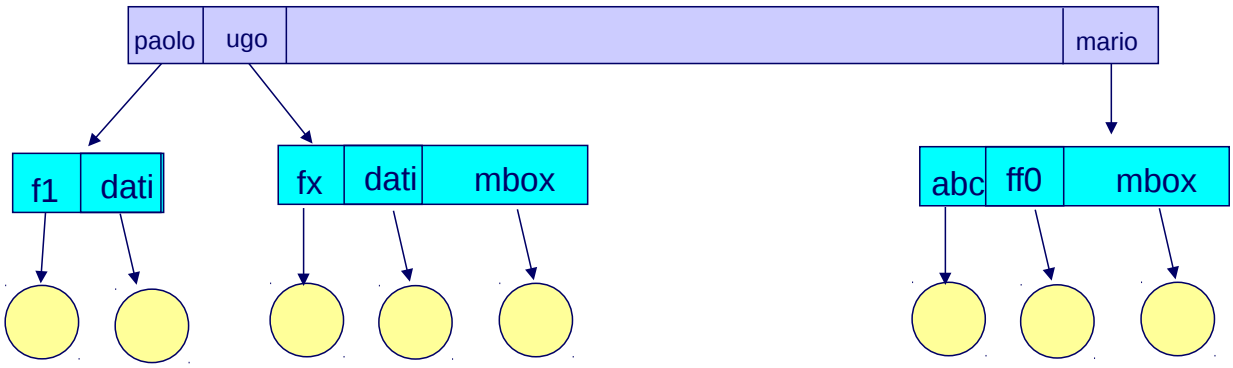
- ***unicità dei nomi***
- ***multiutenza***: come separare i file dei diversi utenti?

Tipi di directory

Struttura a due livelli

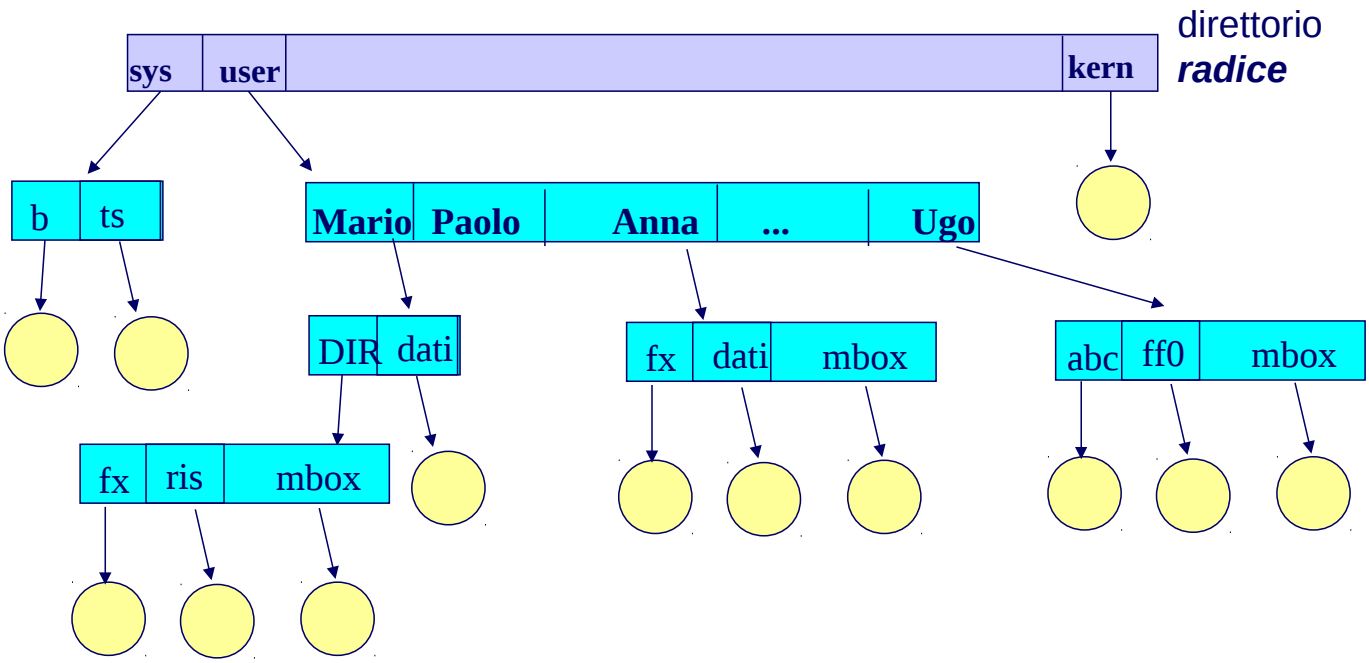
primo livello (***directory principale***): contiene una directory per ogni utente del sistema

secondo livello: ***directory utenti*** (a un livello)



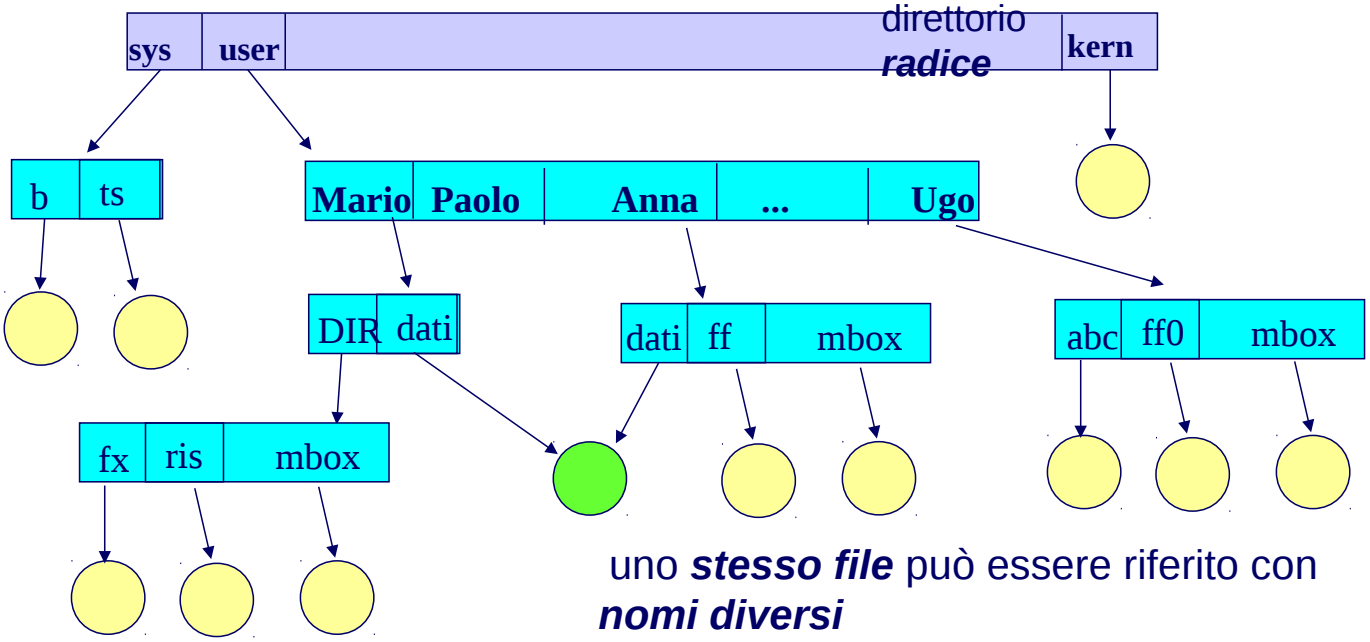
Tipi di directory

Struttura ad albero: *organizzazione gerarchica a N livelli*. Ogni direttorio può contenere file e altri direttori



Tipi di directory

Struttura a grafo aciclico (es. UNIX): estende la struttura ad albero con la possibilità di ***inserire link differenti allo stesso file***

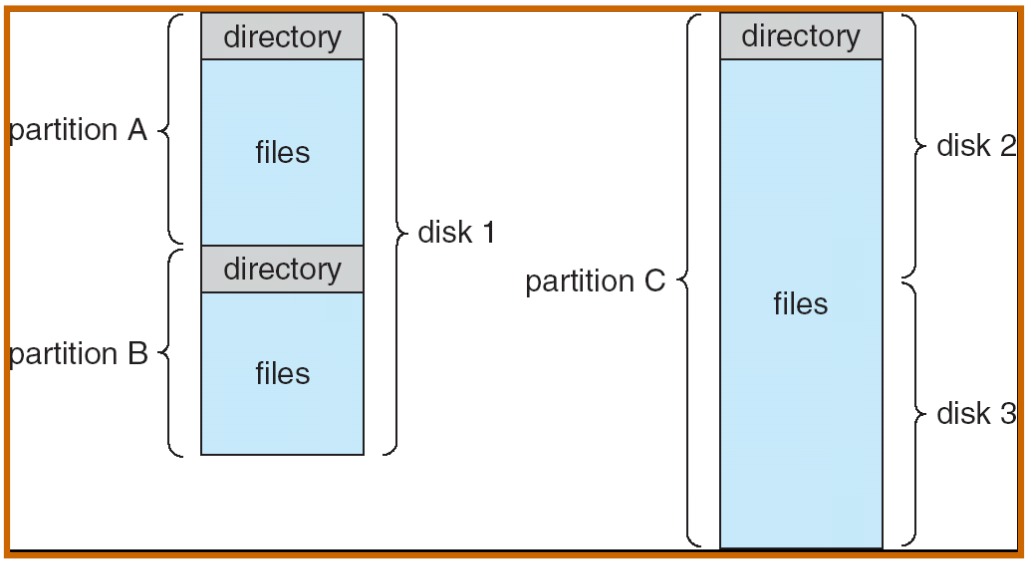


uno ***stesso file*** può essere riferito con ***nomi diversi***

Directory e partizioni

Una singola unità disco può contenere più partizioni

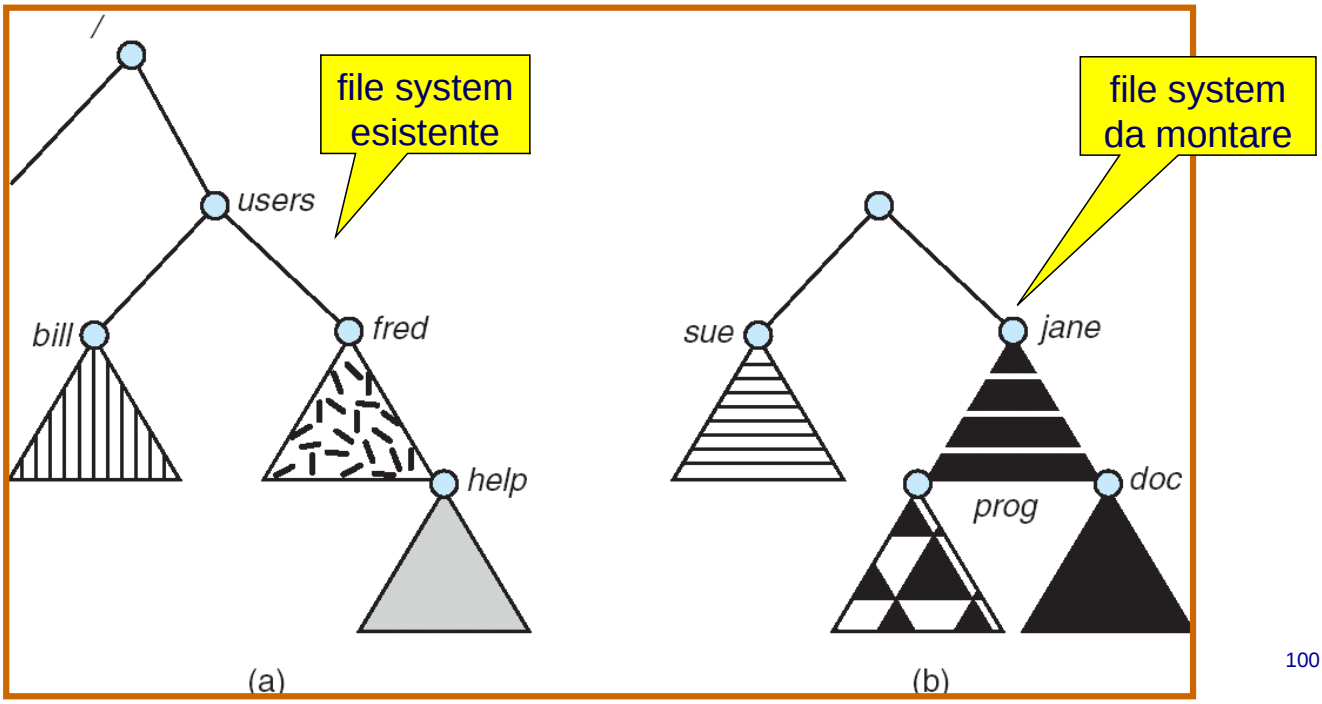
Una singola partizione può utilizzare più di una unità disco



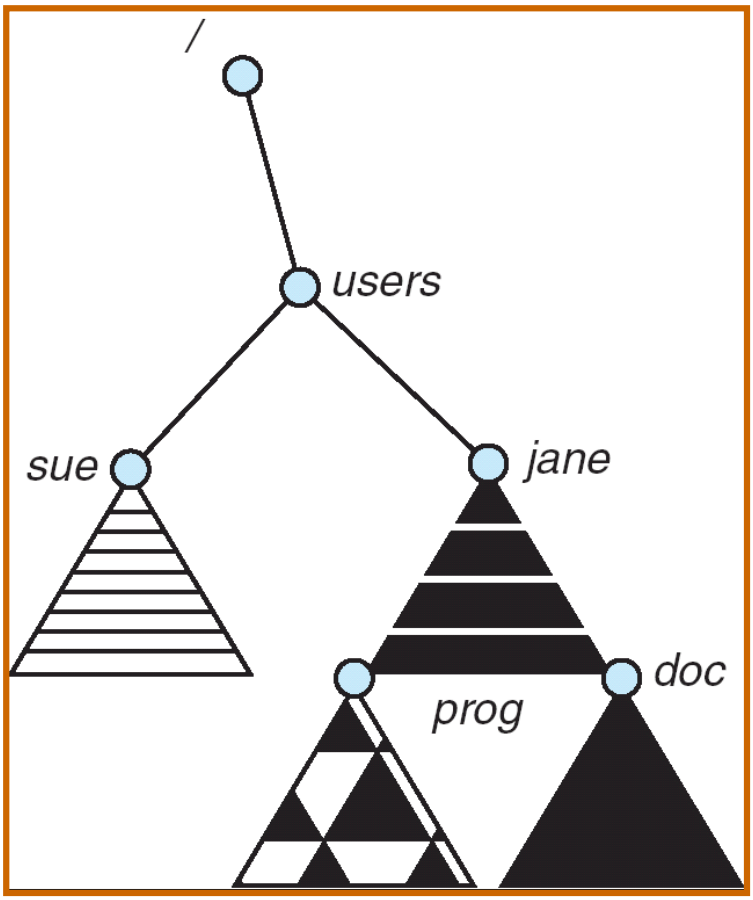
Unità disco e organizzazione/posizione di directory all'interno del file system devono essere correlati?

File System Mounting

Molti SO richiedono il ***mounting esplicito*** all'interno del file system prima di poter usare una (nuova) ***unità disco***



Dopo il mounting ad un determinato mount point



File system e protezione

Il ***proprietario/creatore*** di un file dovrebbe avere la possibilità di ***controllare***:

quali azioni sono consentite sul file

da parte di chi

Possibili tipologie di accesso

- **Read**
- **Write**
- **Execute**
- **Append**
- **Delete**
- **List**

Liste di accesso e gruppi (es. UNIX)

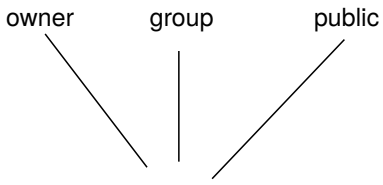
Modalità di accesso: read, write, execute

3 classi di utenti

			RWX
1) **owner access**	7	⇒	1 1 1
			RWX
2) **group access**	6	⇒	1 1 0
			RWX
3) **public access**	1	⇒	0 0 1

Amministratore può creare gruppi (con nomi unici) e inserire/eliminare utenti in/da quel gruppo

Dato un file o una directory, si devono definire le regole di accesso desiderate

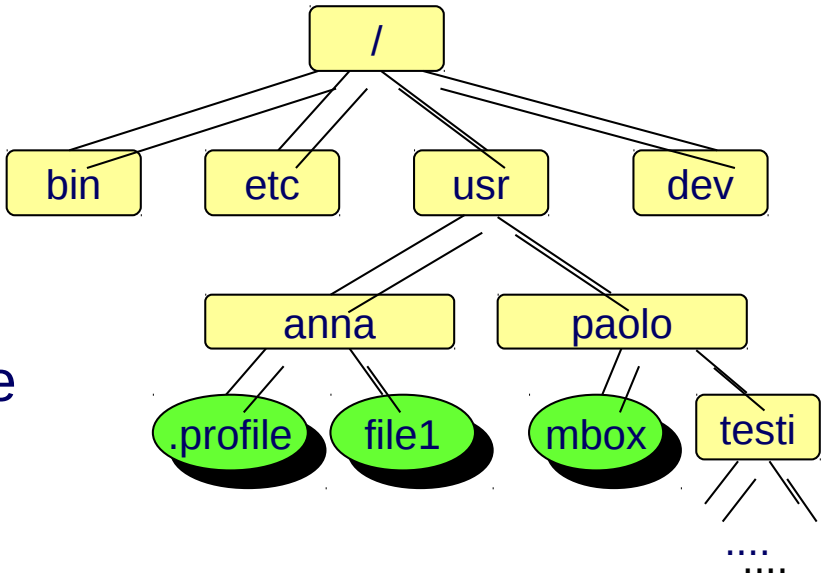


Cambiamento di gruppo: **`chgrp`** `G game`

Cambiamento di diritti di accesso: **chmod** 761 game

Il File System di UNIX

UNIX file system: organizzazione logica



omogeneità: tutto è file

tre categorie di file

- file ***ordinari***
- ***direttori***
- *dispositivi fisici*: file speciali (nel direttorio ***/dev***)

Nome, *i-number, i-node*

ad ogni file possono essere associati ***uno o più nomi simbolici***

ma

ad ogni file è associato ***uno ed un solo descrittore (i-node),*** univocamente identificato da un ***intero (i-number)***

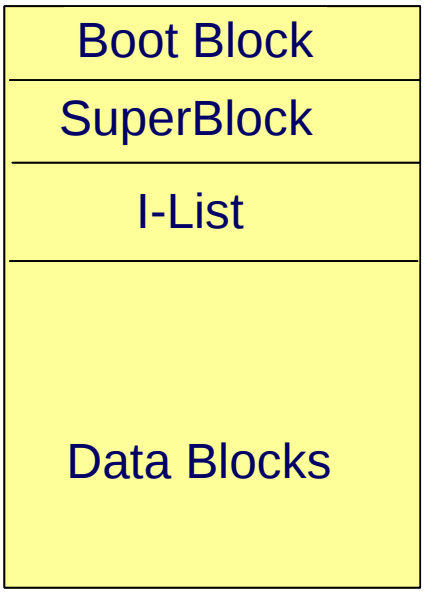
UNIX file system: organizzazione fisica

metodo di allocazione utilizzato in UNIX è ad ***indice*** (*a più livelli di indirizzamento*)

formattazione del disco in ***blocchi fisici*** ***(dimensione del blocco: 512B-4096B)***

superficie del disco partizionata in ***4 regioni***:

- ***boot block***
- ***super block***
- ***i-list***
- ***data blocks***



UNIX file system: organizzazione fisica

Boot Block contiene le ***procedure di inizializzazione*** del sistema (da eseguire al *bootstrap*)

Super Block fornisce

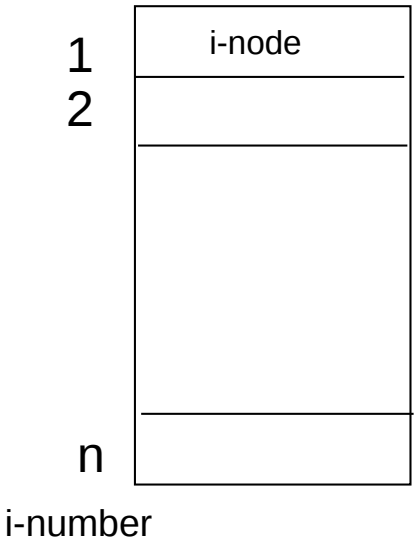
- i limiti delle 4 regioni
- il puntatore a una ***lista dei blocchi liberi***
- il puntatore a una ***lista degli i-node liberi***

Data Blocks - area del disco effettivamente disponibile per la memorizzazione dei file. Contiene:

- i ***blocchi allocati***
- i ***blocchi liberi*** (organizzati in una ***lista collegata***)

UNIX file system: organizzazione fisica

i-List contiene la ***lista di tutti i descrittori (i-node)*** dei file normali, direttori e dispositivi presenti nel file system (accesso con l'indice ***i-number***)



i-node

i-node è il descrittore del file

Tra gli **attributi** contenuti nell'*i-node* vi sono:

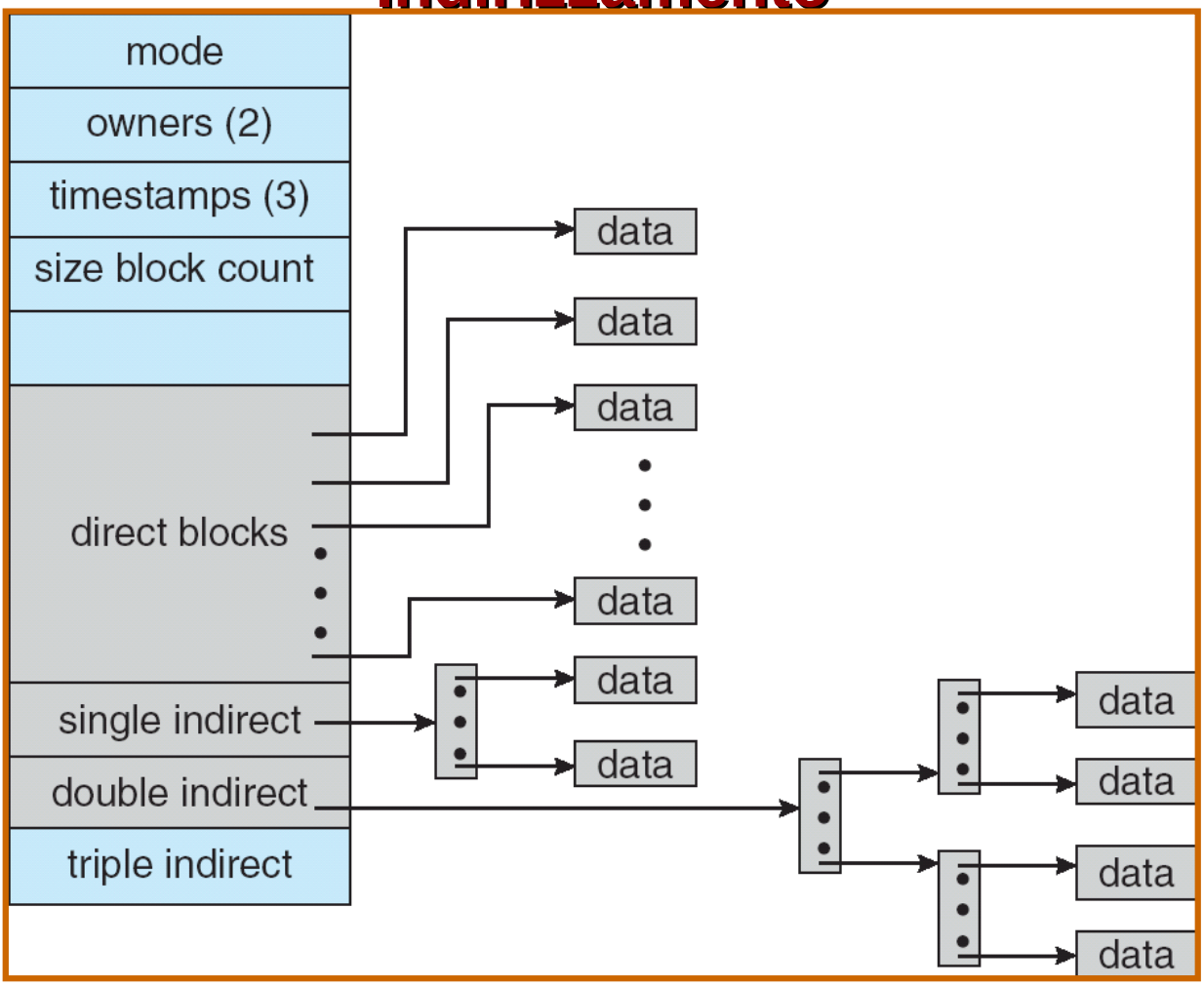
- ***tipo*** di file:
  - **ordinario**
  - **direttorio**
  - file **speciale**, per i dispositivi
- ***proprietario, gruppo*** (user-id, group-id)
- ***dimensione***
- ***data***
- 12 bit di ***protezione***
- numero di ***link***
- ***13-15 indirizzi*** di blocchi (a seconda della realizzazione)

Indirizzamento

L'allocazione del file ***NON è*** su blocchi fisicamente contigui. ***Nell'i-node sono contenuti puntatori a blocchi*** (ad esempio **13**)**,** dei quali:

- **i primi 10 indirizzi** riferiscono blocchi di dati (indirizzamento *diretto*)
- **11° indirizzo**: indirizzo di un blocco contenente a sua volta indirizzi di blocchi dati (primo livello di *indirettezza*)
- **12° indirizzo**: secondo livello di *indirettezza*
- **13° indirizzo**: terzo livello di *indirettezza*

Indirizzamento



Indirizzamento

Se: dimensione del blocco **512B**=0,5KB
indirizzi di 32 bit (4 byte)
→ 1 blocco contiene **128** indirizzi

10 blocchi di dati sono accessibili direttamente
file di dimensioni minori di 10*512 B=5KB sono accessibili direttamente

128 blocchi di dati sono sono accessibili con indirettezza singola (mediante il puntatore 11): 128*512 B=64KB

128*128 blocchi di dati sono accessibili con indirettezza doppia (mediante il puntatore 12): 128*128*512 B=8MB

128*128*128 blocchi di dati sono accessibili con indirettezza tripla (mediante il puntatore 13):
128*128*128*512 B=1GB

Indirizzamento

la dimensione massima del file realizzabile è dell'ordine del **GB**

Dimensione massima = 1GB+ 8MB+64KB+5KB

➨ vantaggio aggiuntivo: ***l'accesso a file di piccole dimensioni è più veloce*** rispetto al caso di file grandi

Una parentesi su Linux file system

All'utente finale il file system Linux appare come una struttura ad albero gerarchico che ***obbedisce alla semantica UNIX***

Internamente, il kernel di Linux è capace di gestire ***differenti file system multipli*** fornendo un livello di ***astrazione uniforme: virtual file system (VFS)***

Linux VFS sfrutta:

- Un insieme di ***descrittori*** che definiscono come un file debba ***apparire*** (*inode object, file object, file system object)*
- Un insieme di funzionalità software per gestire questi oggetti

Cenni pratici introduttivi all'utilizzo del file system Linux

File

File come ***risorsa logica*** costituita da ***sequenza di bit***, a cui viene dato un nome

Astrazione molto potente che consente di ***trattare allo stesso modo entità fisicamente diverse*** come file di testo, dischi rigidi, stampanti, direttori, tastiera, video, ...

Ordinari

archivi di dati, comandi, programmi sorgente, eseguibili, …

Directory

gestiti direttamente solo da SO, contengono riferimenti a file

Speciali

dispositivi hardware, memoria centrale, hard disk, ...

In aggiunta, anche:

FIFO (pipe) - file per la comunicazione tra processi

soft link - riferimenti (puntatori) ad altri file o direttori

File: nomi

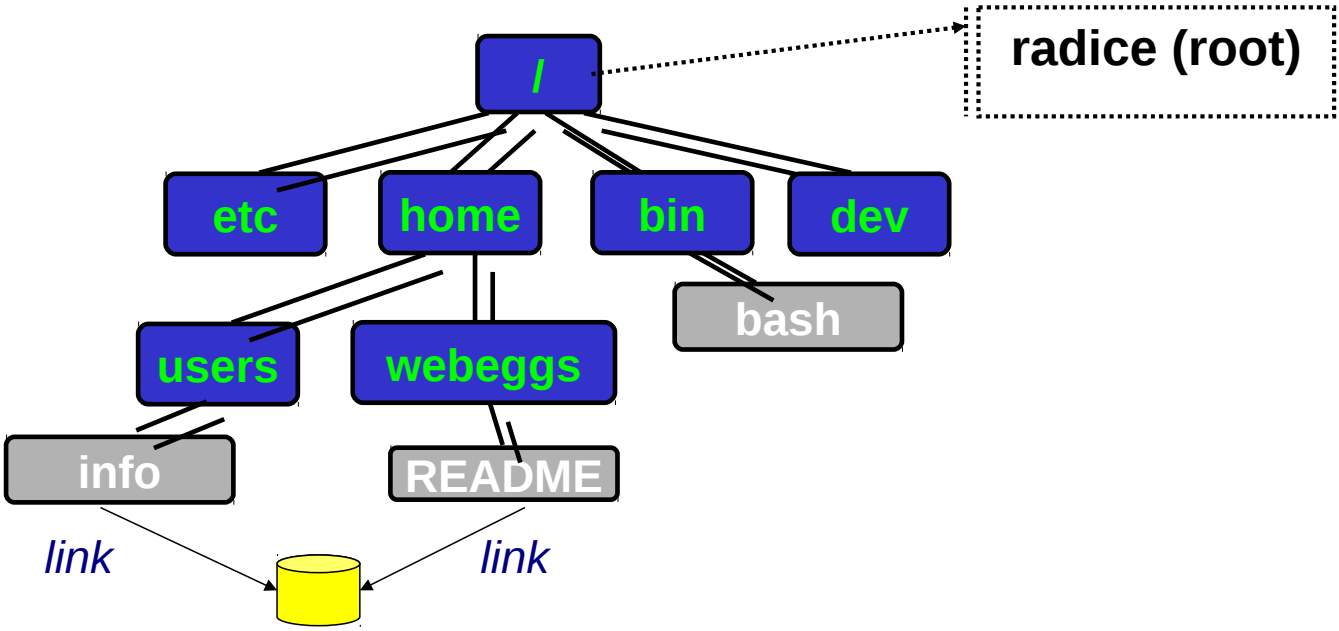
È possibile nominare un file con una ***qualsiasi sequenza di caratteri (max 255)***, a eccezione di '.' e '..'

È sconsigliabile utilizzare per il nome di file dei caratteri speciali, ad es. ***metacaratteri e segni di punteggiatura***

ad ogni file possono essere associati ***uno o più nomi simbolici (link)*** ma ad ogni file è associato ***uno e un solo descrittore (i-node)*** identificato da un intero (i-number)

directory

File system Linux è organizzato come un grafo diretto aciclico (DAG)



Gerarchie di directory

All'atto del login, l'utente può cominciare a operare all'interno di una specifica directory (***home***). In seguito è possibile cambiare directory

È possibile visualizzare il percorso completo attraverso il ***comando pwd*** (print working directory)

Essendo i file organizzati in ***gerarchie di directory***, SO mette a disposizione dei comandi per muoversi all'interno di essi

Nomi relativi/assoluti

Ogni utente può specificare un file attraverso

- ***nome relativo***: è riferito alla posizione dell'utente nel file system (direttorio corrente)
- ***nome assoluto***: è riferito alla radice della gerarchia ***/***

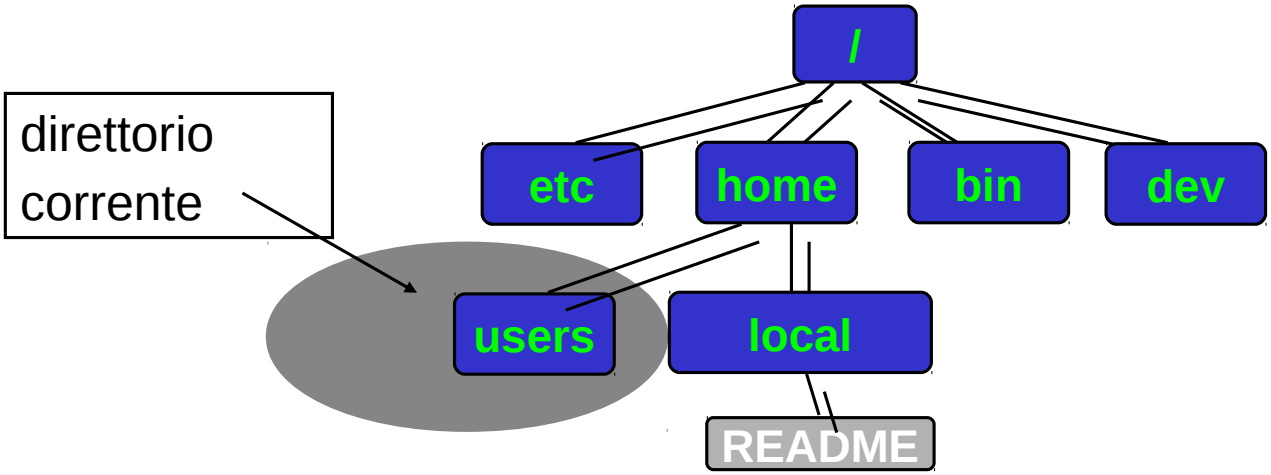
Nomi particolari

- **.** è il direttorio corrente (visualizzato da **pwd**)
- **..** è il direttorio 'padre'
- ~ è la propria home utente

Il comando ***cd permette di spostarsi all'interno del file system***, utilizzando sia nomi relativi che assoluti

- **cd** senza parametri porta alla home dell'utente

Nomi relativi/assoluti: esempio



nome assoluto: **`/home/local/README`**

nome relativo: **`../local/README`**

Link

Le informazioni contenute in uno ***stesso file*** possono essere ***visibili come file diversi***, tramite "riferimenti" (link) allo stesso file fisico

SO considera e gestisce la molteplicità possibile di riferimenti:

se un file viene cancellato, le ***informazioni sono veramente eliminate solo se non ci sono altri link a esso***

Il link ***cambia i diritti***? ➔ ***Meglio di no***

Due tipi di link:

link fisici (si collegano le strutture del file system)

link simbolici (si collegano solo i nomi)

comando: **`ln [-s]`**

Filesystem Linux

Collocazione delle risorse

Partizioni

Filesystem

Collocazione delle risorse

FHS (Filesystem Hierarchy Standard) definisce la struttura del filesystem Unix allo scopo di rendere più facile a programmi automatici ed utenti l'individuazione delle risorse, rendere più efficiente la condivisione di parti del filesystem e rendere più sicura la memorizzazione dei dati.

http://www.pathname.com/fhs/pub/fhs-2.3.html

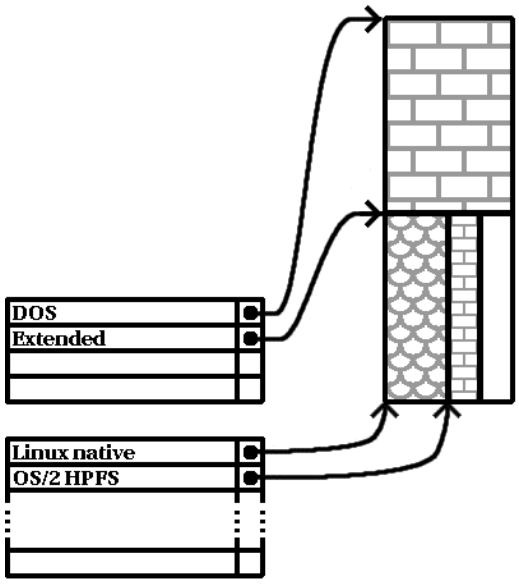
Le distinzioni base che guidano alla corretta collocazione dei dati in FHS sono 2:

	condivisibili	non condivisibili
statici	es. /usr /opt	es. /etc /boot
variabili	es. /var/mail /var/spool/news	es. /var/run /var/lock

Partizionamento dei dischi

Le quattro tipologie di dati sono, nei sistemi più strutturati, collocate in partizioni differenti, al fine di ottenere:

- ♦ razionalizzazione dello spazio
- ♦ possibilità di coesistenza sullo stesso disco di filesystem diversi
- ♦ aumento della resistenza ai guasti



Su ogni disco è possibile ricavare fino a 4 partizioni *primarie*, oppure 3 primarie ed 1 *estesa.*
All'interno della partizione estesa si possono definire fino a 12 *unità logiche*, con funzionalità analoga a quella delle partizioni primarie.

Partizionamento dei dischi

Linux assegna i nomi delle partizioni secondo uno schema ben definito, pensato per convogliare più informazioni ed essere usato in modo flessibile. L'accesso alle partizioni avviene per mezzo di file contenuti tipicamente nella directory ***/dev/*** , ad esempio */dev/sda1*

I file in /dev/ sono *block special* o *character special*, cioè non sono fie che contengono dati, ma punti di accesso ai device driver del sistema operativo, in particolare i primi adatti per l'accesso a blocchi di dati su dispositivi buffered (come appunto i dischi), i secondi per l'accesso a carattere a dispositivi come le porte ed i bus seriali.

Oltre ai dispositivi fisici, un block device può rappresentare il punto d'accesso anche a dispositivi logici che si comportano allo stesso modo, ad esempio volumi RAID, LVM o di rete.

Partizionamento dei dischi

I dischi fisici sono individuati da un block device file come */dev/xxyn*

xx – Le prime due lettere nel nome della partizione indicano il tipo di disco su cui la partizione risiede, normalmente *hd* (disco IDE) o *sd* (disco SCSI)

y – La terza lettera indica su quale disco è la partizione. I dischi SCSI vanno da /dev/sda in poi in ordine di ID. I dischi IDE sono ordinati in questo modo:

/dev/hda	Disco master sul canale primario del controller.
/dev/hdb	Disco slave sul canale primario del controller.
/dev/hdc	Disco master sul canale secondario del controller.
/dev/hdd	Disco slave sul canale secondario del controller.
...	

N – Il numero finale individua la partizione sul disco. Le prime 4 partizioni (primarie o estesa) sono numerate da 1 a 4, le unità logiche partono da 5.

Partizionamento dei dischi

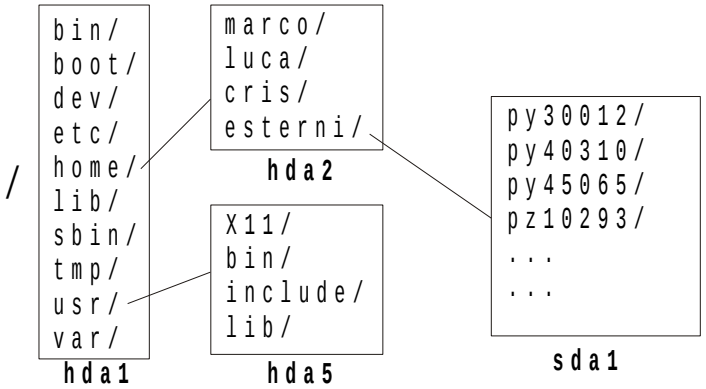
Ogni partizione è etichettata nella *partition table* da un tipo definito da un "numero magico":

Tipo	NM	Tipo	NM	Tipo	NM	Tipo	NM
Empty	00	OS/2 Boot Man	0a	Novell Netw. 386	65	OpenBSD	a6
DOS 12-bit FAT	01	Win95 FAT32	0b	PC/IX	75	NEXTSTEP	a7
XENIX root	02	Win95 FAT32 LBA	0c	Old MINIX	80	BSDI fs	b7
XENIX usr	03	Win95 FAT16 LBA	0e	Linux/MINIX	81	BSDI swap	b8
DOS 16-bit <=32M	04	Win95 Extd (LBA)	0f	**Linux swap**	**82**	Syrinx	c7
Extended	05	Venix 80286	40	**Linux native**	**83**	CP/M	db
DOS 16-bit >=32M	06	Novell?	51	Linux extended	85	DOS access	e1
OS/2 HPFS	07	Microport	52	Amoeba	93	DOS R/O	e3
AIX	08	GNU HURD	63	Amoeba BBT	94	DOS secondary	f2
AIX bootable	09	Novell Netw. 286	64	BSD/386	a5	BBT	ff

Dopo aver realizzato le partizioni desiderate, in ognuna deve essere realizzato un filesystem. Avere partizioni separate consente di utilizzare, se il caso, filesystem differenti, ottimizzati per le diverse parti del sistema. naturalmente il fatto che il software di partizionamento (**fdisk**) di Linux sia in grado di riconoscere tutte queste etichette non implica che il kernel possa accedere ai dati di tutti i corrispondenti filesystem.

Partizionamento dei dischi

Nei sistemi DOS la visione fisica e quella logica dei dischi sono inseparabili: ad ogni partizione è assegnata una lettera di unità (C:, D:, …). Nei sistemi Linux, invece, esiste un unico albero di directory, e le partizioni sono associate ai “rami” dell’albero per mezzo dell’operazione *mount*, che rende disponibili le risorse contenute nella partizione a partire da un *mount point*.



Partizione	Mount point
/dev/hda1	/
/dev/hda2	/home/
/dev/hda5	/usr/
/dev/sda1	/home/esterni/

Partizionamento dei dischi

Quante partizioni fare? Esiste un unico vincolo imposto da certi BIOS (ormai di interesse solo storico, almeno sui PC standard), per cui le informazioni necessarie al boot devono essere:

- ♦ Avendo dischi IDE, su uno dei due dischi del controller primario
- ♦ Avendo dischi SCSI, su uno dei primi due dischi in ordine di ID (0 o 1)
- ♦ Avendo dischi di entrambi i tipi, sul primo disco di ognuno (SCSI 0 o hda)
- ♦ Contenute interamente entro il cilindro 1023 del disco

→ è consigliabile fare una piccola partizione all’inizio del primo disco e montarla su */boot*, che è la directory dove il boot manager cerca i dati per l’avvio ed il kernel da caricare

→ se si vuole utilizzare la memoria virtuale bisogna creare una partizione dedicata allo *swap*

Partizionamento dei dischi

Per il resto si possono seguire due approcci:

- ♦ minimale: unica partizione con spazio sufficiente per tutto, montata su /
- ♦ per funzioni: una partizione per ogni “tipo di accesso”, ad esempio:
  - una partizione per / (obbligatoria)
  - una partizione per */boot*
  - una partizione per */usr* (applicazioni → tipicamente in sola lettura)
  - una partizione per */var* (code e log → alto traffico in lettura e scrittura)
  - una partizione per */home* (aree utente → alto traffico e necessità di *quotas*)

Vediamo nel seguito come FHS organizza i dati nell'albero Unix.

ROOT filesystem

E’ l’origine della gerarchia. Deve contenere tutti i dati necessari all’avvio del sistema, ma essere il più compatto possibile per ridurre i rischi di corruzione accidentale e poter essere alloggiato in media di scarsa capacità (es. floppy). Per rispondere a questi requisiti ed inoltre ospitare i punti iniziali di sottogerarchie più flessibili, deve contenere:

```
/ -- the root directory
       +-bin       Essential command binaries
       +-boot      Static files of the boot loader
       +-dev       Device files
       +-etc       Host-specific system configuration
       +-lib       Essential shared libraries and kernel modules
       +-mnt       Mount point for mounting a filesystem temporarily
       +-opt       Add-on application software packages
       +-sbin      Essential system binaries
       +-tmp       Temporary files
       +-usr       Secondary hierarchy
       +-var       Variable data
```

ROOT filesystem – alcuni componenti

/dev

La directory /dev contiene i file che costituiscono il punto di accesso, per i programmi utente, agli apparati connessi al sistema. Questi file sono essenziali per il funzionamento del sistema stesso.

/etc

La directory /etc è riservata ai file di configurazione locali del sistema. In /etc non devono essere messi eseguibili binari. I binari che in passato erano collocati in /etc devono andare in /sbin o /bin

X11 e **skel** devono essere subdirectories di /etc/

/etc
|- X11
+- skel

La directory X11 è per i file di configurazione del sistema X Window, come XF86Config. La directoy skel è per i prototipi dei file di configurazione delle aree utente.

ROOT filesystem – alcuni componenti

/lib

La directory /lib deve contenere solo le librerie richieste per il funzionamento dei programmi che si trovano in /bin e /sbin.

/proc

La directory /proc contiene file speciali che permettono di ottenere informazioni dal kernel o di inviare run-time informazioni al kernel, e merita di essere esplorata con attenzione.

/sbin

La directory /sbin è riservata agli eseguibili utilizzati solo dall'amministratore di sistema, possibilmente solo quelli necessari al boot ed al mount dei filesystem. Qualunque cosa eseguita dopo che /usr sia stato montato correttamente dovrebbe risiedere in /usr/sbin o in /usr/local/sbin
Come minimo devono essere presenti in /sbin i seguenti programmi:
clock, *getty*, *init*, *update*, *mkswap*, *swapon*, *swapoff*, *halt*, *reboot*, *shutdown*, *fdisk*, *fsck*.*, *mkfs*.*, *lilo*, *arp*, *ifconfig*, *route*

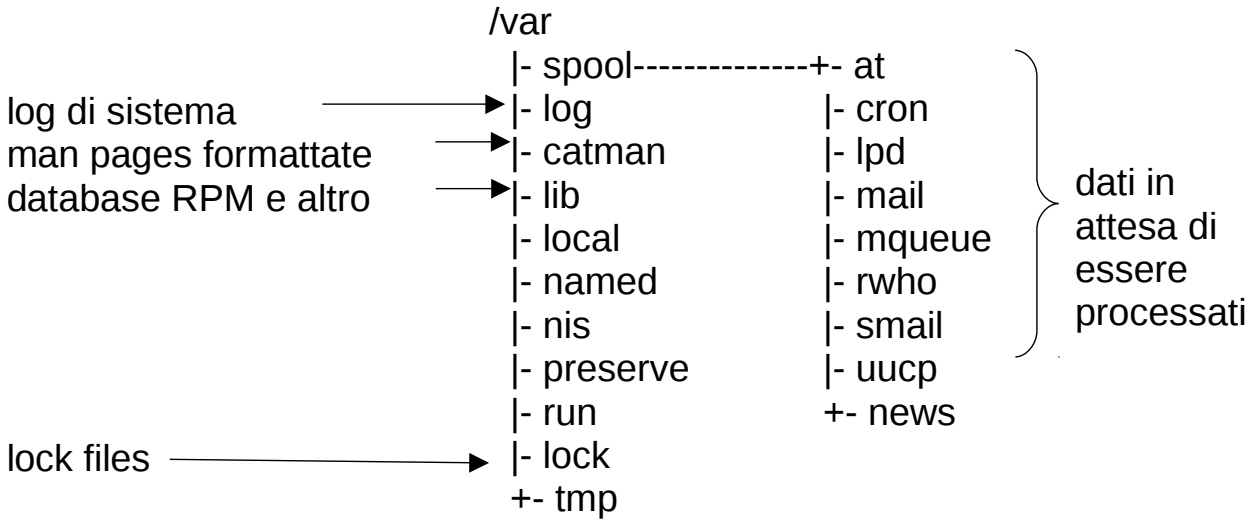
USR filesystem

La directory /usr è per i file condivisibili e statici. Risiede di preferenza su di una propria partizione, e dovrebbe essere montata read-only. Subdirs:

/usr	
|- X11R6	X Window System
|- bin	eseguibili
|- dict	
|- doc,	documentazione diversa dalle man pages
|- etc	file di configurazione validi per il sito
|- games	
|- include	C header files, , , (those that do not belong in /sbin), and
|- info	GNU info files
|- lib	librerie
|- local	
|- man	man pages
|- sbin	programmi per l'amministrazione di sistema
|- share	
+- src	codice sorgente

VAR filesystem

La directory /var è riservata ai file non statici, sia condivisibili che non, ad esempio i file di log, di spool, di lock, di amministrazione e temporanei. Dovrebbe contenere le seguenti subdirs:



Formattazione delle partizioni

All'interno di una partizione deve essere creato un filesystem per mezzo della formattazione, che predispone i metadati necessari all'organizzazione dei file.

Linux conosce una varietà di filesystem diversi, alcuni sviluppati nativamente, altri portati da varianti commerciali di Unix

I più moderni sono *journaled*, ovvero mantengono un log transazionale (il journal, per l'appunto) che registra gli accessi in modo da garantire che a fronte di un crash non si debba esaminare l'intero FS per verificarne l'integrità.

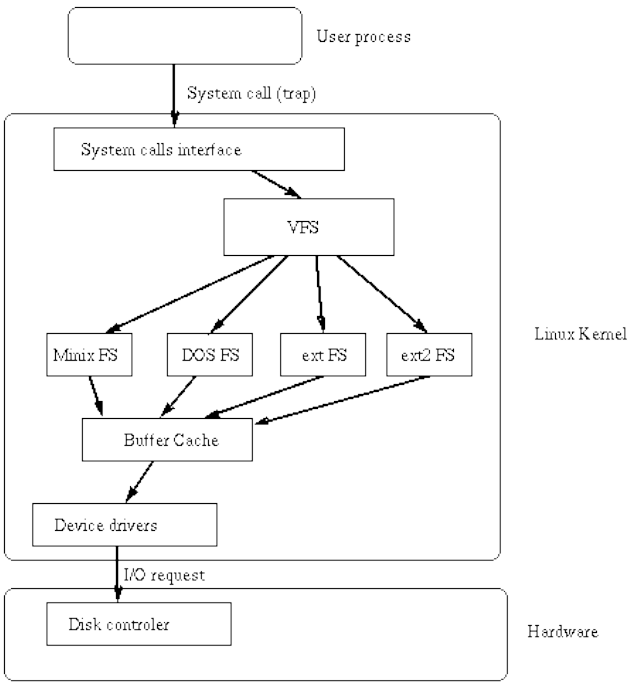
Il comando per la creazione di un FS è tipicamente **mkfs.*tipo***.
Il comando per la verifica dell'integrità di un FS è tipicamente **fsck.*tipo***.

Vediamo l'architettura di Linux a supporto dei filesystem ed i tipi principali.
(immagini tratte da http://web.mit.edu/tytso/www/linux/ext2intro.html)

VFS

Il kernel di Linux contiene uno strato detto Virtual File System

- meccanismo indiretto per accedere ai file
- presenta un'interfaccia indipendente dal filesystem utilizzato sul disp.
- invoca le funzioni proprie della realtà fisica sottostante
  - le operazioni sono suddivise in tre categorie:
  - filesystem ops (es. mount)
  - inode ops (es. unlink)
  - open file ops (es. read)



ext2

Il *second extended filesystem* è il più tradizionale in Linux.

- Non è journaled, e quindi anche se piuttosto robusto (e molto veloce) non è adatto a realizzare FS di grandi dimensioni, poichè il minimo guasto richiederebbe ore per la rilevazione e riparazione.

- Con blocchi di 4KB, le dimensioni massime sono
  - 2TB per i file
  - 16TB per l'intero filesystem

- La struttura è quella del FS Unix, con inodes che supportano fino a due livelli di indirettezza.

- Poichè le directory non sono indicizzate (la ricerca dei file avviene sequenzialmente) è bene non collocare più di 10-15000 file in una directory per non rallentare troppo le operazioni

ext3

ext3 è nato per essere essenzialmente ext2 più un journal, mantenendo la compatibilità col predecessore. Per questo motivo non esibisce le prestazioni dei FS nativamente journaled, ma rispetto a ext2 ha due ulteriori vantaggi significativi:

- la possibilità di crescere, anche a caldo nelle ultime versioni
- la possibilità di indicizzare le directory con htree, e quindi di gestire directory contenenti un maggior numero di file

Il journaling di ext3 può essere regolato su tre livelli:

- *journal* - registra sia i dati che i metadati nel journal prima del commit sul filesystem
- *ordered* - registra solo i metadati, ma garantisce che ne sia fatto il commit solo dopo che i dati corrispondenti sono stati scritti sul FS
- *writeback* - registra solo i metadati senza alcuna garanzia sui dati

http://www.xs4all.nl/~carlo17/howto/undelete_ext3.html

ext4

L'ultima evoluzione del filone “ext”, come sempre nata per offrire nuove feature pur consentendo un certo grado di retrocompatibilità, è stata introdotta in forma stabile nel kernel 2.6.28.

Rispetto ad ext3, aumenta i limiti

- di dimensione dei singoli file da 2 TB a 16 TB
- di dimensione del filesystem da 16 TB a 1 EB (= 1 milione di TB)
- di numero di subdirectory da 32.000 a infinito

Introduce i concetti di:

- *extent* al posto del sistema di allocazione indiretta;
  - i file grandi sono allocati in modo contiguo anzichè essere suddivisi in moltissimi blocchi indirizzati in modo indiretto
- allocazione dei blocchi a gruppi;
  - l'allocatore dei blocchi disco può essere usato per riservarne più di uno per volta
- allocazione ritardata;
  - l'allocatore è invocato solo quando c'è l'effettiva necessità di scrivere su disco
- journal checksumming
  - la consistenza del journal è ottenuta con un checksum invece che con una procedura di commit a due fasi, migliorando affidabilità e velocità

ext4 (continua)

Caratteristiche particolarmente interessanti per i sistemi embedded e real-time sono:

- journal disattivabile
  - elimina la causa delle maggiori lentezze di accesso ai dischi SSD e l'eccesso di scritture concentrate in pochi blocchi (usura);
- inode più grandi
  - come sopra, poichè molti attributi estesi potranno essere ospitati direttamente nell'inode
- inode reservation
  - aumenta il determinismo dei tempi di creazione dei file nelle directory
- persistent preallocation
  - aumenta il determinismo dei tempi di scrittura dei dati in un file

Si deve porre attenzione a due aspetti prima di scegliere ext4:

- l'allocazione ritardata riduce la resistenza agli spegnimenti bruschi
- il bootloader grub non è in grado di accedervi per caricare il S.O., per la partizione di boot è meglio restare su ext3

Altri Journaled FS

- **ReiserFS -** Il primo journaled FS ad essere inserito nel kernel di linux, è teoricamente molto performante su sistemi che gestiscono grandi quantità di file di piccole dimensioni, ma ha attratto critiche in merito alla sua stabilità. Inoltre lo sviluppatore capo ha imposto un modello molto conflittuale nei confronti della comunità, ed ha perso definitivamente credibilità per i suoi guai giudiziari.

- **XFS -** Sviluppato da Silicon Graphics per IRIX e **JFS -** Sviluppato da IBM per AIX, sono stabili, maturi, nativamente a 64bit (quindi su S.O. altrettanto a 64bit possono reggere partizioni da 8 ExaByte) ed offrono diverse ottimizzazioni molto vantaggiose; la loro scarsa diffusione è probabilmente dovuta solo alla consuetudine della maggior parte degli utenti Linux verso ext2/3

Il futuro: btrfs?

Le migliorie introdotte nella serie ext sono di innegabile efficacia, ma il modello di base è rimasto essenzialmente quello di UFS, vecchio di 30 anni.

B-tree FS, abbreviato in **btrfs** e pronunciato “Butter F S”, è un progetto sviluppato da Oracle sotto licenza GPL.

Gli obiettivi sono quelli di integrare funzionalità per la scalabilità enterprise, simili a quelle offerte da ZFS di Sun:

- pooling di risorse HW, multi-device spanning
- copy-on-write (versioning, writable snapshots, uso di supporti RO)
- checksum su dati e metadati
- compressione/deframmentazione/checking on line

Lo stato attuale è ancora di sperimentazione.
Per approfondimenti:
http://lwn.net/Articles/342892/
http://btrfs.wiki.kernel.org/

Filesystem virtuali

Esaminando un tipico sistema Linux si osservano diversi filesystem montati, che non hanno corrispondenza in alcun dispositivo fisico:

```
# mount
...
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
varrun on /var/run type tmpfs (rw,nosuid,mode=0755)
varlock on /var/lock type tmpfs (rw,noexec,nosuid,nodev,mode=1777)
udev on /dev type tmpfs (rw,mode=0755)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=620)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
lrm on /lib/modules/2.6.28-13-generic/volatile type tmpfs
(rw,mode=755)
none on /proc/bus/usb type usbfs (rw,devgid=129,devmode=664)
```

Filesystem virtuali (continua)

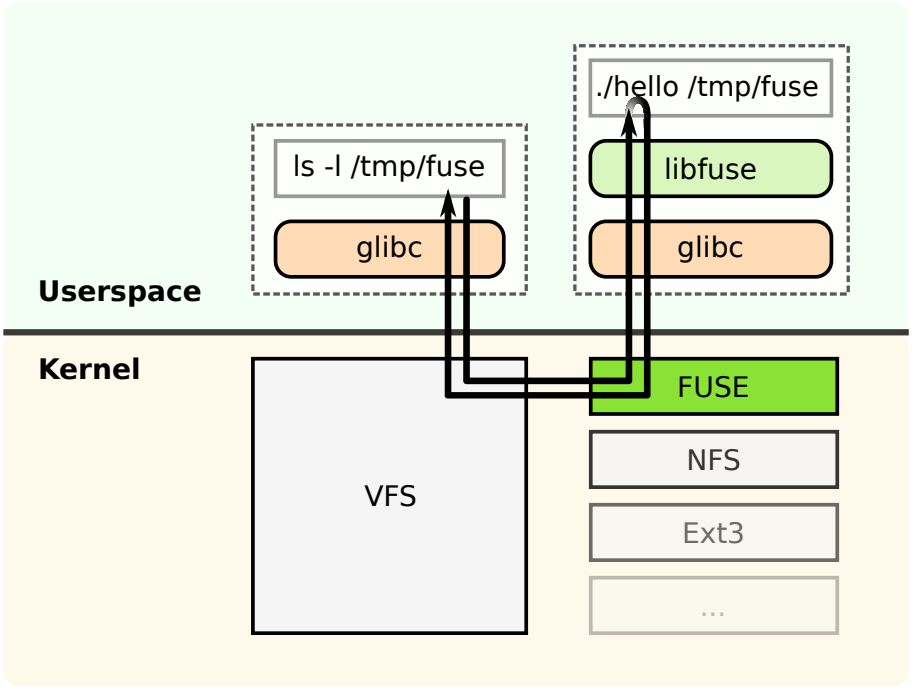
Tra i filesystem virtuali notiamo:

- **proc** e **sys**: permettono l'accesso diretto ai dati del kernel, quali
  - aree di memoria
  - parametri dei processi
  - parametri di configurazione dei moduli
    (vale la pena dare un'occhiata direttamente per rendersi conto di cosa è disponibile)
- **udev:** permette la generazione automatica da parte dei device drivers degli special file per l'accesso ai dispositivi
- **varrun**, **varlock**, **tmpfs**, **lrm**: aree per la mappatura in memoria anzichè su disco di dati volatili

FUSE

Il driver FUSE (Filesystem in Userspace - *http://fuse.sourceforge.net/*) offre un metodo per accedere a filesystem diversi da quelli previsti nel kernel, attraverso un processo che gira in user space, quindi senza modificare il kernel stesso.

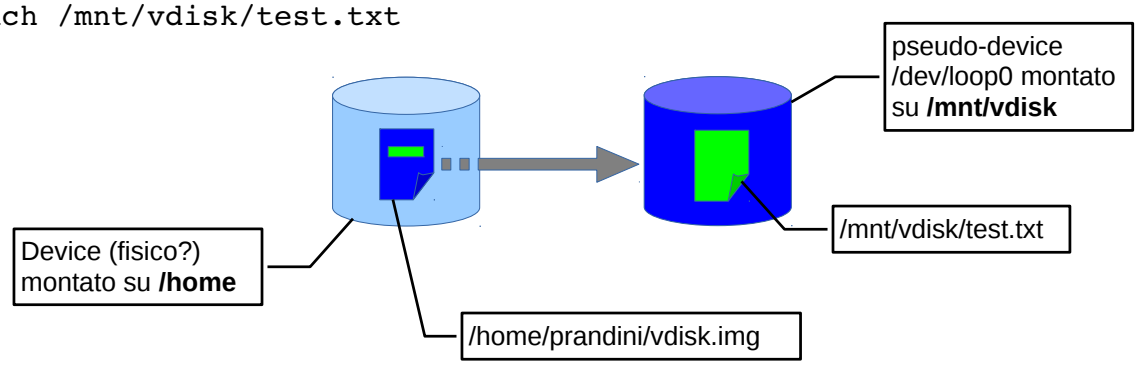
È tipicamente usato per accedere a filesystem che non necessitano di un vero e proprio accesso all'hardware, come quelli di rete.



Loopback devices e mount

Tramite il comando **losetup** è possibile associare uno pseudo-device a blocchi ad un file: in questo modo è possibile utilizzare tutte le system call tipiche per l'accesso ai device a blocchi, che verranno mappate in operazioni di ricerca, lettura e scrittura dei byte del file anzichè in comandi per un vero drive.

```
dd if=/dev/zero of=/home/prandini/vdisk.img bs=1024k count=100
losetup /dev/loop0 /home/prandini/vdisk.img
mkfs.ext3 /dev/loop0
mount /dev/loop0 /mnt/vdisk
touch /mnt/vdisk/test.txt
```



Tuning

Ogni filesystem dispone di numerosi parametri che influiscono sulle prestazioni e sulla robustezza; per la serie ext2/3/4 sono impostabili in uno o più dei modi seguenti:

- alla creazione (**mkfs.ext[234]**)
- al mount (**mount**)
- a run time (**tune2fs**)

Alcuni interventi comuni:

- [dis]attivare il journal con -O [^]has_journal
- [dis]attivare l'indicizzazione ad albero con -O [^]dir_index
- ottimizzare le dimensioni di stride/stripe per device RAID
- disattivare l'aggiornamento dell'access time con -O noatime,nodiratime
- restringere l'utilizzo dei file (es. impedire la collocazione di device files sul filesystem)
- abilitare estensioni per la sicurezza (cifratura, acl, bit speciali)

Mount automatico delle partizioni

L'associazione tra partizione e mount point è mantenuta nel file */etc/fstab* perchè all'avvio del sistema il filesystem possa essere automaticamente predisposto

```
# <file system> <mount point>   <type>  <options>    <dump>  <pass>
/dev/sda1  /      ext3    defaults,errors=remount-ro,relatime  0 1
/dev/sda4  /home  ext3    defaults,relatime                    0 2
```

La regola AAA

- **Autenticazione**: attribuzione certa dell'identità di un soggetto che utilizza le risorse
  - Normalmente include una *identificazione* preliminare
  - È una separazione importante!
  - Errore comune usare elementi *identificativi* come "segreti" che supportano una *autenticazione* solo perché sembrano "oscuri"
- **Autorizzazione**: verifica dei diritti di un soggetto di compiere una determinata azione su di un oggetto
  - decisione esplicita di concessione o negazione del permesso
- **Auditing**: tracciamento affidabile delle decisioni (tutte) di autenticazione e autorizzazione
  - verifica dell'efficacia delle politiche
  - compromesso difficile sui dettagli tra utilità e usabilità

Autenticazione

- L'autenticazione è basata sull'utilizzo di uno di questi fattori - qualcosa che solo l'utente:
  - **Conosce** (ad es. Password, PIN, risposta segreta)
  - **Possiede** (ad es. Carta bancomat, telefono cellulare, hard token, Yubikey)
  - **È (fisicamente)** (ad es. Biometrico: iride, impronta digitale, ecc.)
  - **È (posizione)** (ad es. GPS, geolcation, Centralizzata Auto, Allarme in casa.)
- Soffermiamoci sui modi che un **Prover** ha per dimostrare a un **Verifier** di essere a conoscenza di un segreto

Autenticazione passiva

- Autenticazione *passiva*
  - P e V concordano il segreto e lo memorizzano
  - P invia il segreto a V per dimostrare di conoscerlo
  - V lo confronta con la copia in suo possesso per autenticare P
- Problemi di comunicazione
  - invio in chiaro → intercettazione da parte di attaccante passivo
  - invio offuscato ma sempre uguale → replay attack

  servono protocolli di autenticazione più sofisticati
  - ... oppure un canale cifrato bene
- Problemi di memorizzazione
  - furto da V

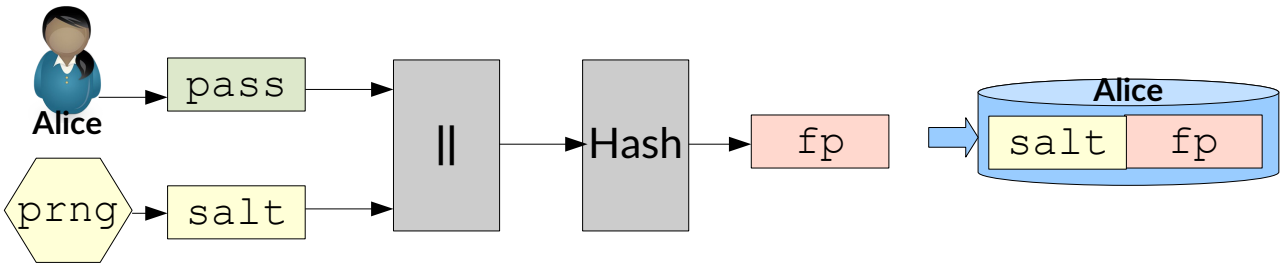
Memorizzazione delle password

- Requisiti
  - V non deve conoscere le password
  - Il furto di file da V deve essere inefficace
  - V deve discriminare una password corretta da una errata

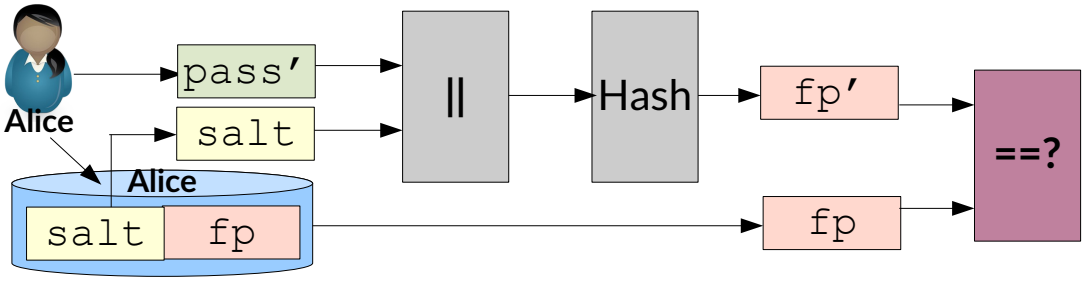
  → non si memorizza la password ma la sua impronta, calcolata con una funzione hash
- Problemi
  - attacco con dizionario
    - vedi approfondimento su Rainbow Tables
  - stessa password su macchine diverse
- Mitigazione: salt
  - variazione random inserita alla scelta della password

Salt

- Scelta della password



- Verifica della password



Salt

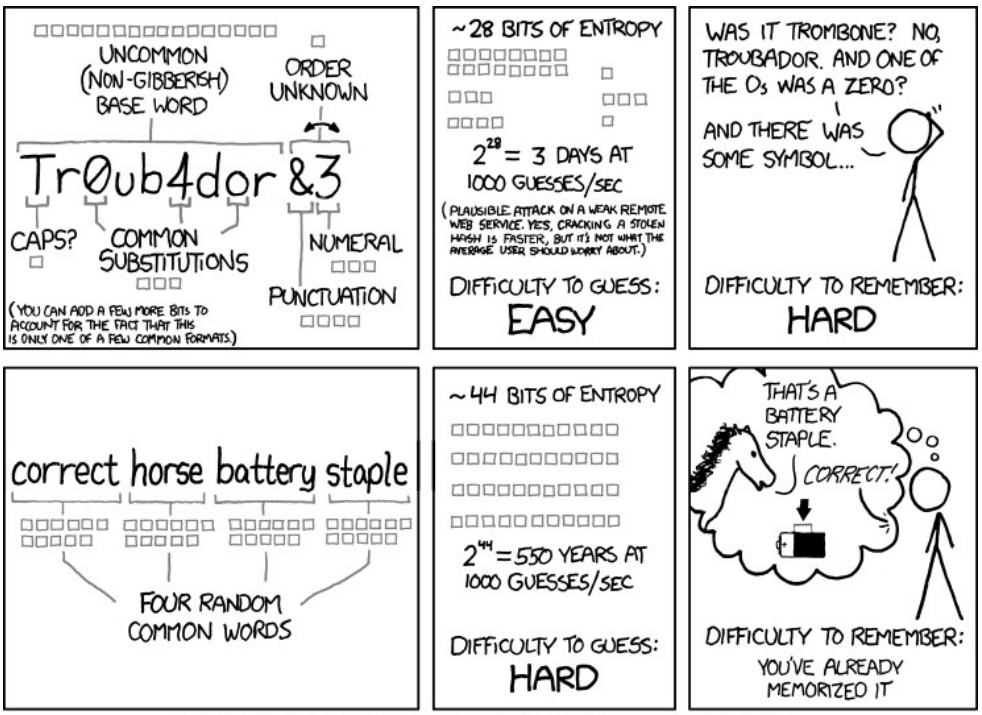
- Esempio da /etc/shadow

6ViDM2ltuaSNPBxfO$Ip40UoquO.OiFafqlVeMqzZplisBV.CC76j
Ryead9rvidbyWcAr200dH.7N8budnpS3FzJJdDxrKGQjekwTFU0

identificatore dell'algoritmo hash usato
salt
fingerprint calcolato su concatenazione pass||salt

- a ogni scelta o rinnovo della password, il salt cambia
  - stessa pass, fingerprint diverse
  - non permette di precalcolare le fingerprint partendo da un dizionario
- è inefficace contro attacchi offline, cioè avviati dopo aver sottratto il file delle password
  - la contromisura essenziale per questi è che la password non sia facile da indovinare

Scelta delle password

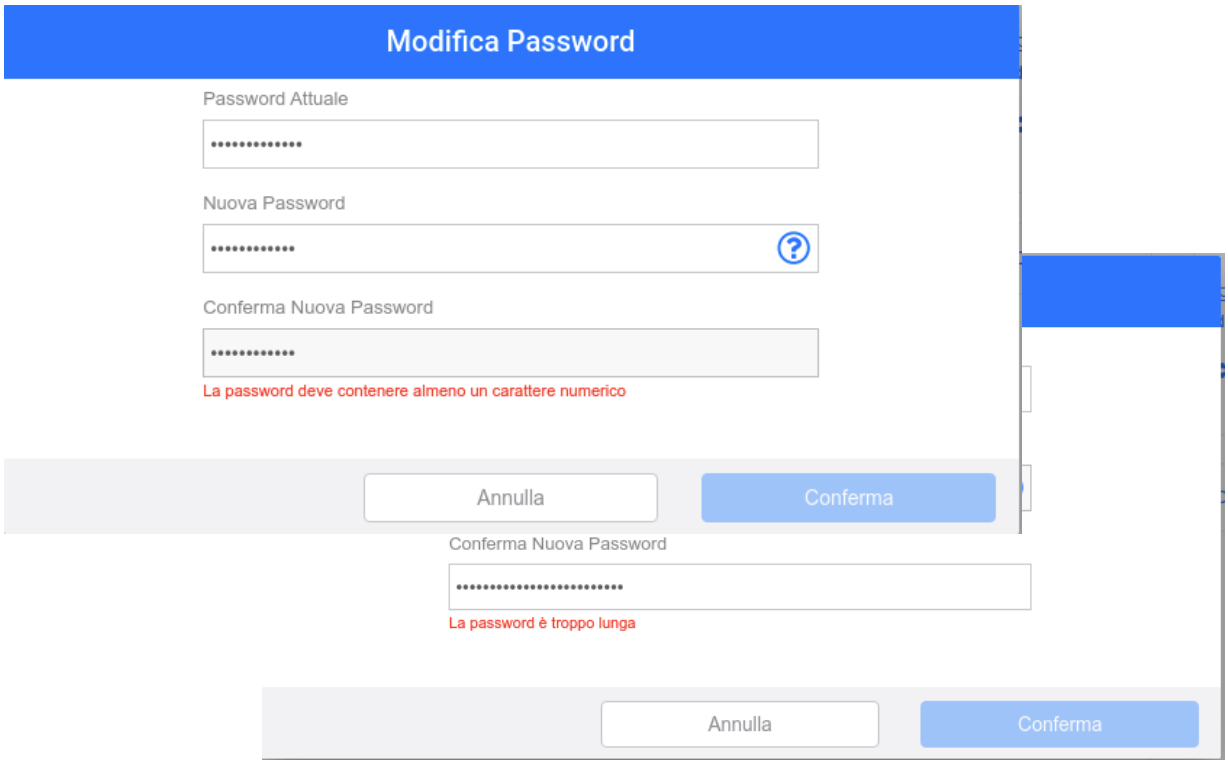


- http://xkcd.com/936

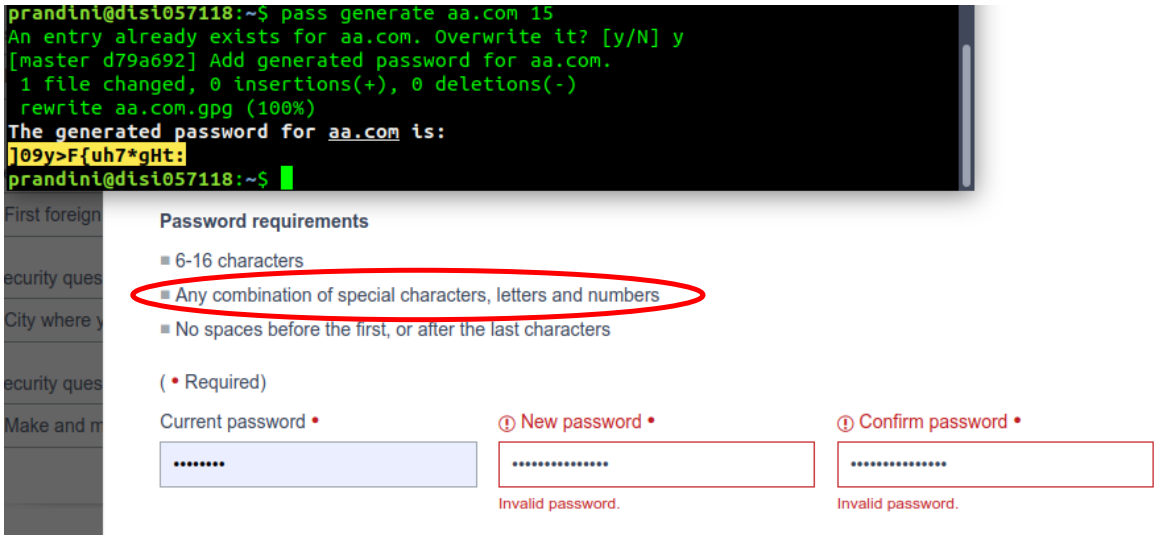
Protezione delle password

- In ogni caso, una password scoperta grazie a un lungo lavoro di cracking su di un *leak* da un sistema, sarà usabile ovunque
- Bisogna usare password diverse a prescindere
  - decine? centinaia??
  - *password manager*
    - database cifrato con passphrase
    - quella va scelta bene
- Rischio mitigato da sistemi a più fattori
  - vedi seguito

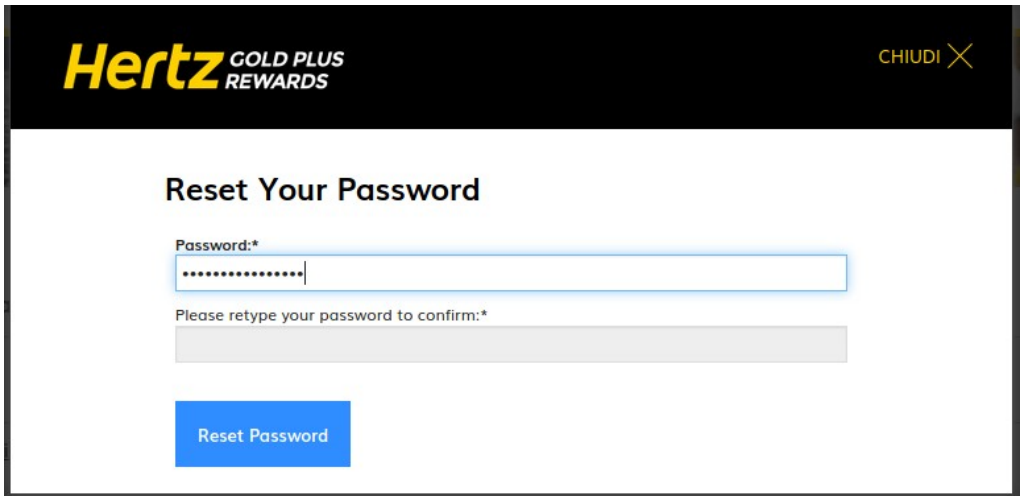
Cattive pratiche



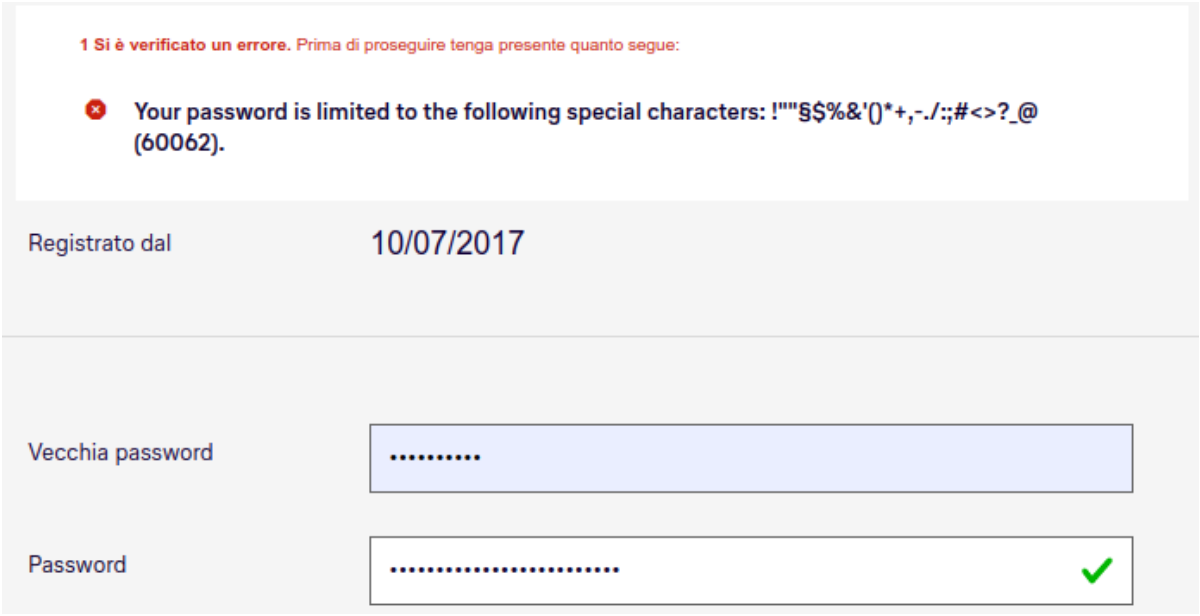
Cattive pratiche



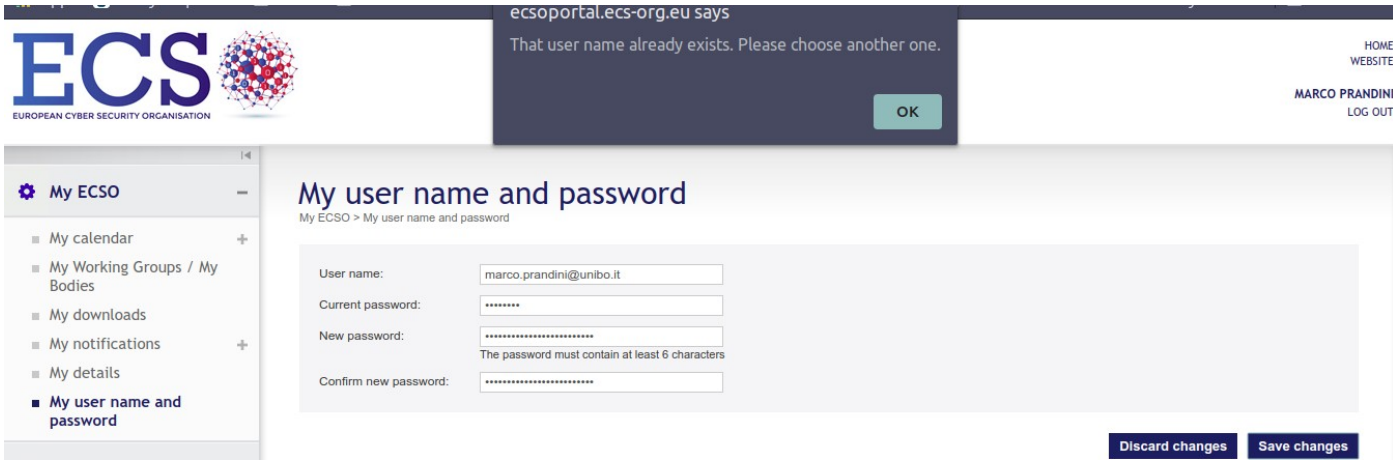
Cattive pratiche



Cattive pratiche



Cattive pratiche (collaterali)

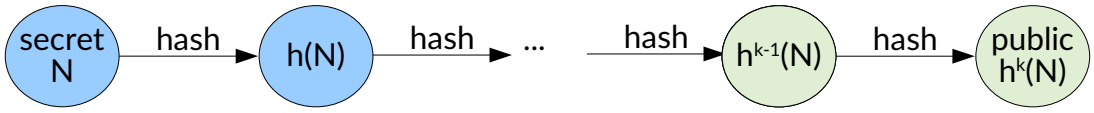


Autenticazione attiva

- P convince V di possedere il segreto autentico <u>senza svelarlo</u> e mandando <u>ogni volta un dato diverso</u>
  - Il furto del dato di confronto da V è intrinsecamente inutile
  - Il furto dal canale è inutile per autenticazioni future
  - attenzione comunque all'uomo nel mezzo!

S-KEY One-Time Password

- P conosce il proprio segreto N
- V viene inizializzato col risultato dell'applicazione ripetuta k volte di una funzione hash a N: $h^k(N)$

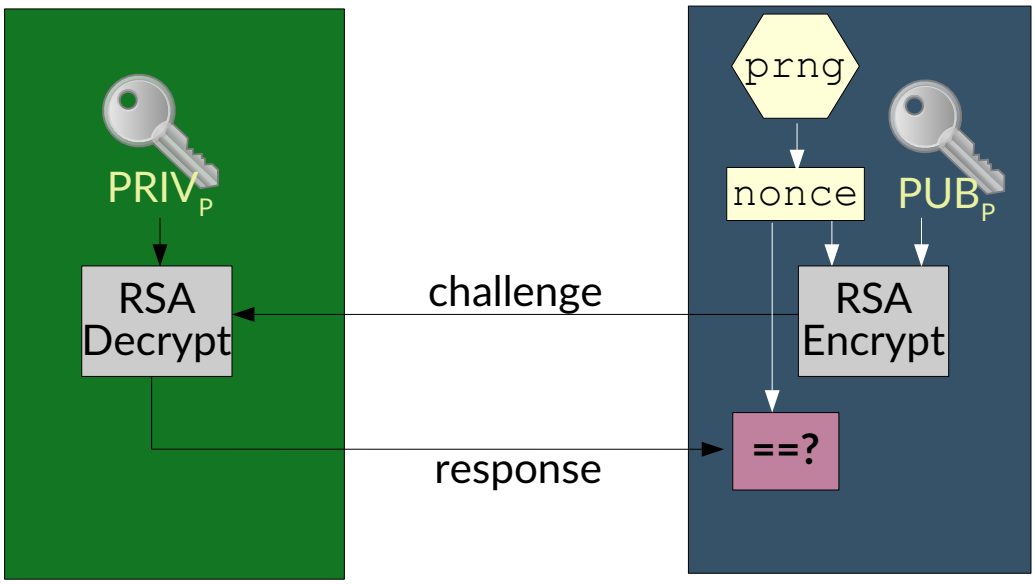


- Alla prima autenticazione P invia $h^{k-1}(N)$
  - V verifica facilmente che $h(h^{k-1}(N)) = h^k(N)$
  - V scarta $h^k(N)$ e ricorda $h^{k-1}(N)$ come riferimento per la prossima autenticazione

- L'hash è
  - facile da calcolare → efficiente
  - difficile da invertire → sicuro
- Il sistema va però reinizializzato dopo k passi
  - ci sono varianti senza limiti

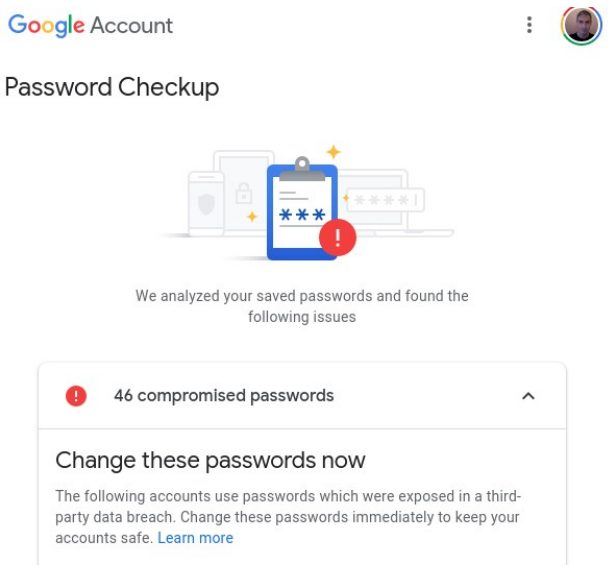
Sistemi a sfida e risposta

- Tipicamente utilizzati con crittografia asimmetrica
- P può provare il possesso di una chiave privata senza svelarla, se V possiede la chiave pubblica



2FA

- **Le credenziali vengono comunemente rubate tramite:**
  - Attacchi di phishing mirati.
  - Siti di terze parti compromessi con stesso nome utente / password utilizzati
  - Esempio Leak. Haveibeenpwned?
- **L'autenticazione a due fattori consiste nell'utilizzo di almeno DUE dei TRE(quattro?) fattori precedenti.**
- **L'autenticazione a due fattori aggiunge un ulteriore livello di autenticazione che impedisce agli aggressori di accedere a un account anche se ottengono le credenziali.**



2FA vs 2SA (Two steps autentication/verification)

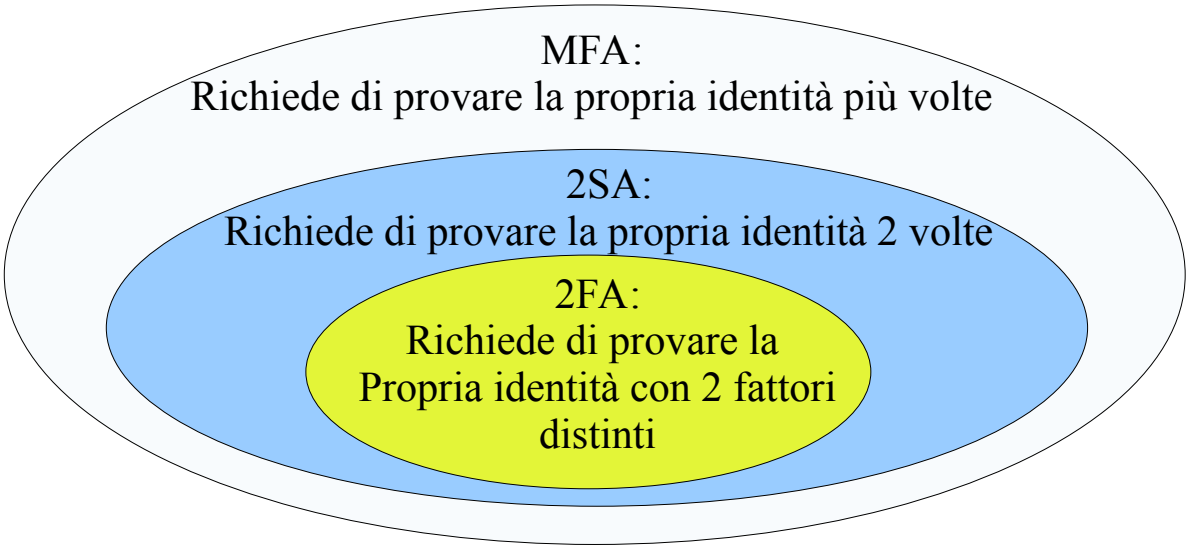
- **È importante che per fare la 2FA i fattori di autenticazione siano DISTINTI**
- **Al giorno d'oggi infatti la 2FA viene erroneamente confusa con la two-steps, dove il livello aggiuntivo di autenticazione non è riconosciuto come distinto.**
- **Ad esempio un codice di verifica inviato per SMS o per mail in aggiunta ad una password non è considerato qualcosa che si POSSIEDE (in aggiunta a qualcosa che sa) perché l'SMS è la mail sono "facilmente" oggetto di attacchi MITM**

2FA vs 2SA (Two steps autentication/verification)

- Implementare 2SA è comunque meglio di non implementarla, però è bene distinguerne le differenze con la 2FA
- Il device per l'autenticazione aggiuntiva deve essere dedicato solo a quello scopo. (Il telefono può essere violato da remoto invalidando il concetto del "possiede")
- Sbloccare un'app con un PIN per avere un token di accesso aggiuntivo è considerato come "conoscenza aggiuntiva" per cui non un fattore di autenticazione aggiuntivo

2FA vs 2SA vs MFA

- È infine possibile considerare anche i casi dove più di due fattori, anche distinti, vengano usati per l'autenticazione, in quel caso parliamo Multi-factor Autentication



ESEMPI: OTP

- OTP (One Time Password) la password aggiuntiva, sottoforma di token è valida solo per un utilizzo.

Source: times.com

ESEMPI: TOTP

- TOTP (Time One Time Password) la password aggiuntiva, sottoforma di token è valida solo per un utilizzo ed è limitata nel tempo.
  Il tempo può variare dai 5 secondi a pochi minuti.

Source: ftsafe.com

Il processo di controllo dell'accesso

- Un soggetto autenticato deve essere autorizzato a svolgere operazioni sulle risorse del sistema
- Tre passaggi:
  - definire il modello del sistema controllato
    (limitatamente ai fattori critici per il controllo degli accessi)
  - definire la politica di accesso
    (le regole in base alle quali l'accesso è regolamentato)
  - attuare la politica
    (tramite opportuni meccanismi HW / SW)
- Come già detto è molto utile separare politiche e meccanismi
  - per confrontare diverse politiche senza essere sommersi dai dettagli di implementazione
  - per modellare i componenti al livello di astrazione più appropriato e identificare la serie minima di requisiti che qualsiasi sistema di controllo degli accessi dovrebbe rispettare
  - per progettare meccanismi come elementi costitutivi, utilizzabili per diversi tipi di politiche

Caratteristiche delle politiche

- Principio del privilegio minimo
  - qualsiasi accesso deve avvenire concedendo l'insieme di autorizzazioni più ristretto possibile
- Consistenza
  - deve esistere uno schema di risoluzione non ambiguo da impiegare quando è possibile applicare autorizzazioni diverse alla stessa richiesta di accesso
  - nessuna soluzione unica!
    - la regola più / meno specifica vince
    - default allow vs. default deny
    - vince la prima / ultima regola incontrata in ordine spazio / temporale
  - gerarchia degli autori delle regole
- Completezza e correttezza
  - qualsiasi richiesta di accesso deve ricevere risposta entro un limite di tempo predeterminato
  - ci deve essere una regola predefinita da applicare quando non è possibile trovare un'autorizzazione esplicita per una richiesta

Caratteristiche dei meccanismi

- Resistenza alle manomissioni
  - deve essere impossibile sabotare un meccanismo di controllo degli accessi senza che nessuno se ne accorga
- Principio di mediazione completa
  - ogni accesso alle risorse deve essere sottoposto al controllo e alla decisione del meccanismo
- Piccolo e autonomo
  - deve essere facile da testare e riparare
- Ragionevolmente economico
  - il suo costo non deve superare i danni causati da accessi non autorizzati

Parametri di decisione

- Identità del soggetto
  - ovvio
- Ruolo del soggetto
  - i soggetti possono assumere ruoli diversi
  - le decisioni di accesso vengono prese in base al ruolo attuale di un soggetto, indipendentemente dalla sua identità
- Modalità di accesso
  - la decisione è presa non solo in base all'identità / dal ruolo, ma anche al tipo di operazione che il soggetto vuole eseguire
- Vincoli spaziali e temporali
  - l'accesso può dipendere da dove si trova il soggetto e da quando viene effettuata la richiesta
- Storia delle attività svolte
  - possono essere imposti limiti alla quantità di attività di un soggetto e al tipo di utilizzo della risorsa

Modelli di controllo dell'accesso

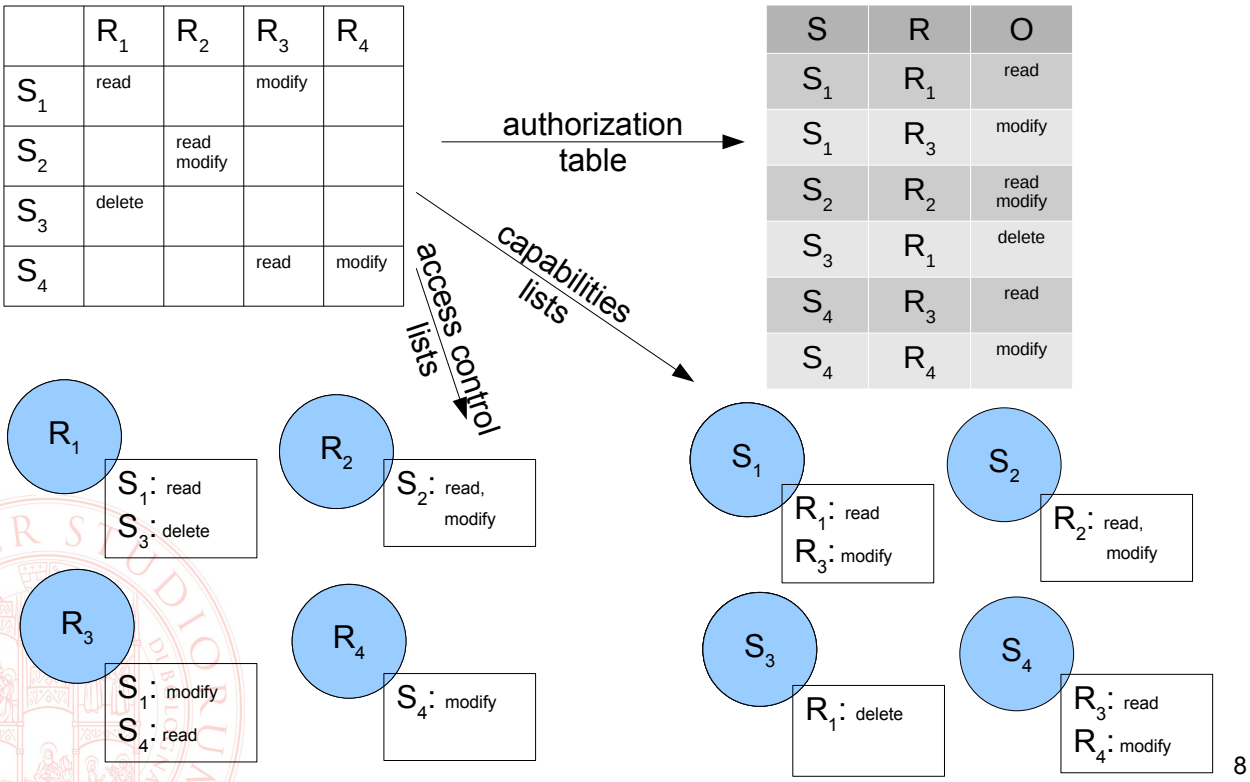
- I due paradigmi fondamentali sono
  - DAC (Discretionary access control):
    - Ogni oggetto ha un proprietario
    - Il proprietario decide i permessi
  - MAC (Mandatory access control):
    - La proprietà di un oggetto non consente di modificarne i permessi
    - C'è una policy centralizzata decisa da un *security manager*
- Ci sono modelli più complessi
  - RBAC (Role-based access control):
    - I permessi sono assegnati ai *ruoli*
    - Utile se i soggetti possono assumere dinamicamente ruoli differenti a seconda del contesto (cosa devono fare, dove si trovano, in che tempi operano…)
  - e varianti...

Generalità sui meccanismi

- In principio, il controllo dell'accesso è decidere se un ***soggetto*** può eseguire una specifica ***operazione*** su di un ***oggetto***
- Il modo più banale per esprimere i *permessi* sarebbe una matrice completa
  - Migliaia di soggetti, milioni di oggetti!
  - La maggior parte delle "celle" è sempre al valore di default → potrebbe essere omessa

Subject / Object	User1	User2	Group3	...	...
File1	read	read write	--	...	...
Dir2	list	modify	--	...	...
Socket3	write	--	read	...	...
...	...	...	...	...	...
…	...	...	...	...	...

Implementazioni efficienti



Implementazioni comuni

- Partizionare la matrice per soggetto: *capability lists*
  - Una lista associata a ogni soggetto del sistema
  - Contiene solo gli oggetti su cui il soggetto ha permessi ≠ default
- Partizionare la matrice per oggetto: *access control lists* (ACL)
  - Una lista associata a ogni oggetto del sistema
  - Contiene solo i soggetti che hanno permessi ≠ default sull'oggetto
  - Esplicitamente implementata da POSIX e MS Windows
  - Il filesystem Unix tradizionale ha negli inode una ACL "rigida", che elenca sempre e solo tre soggetti:
    - L'utente proprietario (U)
    - Il gruppo proprietario (G)
    - Il gruppo implicito che contiene tutti gli utenti ≠ U e $\notin$ G

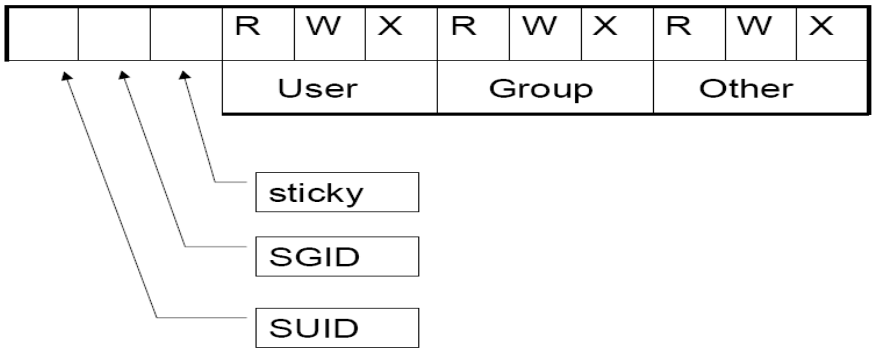
    e i relativi permessi

DAC nei sistemi Linux

- Gli utenti/gruppi possono essere creati con tool grafici o a riga di comando
  - `adduser`, `addgroup`
- Ogni utente DEVE appartenere almeno a un gruppo
  - Normalmente il sistema ne crea uno omonimo, con solo l'utente dentro
- Ogni utente può appartenere a un numero variabile di altri gruppi
- Gli account utente possono essere in uno stato *locked*, che impedisce di usarli per l'accesso interattivo, ma consente ai processi di girare con tale identità
  - Minimo privilegio!
- Il comando *passwd* si usa
  - Per cambiare le password (proprie, salvo root che può cambiarle a tutti)
  - Per settare l'account allo stato lock (`-l`) o unlock (`-u`)
    - Ovviamente solo root può farlo

Autorizzazioni su Unix Filesystem

- Ogni file (regolare, directory, link, socket, block/char special) è descritto da un i-node
- Un set di informazioni di autorizzazione, tra le altre cose, è memorizzato nell'i-node
  - (esattamente un) utente proprietario del file
  - (esattamente un) gruppo proprietario del file
  - Un set di 12 bit che rappresentano permessi standard e speciali



Significato dei bit di autorizzazione

- Leggermente diverso tra file e directory, ma in gran parte deducibile ricordando che
  - Una directory è semplicemente un file
  - Il contenuto di tale file è un database di coppie (nome, i-node)

R = read (lettura del contenuto)

Lettura di un file

Elenco dei file nella directory

W = write (modifica del contenuto)

Scrittura dentro un file

Aggiunta/cancellazione/rinomina di file in una directory

X = execute

Esegui il file come programma

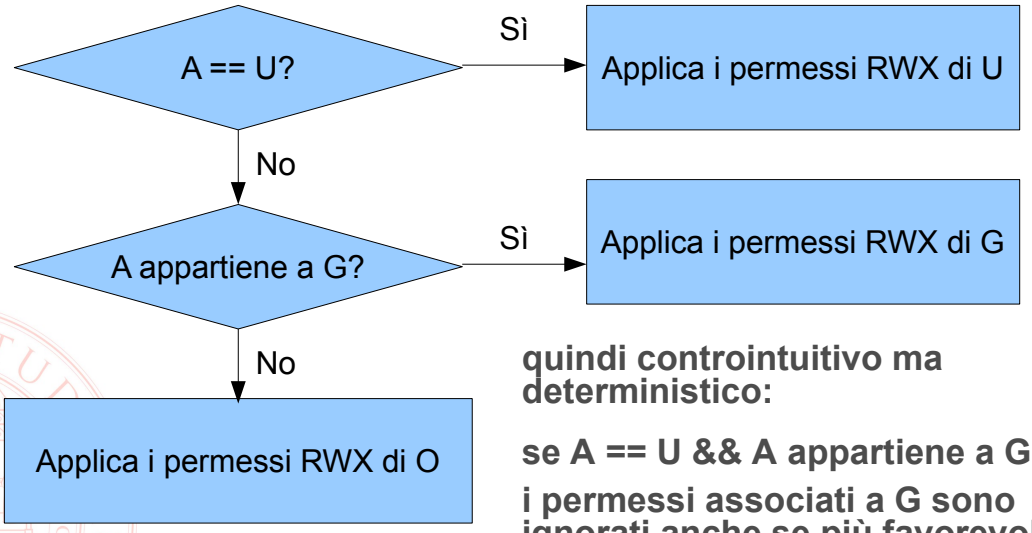
Esegui il lookup dell'i-node nella

NOTA che il permesso 'W' in una directory consente a un utente di cancellare file sul contenuto dei quali non ha alcun diritto

NOTA: l'accesso a un file richiede il lookup di tutti gli i-node corrispondenti ai nomi delle directory nel path → serve il permesso 'X' per ognuna, mentre 'R' non è necessario

Composizione dei permessi

- Quando un utente "A" vuole eseguire un'operazione su di un file, il sistema operativo controlla i permessi secondo questo schema:



quindi controintuitivo ma deterministico:

se A == U && A appartiene a G

i permessi associati a G sono ignorati anche se più favorevoli

Controllo dei permessi predefiniti

- Servono automatismi per assegnare i permessi alla creazione
- Ownership
  - l'utente creatore è assegnato come proprietario del file
  - Il gruppo attivo dell'utente creatore è assegnato come gruppo proprietario
    - Default = gruppo predefinito, da */etc/passwd*
    - L'utente lo può cambiare a mano nella sessione con **newgrp**
    - Può cambiare automaticamente nelle directoy con SGID settato
- Permessi = "tutti quelli sensati" tolta la umask
  - "tutti quelli sensati"
    - `rw-rw-rw-` (666) per i file, l'eseguibilità è un'eccezione
    - `rwxrwxrwx` (777) per le directory, la possibilità di entrarci è la regola
    - la *umask* quindi può essere unica: una maschera che toglie i permessi da non concedere
  - poiché in Linux il gruppo di default group di un utente contiene solo l'utente stesso, una umask sensata è 006 (toglie agli "other" lettura e scrittura)
    - È un settaggio utile per collaborare, crea file manipolabili da tutti i membri del gruppo, a patto che questo sia settato correttamente
  - col comando **umask** si può interrogare e settare interattivamente, per rendere persistente la scelta si usano i file di configurazione della shell

Bit speciali / per i file

I tre bit più significativi della dozzina (11, 10, 9) configurano comportamenti speciali legati all'utente proprietario, al gruppo proprietario, e ad altri rispettivamente

- BIT 11 – SUID (Set User ID)
  - Se settato a 1 su di un programma (file eseguibile) fa sì che al lancio il sistema operativo generi un processo che esegue con l'identità dell'utente proprietario del file, invece che quella dell'utente che lo lancia
- BIT 10 – SGID (Set Group ID)
  - Come SUID, ma agisce sull'identità di gruppo del processo, prendendo quella del gruppo proprietario del file
- BIT 9 – STICKY
  - OBSOLETO, suggerisce al S.O. di tenere in cache una copia del programma

Permessi delicati da tenere sotto controllo

- SUID e SGID sono un modo efficace di implementare interfacce per utenti standard verso processi privilegiati
  - Cambio password: guardare /usr/bin/passwd
  - Pianificazione di attività: guardare /usr/bin/crontab e /var/spool/cron/
  - etc.
- I programmi con questi permessi vanno sorvegliati, perché chiunque li lanci acquisisce temporaneamente privilegi elevati
  - Pochi programmi e molto vincolati
  - Rischi: bug di questi programmi che porti a eseguire operazioni arbitrarie invece di quelle progettate, programmi diversi a cui sono dati per errore questi privilegi
- Usare find per trovarli
  - `find / -type f -perm +6000`
- Altre ricerche interessanti per la sicurezza
  - file world-writable (`-perm +2`)
  - file senza proprietario, rimasti da account cancellati (`-nouser`)

Bit speciali / per le directory

- Bit 11 per le directory non viene usato
- Bit 10 – SGID
  - Precondizioni
    - un utente appartiene (anche) al gruppo proprietario della directory
    - il bit SGID è impostato sulla directory
  - Effetto:
    - l'utente assume come gruppo attivo il gruppo proprietario della directory
    - I file creati nella directory hanno quello come gruppo proprietario
  - Vantaggi (mantenendo umask 0006)
    - nelle aree collaborative il file sono automaticamente resi leggibili e scrivibili da tutti i membri del gruppo
    - nelle aree personali i file sono comunque privati perché proprietà del gruppo principale dell'utente, che contiene solo l'utente medesimo
- Bit 9 – Temp
  - Le "directory temporanee" cioè quelle world-writable predisposte perché le applicazioni dispongano di luoghi noti dove scrivere, hanno un problema: chiunque può cancellare ogni file
  - Questo bit settato a 1 impone che nella directory i file siano cancellabili solo dai rispettivi proprietari

Attributi

- Gli attributi sono primariamente utili per il fs tuning
  - compressed (c), no dump (d), extent format (e), data journalling (j), no tail-merg-ing (t), undeletable (u), no atime updates (A), synchronous directory updates (D), synchronous updates (S), and top of directory hierarchy (T)
- Alcuni sono rilevanti per la sicurezza
  - append only (a) – utile per impedire il taglio dei logfile
  - immutable (I) – vieta cancellazione, creazione di link verso il file, rinomina e scrittura, utile per i file di sistema
  - secure deletion (s) – sovrascrive con zeri i blocchi dei file cancellati (sicurezza molto limitata ma valida contro strumenti in linea)
- Tools
  - **chattr** per modificarli
  - **lsattr** per visualizzarli

POSIX Access Control Lists

- Le ACL estendono la flessibilità di autorizzazione
- Vantaggi:
  - Specificare una lista arbitraria di utenti e gruppi coi relativi permessi (comunque scelti tra rwx) in aggiunta agli owner
  - Ereditare la maschera di creazione dalla directory
  - Limitare tutti i permessi simultaneamente (esempio mask sotto)
- Esempio:

```
user::rw-
user:lisa:rw-          #effective:r--
group::r--
group:toolies:rw-      #effective:r--
mask::r--
other::r--
```

- Strumenti:
  - `setfacl` per impostare, `getfacl` per visualizzare le ACL
    (ls -l mostr un '+' dopo i permessi se ACL è presente per un file)
  - `man acl`

Il super-utente nei modelli DAC

- Esiste tipicamente un utente con privilegi illimitati, che può scavalcare i meccanismi di controllo dell'accesso
  - *root* in Unix
  - *administrator* in Windows
- L'account va difeso contro ingressi abusivi ma va anche minimizzata la probabilità di fare errori
  - Usare un account non-privilegiato, basta per il 99% del tempo
  - Disabilitare l'accesso diretto da GUI e console
  - Ottenere temporaneamente i diritti di super-utente solo per eseguire i task di amministrazione
    - *sudo* in Linux
    - "*esegui come amministratore*" in Windows

Capabilities in Linux

(da non confondere con le capability list tipiche dei modelli MAC)

- I poteri di *root* non sono "monolitici"
- Ci sono ben 41 diverse *capability* (al kernel 5.9)
  - rappresentano autorizzazioni normalmente negate agli utenti standard
  - riguardano molteplici aspetti di controllo delle risorse di calcolo e dei processi e dell'accesso alla rete
  - una nello specifico è CAP_DAC_OVERRIDE: la possibilità di ignorare i permessi sul filesystem
- È possibile assegnare specifiche capability a processi lanciati da utenti standard
  - autorizzazione a svolgere azioni privilegiate senza accesso a root
  - implementazione del principio di minimo privilegio
- `man (7) capabilities (8) getcap (8) setcap`

DAC nei sistemi Microsoft

- Utenti e loro proprietà
- Tipi di gruppi
- Estensione dei gruppi sui domini
- Gruppi predefiniti
- Generalità NTFS
- Implementare la sicurezza di NTFS
- Implementare la condivisione di risorse
- Permessi locali e permessi sulle condivisioni NTFS

Premessa: i domini

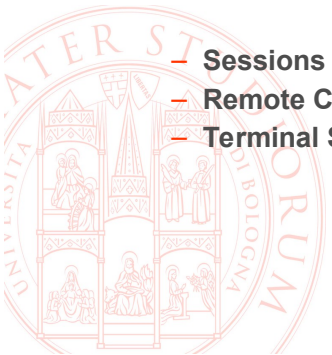
- Nell'uso più comune, i sistemi Microsoft sono raggruppati in un *dominio*: un insieme di computer, comunicanti tra loro e che condividono un directory database comune
- Nella directory sono memorizzati vari tipi di oggetti
  - I computer che fanno parte del dominio
  - Le risorse condivise dai computer (cartelle, stampanti)
  - Gli utenti validi sul dominio
  - I gruppi di utenti
  - I raggruppamenti di altre entità
  - ...

Utenti

- Local user accounts
  - ristretti al sistema su cui sono creati
  - possono avere moderati permessi amministrativi (che non si estendono alla possibilità di accedere ai dati di altri utenti) --> Power Users Group
- Domain user accounts
  - appartiene ad un dominio
  - profilo memorizzato in AD
  - può accedere a risorse non locali, limitatamente ai privilegi che gli sono concessi
    - del proprio dominio
    - dei domini trusted

Proprietà dell'utente

- Sono moltissime,accessibili dai *tab* del wizard qui elencati:
  - Member Of — The user's defined group membership
  - Dial-in — Remote access and callback options
  - General — User's first name, last name, display name description, office location, telephone, e-mail, and Web pages
  - Address — User's post office mailing address
  - Account — Logon name, domain, logon hours, logon to server name, account options, and account expiration date
  - Profile — User profile path, profile script, home directory path and server, and shared document folder location
  - Telephones/Notes — Home, pager, and mobile phone numbers and comments on where to contact user
  - Organization — Job title, company, department, manager, and people who report to user
  - Environment — Applications to run from Terminal server client
  - Sessions — Timeouts for Terminal Services
  - Remote Control — Permissions for monitoring Terminal Service sessions
  - Terminal Service Profile — Location for Terminal Service home directory



Gruppi

- Ogni oggetto di AD può essere membro di uno o più gruppi (di tipo e scope appropriato)
- Distribution Groups
  - possono essere usati da qualsiasi applicazione abbia bisogno di una lista di utenti
  - il sistema operativo non li utilizza
    - non appesantiscono il logon ticket dell'utente
- Security Groups
  - come i DG, ma possono essere soggetti nelle regole che controllano l'accesso alle risorse del sistema
- In Windows 2003 funzionante in Native Mode è possibile la conversione da un tipo all'altro

Group scopes

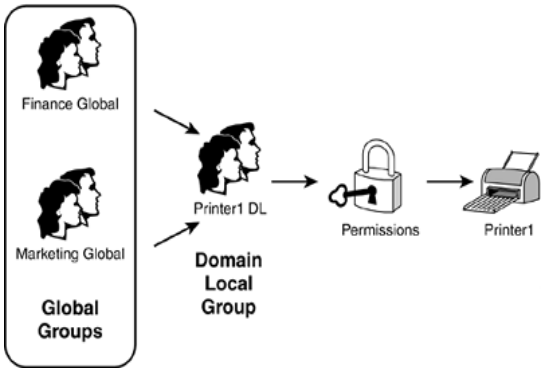
- Sia per i Distribution Group che per i Security Group vale il concetto di *scope* (estensione), che definisce
  - oggetti di quali domini possono far parte del gruppo
  - in quali domini può essere usato un gruppo per definire regole d'accesso
- La prima suddivisione è tra
  - Machine Local (locali ad una singola macchina)
  - Gruppi validi nel dominio
    - Domain Local
    - Global
    - Universal
- Nesting: è possibile *solo in native mode* rendere gruppi membri di altri gruppi

Group scopes

	Può contenere	Può essere membro di	Gli possono essere assegnati permessi su
Domain Local Group (DLG)	Utenti, GG, UG, computer di qualsiasi dominio, DLG dello stesso dominio	Altri DLG dello stesso dominio	Risorse dello stesso dominio
Global Group (GG)	Utenti ed altri GG dello stesso dominio	Qualsiasi DLG e UG, GG dello stesso dominio	Risorse di qualsiasi dominio
Universal Group (UG)	Utenti, GG, UG di qualsiasi dominio	DLG e UG di qualsiasi dominio	Risorse di qualsiasi dominio

Utilizzo tipico e consigliato

- **Sebbene sia possibile assegnare diritti su risorse direttamente a UG e GG, la struttura consigliata è (caso più semplice):**
  - **individuare in ogni dominio utenti con esigenze analoghe e metterli in un GG**
  - **rendere i GG membro degli opportuni DLG**
  - **assegnare i permessi d'uso delle risorse ai DLG**



Utilizzo tipico e consigliato

- In realtà molto ampie è possibile sfruttare gli UG per aggiungere un livello di nesting che consenta all'*enterprise administrator* di raggruppare i GG
  - individuare in ogni dominio utenti con esigenze analoghe e metterli in un GG
    - in questo modo si delega al *domain administrator* che conosce bene la propria realtà il compito di popolare i GG
  - raggruppare i GG omologhi in un UG
    - in questo modo si evita che alla riorganizzazione dei domini, o in generale alla comparsa/scomparsa di GG, i singoli amministratori delle risorse debbano agire sui DLG, ripopolandoli di conseguenza. Sarà l'*enterprise administrator* a sapere quali GG è opportuno assegnare agli UG, mentre questi ultimi saranno creati o distrutti solo in casi eccezionali.
  - rendere l'UG membro degli opportuni DLG
  - assegnare i permessi d'uso delle risorse ai DLG
    - gli amminstratori delle singole risorse possono scegliere i soggetti (GG e UG) preconfigurati ai passi precedenti come membri di DLG, anzichè come soggetti cui attribuire direttamente permessi, in modo che l'aggiunta o la rimozione di un UG/GG da un DLG si applichi automaticamente a tutte le risorse su cui tale DLG può operare

Gruppi predefiniti – domain local

Gruppo	Caratteristiche
Administrators	Controllo completo della macchina locale con tutti i provilegi; membri di default comprendono i Domain Admins, gli Enterprise Admins, e l'account Administrator.
Account Operators	Amministrazione degli utenti del dominio.
Backup Operators	Back up e restore dei file sulla macchina locale indipendentemente dai permessi ad essi associati; log on e shut down. Le Group policies possono limitare questi privilegi di default.
Guests	Logon/shutdown limitato sulla macchina locale.
Print Operators	Amministrazione delle stampanti locali.
Replicator	Gestione delle funzioni e dei servizi di replica di Active Directory.
Server Operators	Amministrazione del sistema locale.
Users	Esecuzione di applicazioni, accesso alle stampanti, logon/shutdown/locking, creazione e modifica fi gruppi locali; tutti gli utenti del dominio sono membri di default.

Gruppi predefiniti – global

Gruppo	Caratteristiche
Domain Admins	Privilegi di amministrazione su tutti i sistemi appartenenti al dominio
Domain Computers	Tutti i computer del dominio
Domain Controllers	Tutti i domain controller
Domain Guests	Appartiene al DLG “Guest”
Domain Users	Appartiene al DLG “Users”
Enterprise Admins	Appartiene al gruppo “Domain Admins” di ciascun dominio, concedendo quindi i privilegi di amministrazione a livello di foresta.
Group Policy Creators Owners	Ai membri è consentito modificare le group policy
Schema Admins	Ai membri è consentito modificare lo schema di Active Directory

Group Policy

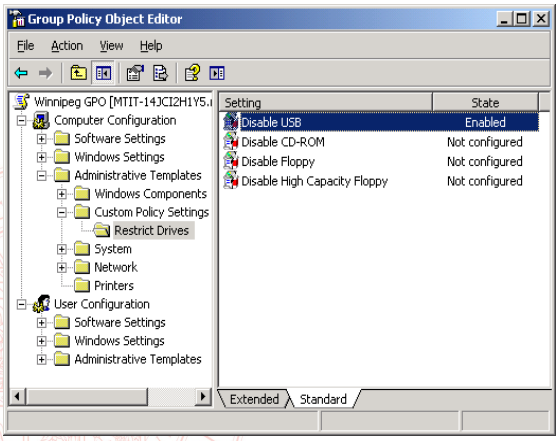
- Group Policy fornisce un quadro di riferimento per controllare l'ambiente di utenti e computer, cioè per assegnare quel tipo di privilegi o restrizioni che non sono legati a risorse fisiche ovvie quali file, cartelle, ecc.
- Le regole vengono definite in un Group Policy Object, che può essere collegato a qualsiasi contenitore di oggetti (una OU, un Site, un Domain) per applicarle a tutti gli oggetti in esso contenuti.
- Ogni GPO contiene due sezioni distinte
  - impostazioni per gli utenti (user settings)
  - impostazioni per i computer (computer settings)
- In ciascuna delle due sezioni le impostazioni sono ulteriormente classificate in:
  - impostazioni software (software settings)
  - impostazioni di Windows (Windows settings)
  - modelli per l'amministrazione (administrative templates)

Group Policy - impostazioni

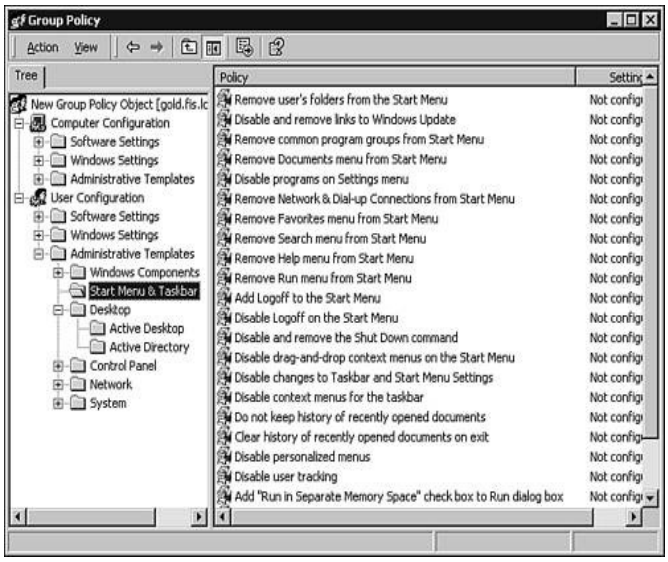
Categoria	Finalità	Disponibile per computer?	Disponibile per utenti?
Software settings	Installare, aggiornare, rimuovere applicazioni	Sì	Sì
Windows settings	Definire scripts ed impostazioni di sicurezza (vedi colonne a fianco)	Start-up e shutdown scripts, numerose impostazioni di sicurezza	Logon e logoff scripts, alcune impostazioni di sicurezza, impostazioni di Internet Explorer, folder redirection
Administrative templates	Definire in modo centralizzato le impostazioni del registro di sistema	Sì	Sì

Group Policy - esempi

Limitare l'uso di una intera categoria di dispositivi, come le porte USB

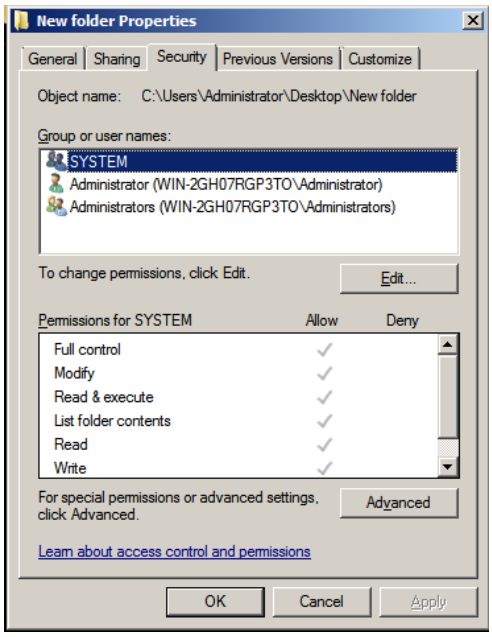


Configurare il contenuto del desktop o del menu avvio

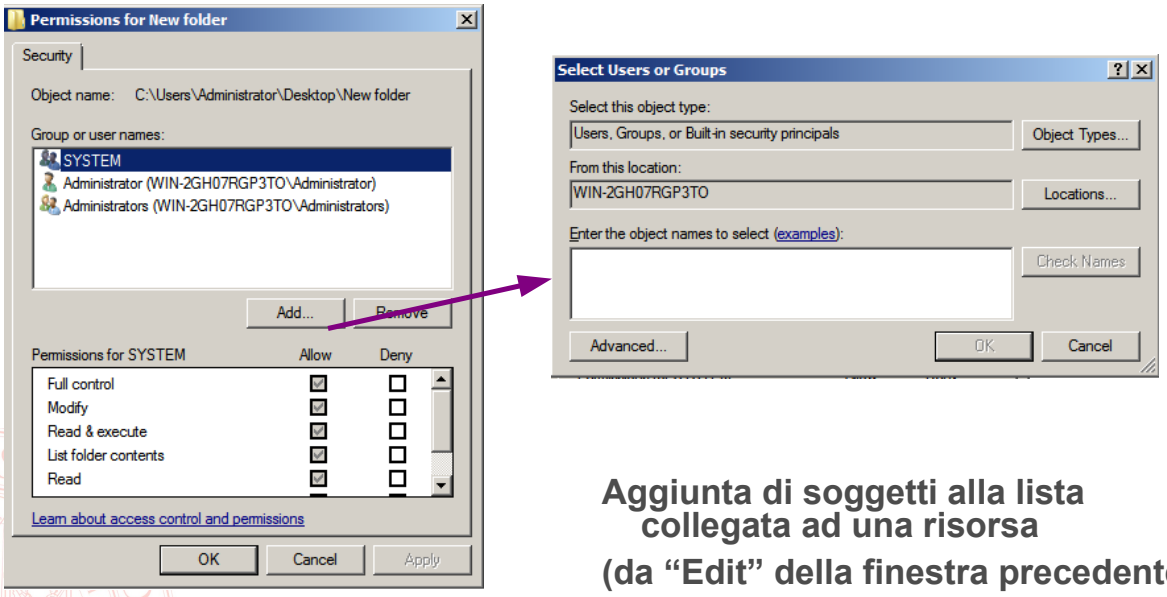


Controllo dell'accesso

- Le autorizzazioni sono assegnate sotto forma di ACL: ad ogni risorsa è associata una lista di soggetti (utenti o gruppi) e dei relativi permessi che essi detengono sulla risorsa
- Le ACL sono disponibili
  - su partizioni NTFS (non FAT)
  - sulle condivisioni di risorse in rete
- Per modificare le ACL è necessario
  - o detenere l'Ownership
  - o che nell'ACL medesima siano assegnati i permessi 'Full Control' o 'Change Permissions'



Esempio di modifica delle ACL



Aggiunta di soggetti alla lista collegata ad una risorsa

(da "Edit" della finestra precedente)

Ownership ed autorizzazioni

- Ownership
  - L'Owner di files e directories ha il pieno controllo (Full Control)
  - Administrator può sempre prendere l'ownership
  - L'Owner può assegnare le permissions per prendere l'Ownership
  - Nota: gli utenti che creano un file o una directory ne detengono l' Ownership
- Autorizzazioni NTFS predefinite
  - Ad Everyone viene assegnato automaticamente Full Control
  - I nuovi file ereditano le autorizzazioni della cartella in cui vengono creati (questo vale anche per i files che vengono copiati in un direttorio)

Accesso ed auditing

- Le ACL per il controllo dell'accesso fanno sì che, ad ogni tentativo di utilizzo di una risorsa, il sistema risponda autorizzando o negando l'operazione
- Ad ogni risorsa è inoltre associata una SACL utilizzata per l'auditing, che si presenta come una normale ACL, ma permette di tracciare gli esiti dei tentativi di utilizzo
- Le regole nella SACL possono essere impostate in modo che
  - quando un determinato soggetto tenta un'operazione e, grazie alla configurazione della ACL standard, *riesce*, questo evento sia registrato
  - quando un determinato soggetto tenta un'operazione e, grazie alla configurazione della ACL standard, *viene bloccato*, questo evento sia registrato

Autorizzazioni standard e speciali

- L'obiettivo del sistema di controllo delle autorizzazioni è duplice
  - elevata precisione nel controllo dell'accesso
    - in termini di tipo di azioni da concedere/negare
    - in termini di gestione delle complesse relazioni tra utenti e gruppi che possono essere titolari delle autorizzazioni
  - facilità d'uso
    - il sistema è Discretionary Access Control (DAC), quindi consente ad ogni utente anche non tecnico di manipolare le autorizzazioni sulle proprie risorse
    - anche l'utente più esperto per il 90% del tempo fa cose semplici

Autorizzazioni standard e speciali (cont.)

- La soluzione ai due problemi è realizzata con un sistema che prevede tre strati di interfaccia utente per “svelare” all'occorrenza i dettagli che servono:
  - a basso livello il sistema supporta
    - molte autorizzazioni (*autorizzazioni speciali*) --> possibilità di controllo fine sui permessi
    - con una logica a tre valori (*allow, deny, not set*) --> possibilità di definire regole di interazione quando diverse ACL vengono combinate
  - le autorizzazioni speciali sono aggregate in un set più ridotto di *autorizzazioni standard*
  - le autorizzazioni standard possono essere visualizzate a due valori (*allow, not set*) o mostrando esplicitamente i tre valori

Autorizzazioni standard (elenco)

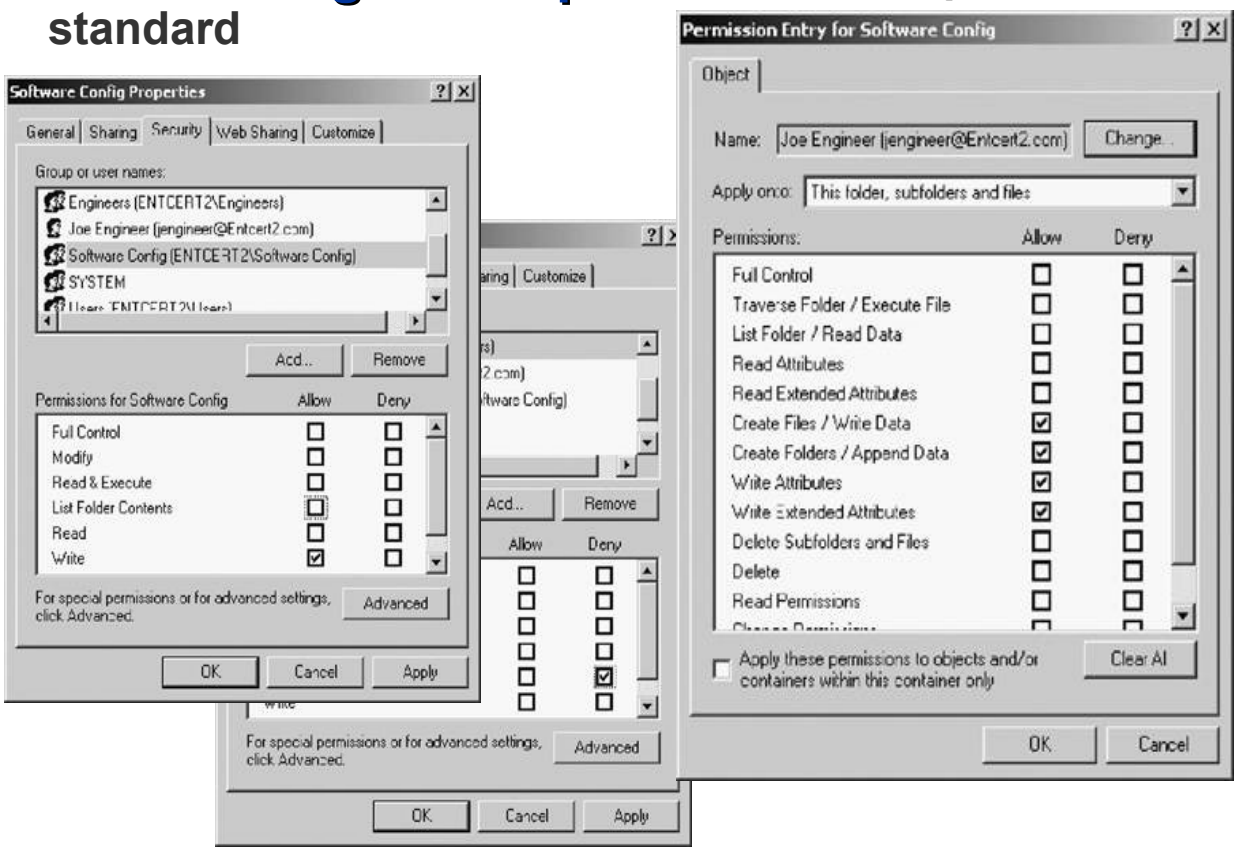
Abbreviation	Type	Description
R	Read	Provides the designated user or group the ability to read the file or the contents of the folder.
W	Write	Provides the designated user or group the ability to create or write files and folders.
RX	Read & Execute	Provides the designated user or group the ability to read file and folder attributes, view folder contents, and read files within the folder. If this permission is applied to a folder, files with inheritance set will inherit it (see the inheritance discussion).
L	List Folder Contents	Same as Read & Execute, but not inherited by files within a folder. However, newly created subfolders will inherit this permission.
M	Modify	Provides the ability to delete, write, read, and execute.
F	Full Control	Provides the ability to perform any action, including taking ownership and changing permissions. When applied to a folder, the user or group may delete subfolders and files within a folder.

Corrispondenza tra autorizzazioni standard e speciali

Types	Full Control	Modify	Read & Execute	List Folder Contents	Read	Write
Traverse Folder/Execute File	X	X	X	X		
List Folder/Read Data	X	X	X	X	X	
Read Attributes	X	X	X	X	X	
Read Extended Attributes	X	X	X	X	X	
Create Files/Write Data	X	X				X
Create Folders/Append Data	X	X				X
Write Attributes	X	X				X
Write Extended Attributes	X	X				X
Delete Subfolders and Files	X					
Delete	X	X				
Read Permissions	X	X	X	X	X	
Change Permissions	X					
Take Ownership	X					

ACL editing - esempi

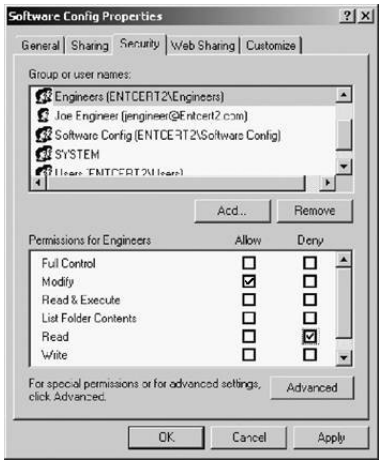
standard special

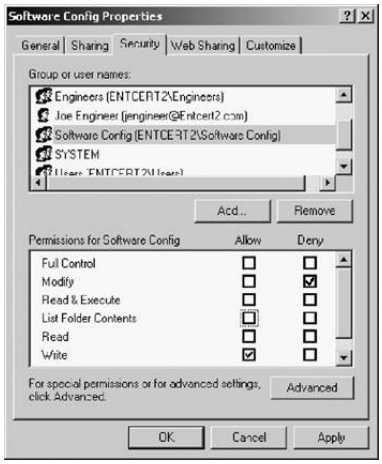


Composizione

- Come si può vedere, ogni permesso può essere impostato in tre modi
  - esplicitamente **allow**
  - esplicitamente **deny**
  - o **non impostato**
- Windows segue un modello default deny: ciò che non è esplicitamente consetito è proibito
  - quindi **non impostato** == **deny** ?
  - a cosa servono due modi diversi di proibire l'accesso?
- Un utente può appartenere a molti gruppi!
  - ogni gruppo può avere permessi distinti nell'ACL di una risorsa
  - il permesso complessivo dell'utente sarà la somma, bit a bit, di tutti i permessi ottenuti in quanto membro dei propri gruppi
- **non impostato** (sia allow che deny sono bit a zero) presente nei permessi di un gruppo consente che un **allow** ottenuto da un altro gruppo possa avere effetto: **non impostato == deny "debole"/scavalcabile**
- unt **deny** esplicito prevarrà sempre su di un eventuale **allow** ottenuto da un altro gruppo: **deny esplicito == deny "forte"/non scavalcabile**

Composizione





i membri del gruppo Software Config non potrebbero ottenere **modify** anche se appartenessero a un gruppo (come Engineers) che lo ha (strong deny)

i membri di Engineers non hanno il permesso **write** ma lo potrebbero ottenere se appartenessero a un gruppo (come Software config) che lo ha

permessi dei membri del gruppo Engineers

permessi dei membri del gruppo Software Config

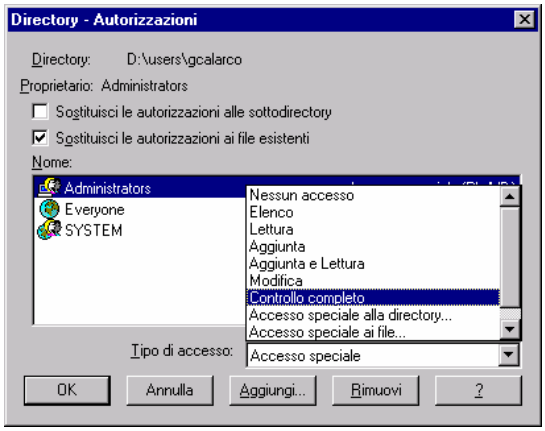
← non imp. resta non imp. == deny

← deny prevale su allow == deny

← deny esplicito == deny

← allow esplicito == allow

Esempio di manipolazione delle ACL (vecchia interfaccia semplificata)



Nelle nuove versioni di Windows, tutte le interfacce mostrano le colonne Allow e Deny, nelle vecchie esisteva una vista ancor più semplificata per scegliere

- **una delle autorizzazioni standard (implicitamente equivalente a settare "Allow" su tutte le autorizzazioni speciali corrispondenti)**
- **oppure "Nessun accesso", equivalente a deny su tutti i permessi**

Autorizzazioni di accesso alle cartelle

- Sulle cartelle è possibile impostare una delle seguenti autorizzazioni standard:
  - Nessun accesso
  - Elenco
  - Lettura
  - Aggiunta
  - Aggiunta e Lettura
  - Modifica
  - Controllo completo
- Nota: I gruppi o gli utenti a cui è stata concessa l'autorizzazione 'Controllo completo' su una cartella sono in grado di eliminarne i file, indipendentemente dall'autorizzazione che li protegge.

Autorizzazioni di accesso ai file

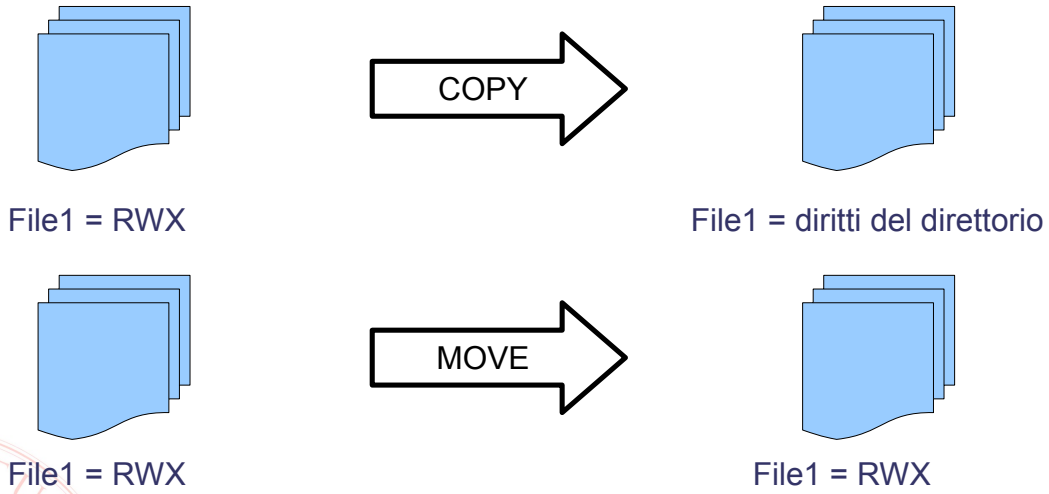
- Sui file è possibile impostare le seguenti autorizzazioni standard:
  - Nessun accesso
  - Lettura
  - Modifica
  - Controllo completo
- Impostando le autorizzazioni di accesso a un file sarà possibile specificare il tipo di accesso al file consentito a un gruppo o a un utente. Altrimenti, un file eredita le autorizzazioni proprie della cartella in cui è stato creato.
- Nota I gruppi o gli utenti cui si concede l'autorizzazione Controllo completo su una cartella possono eliminarne i file, indipendentemente dall'autorizzazione che li protegge.

Esempio di composizione di permessi

Permessi di Michael	Permessi di Research	Permessi di Development	Permessi di Michael (effettivi)
Read	Read	-	Read
Write	-	Read	Change
Take Ownership	Read	Change	Take Ownership & Change
No Access	Read	Change	No Access
Change	No Access	Change	No Access

RWX = Read, Write, Execute
DPO = Delete, Permissions, Ownership
Change = RWXD

Permessi dopo un copy/move di file

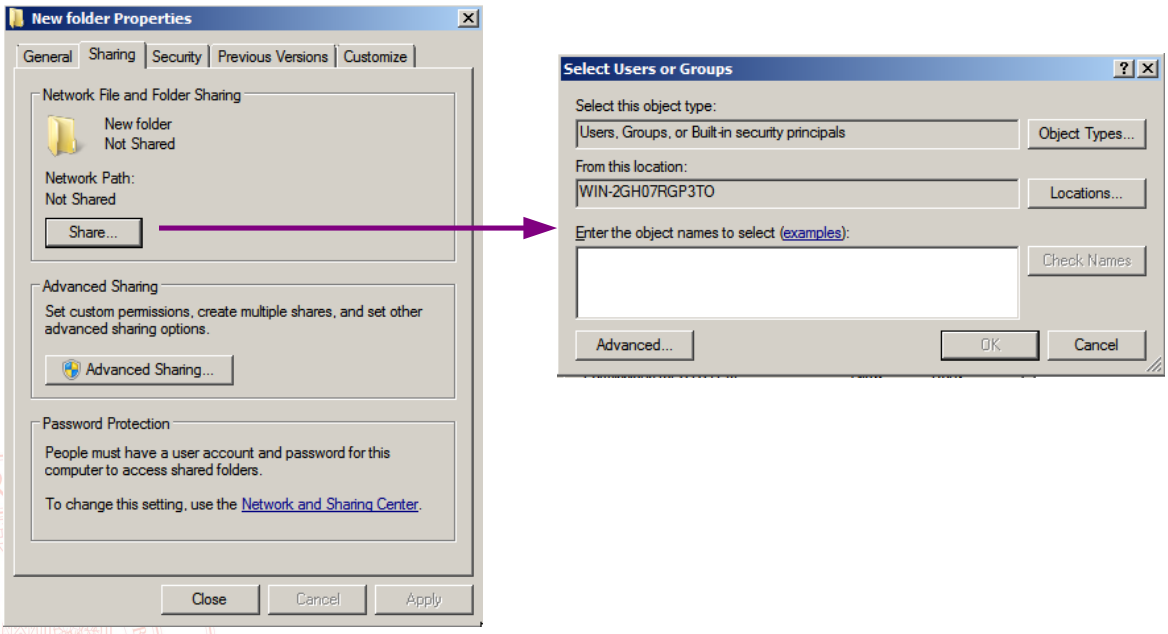


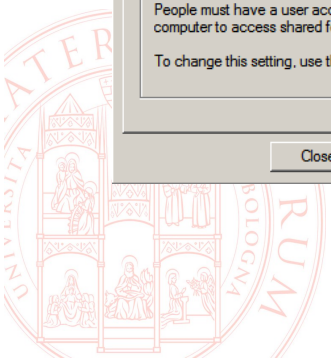
NOTA: MOVE verso una partizione diversa dalla sorgente = COPY (+delete)!

Condivisione di risorse

- Share = directory (folder) condivisa
- I diritti necessari per attivare le condivisioni sono concessi di default ai gruppi
  - Administrators
  - Server Operators (se in un dominio)
  - Power Users (se in un workgroup)
- Gli Users devono avere almeno il permesso List per fruire della directory condivisa

Condividere una cartella





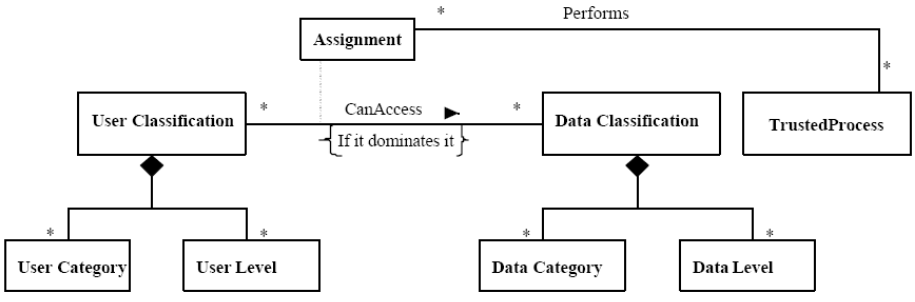
Permessi locali vs. Permessi sulla condivisione

	Permessi assegnati	Permessi di Michael
Permessi Share	Everyone: Read Michael: Change	Change (RWXD)
Permessi locali	Everyone: Read Michael: Read	Read (RX)
Permessi effettivi		Read (RX)

Le ACL di Share si comportano come quelle di NTFS in termini di composizione di permessi, però vengono applicate in serie una all'altra, per cui complessivamente l'autorizzazione effettiva è quella più restrittiva tra le due

Un breve cenno a MAC

- MANDATORY: le regole di controllo degli accessi sono dettate da un'autorità centrale e i soggetti non possono modificarne alcun dettaglio
- Ad ogni soggetto o risorsa viene assegnata una classe di accesso che specifica tipicamente
  - un livello di sicurezza all'interno di un insieme ordinato di valori, ad es {TopSecret ► Segreto ► Riservato ► Non classificato}
  - una categoria (compartment) all'interno di un insieme non ordinato, ad es. {armi, piani di battaglia, unità di combattimento, …} che riflette le aree funzionali del sistema



Come si usano le classi

- le risorse sono etichettate (classification) con un livello di sicurezza (sensitivity) che rappresenta la gravità delle conseguenze di una violazione delle policy che le riguarda
- i soggetti sono etichettati con un livello di sicurezza che rappresenta la loro affidabilità: clearance
- le categorie sono utilizzate per raffinare le politiche
- vengono stabilite relazioni di dominanza tra classi; detti
  - S un livello di sicurezza
  - C un insieme di categorie
  - $L_1 = \langle S_1, C_1 \rangle$
  - $L_2 = \langle S_2, C_2 \rangle$

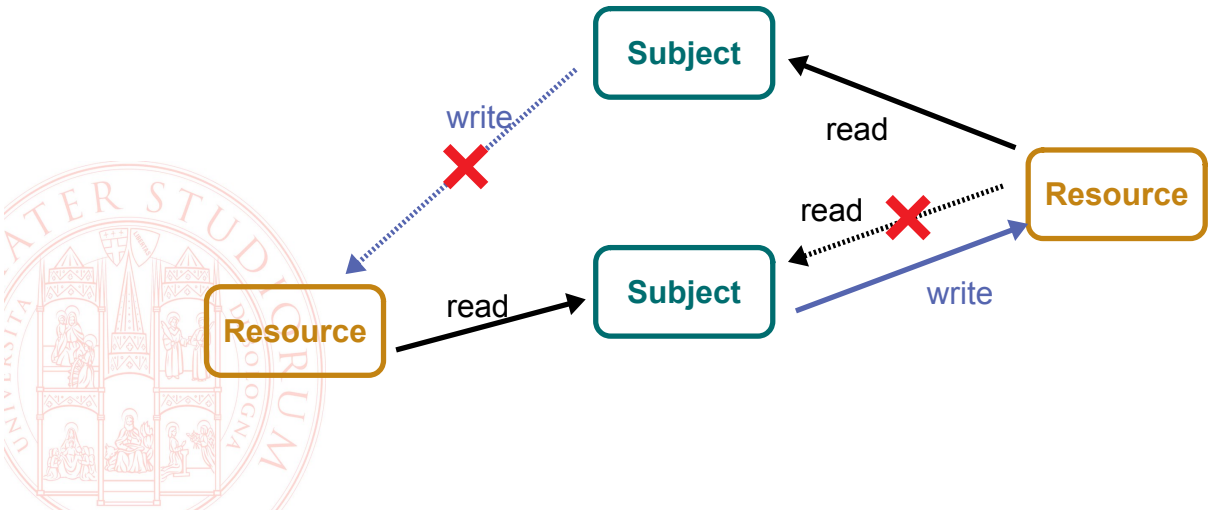
L_1 domina L_2 se e solo se $S_1 \geq S_2$ e $C_1 \supseteq C_2$

Come si usano le classi

- Le relazioni di dominanza vengono usate in modo diverso a seconda della proprietà di sicurezza da proteggere
  - riservatezza → Bell-LaPadula model
  - integrità → Biba model
- L'applicazione simultanea dei due modelli è possibile assegnando due classi di accesso a ogni soggetto e risorsa, una usata per controllare la riservatezza e l'altra per controllare l'integrità

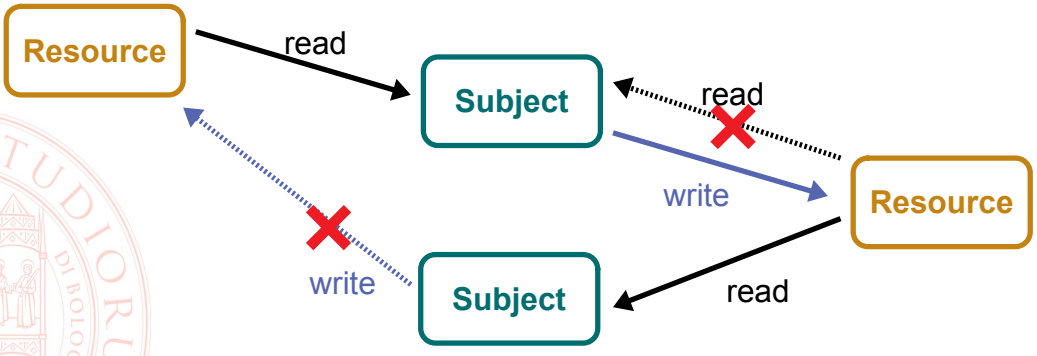
BLP (Bell-LaPadula)

- Due regole per proteggere la riservatezza
  - NO-READ-UP: un soggetto può leggere una risorsa solo se la sua classe di accesso domina la classe di accesso della risorsa (altrimenti leggerebbe una risorsa troppo sensibile per il suo livello)
  - NO-WRITE-DOWN: un soggetto può modificare una risorsa solo se la sua classe di accesso è dominata dalla classe di accesso della risorsa (altrimenti potrebbe far trapelare un segreto in luoghi accessibili a soggetti con minor clearance)



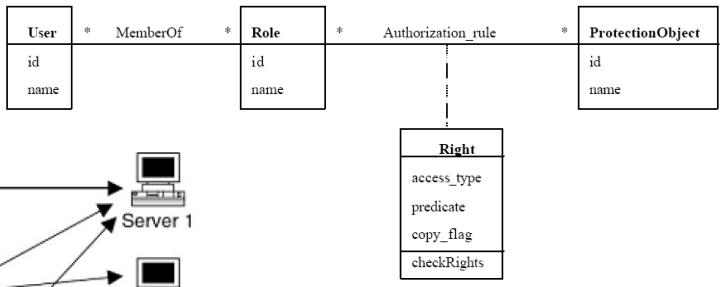
Biba

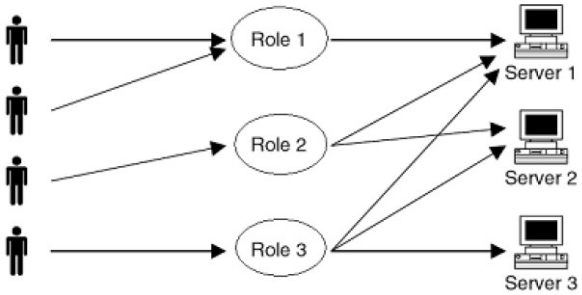
- Due regole per proteggere l'integrità:
  - NO-READ-DOWN: un soggetto può leggere una risorsa solo se la sua classe di accesso è dominata dalla classe di accesso della risorsa (altrimenti utilizzerebbe informazioni meno attendibili al proprio livello di fiducia più elevato)
  - NO-WRITE-UP: un soggetto può scrivere una risorsa solo se la sua classe di accesso domina la classe di accesso della risorsa (altrimenti modificherebbe una risorsa troppo sensibile per il suo livello)



Un brevissimo cenno a RBAC

- le autorizzazioni non sono concesse a utenti, ma a *ruoli*
- il ruolo ricoperto da un utente può cambiare dinamicamente
  - nel tempo
  - secondo il contesto





RBAC

- RBAC è un modello "policy neutral" che permette di esprimere tutti i principi fondamentali per la sicurezza:
  - minimo privilegio
  - separazione delle responsabilità
  - astrazione (es. usando generici "debiti" e "crediti" al posto di permessi specifici delle risorse come “possibilità di lettura/scrittura”)
- Vantaggio: le autorizzazioni cambiano poco o per nulla, se correttamente modellate
  - il ruolo dell’amministratore della sicurezza diventa essenzialmente quello di assegnare il ruolo appropriato ai soggetti
- Esiste un modello standard (ANSI/INCITS 359-2004) con livelli di funzionalità definiti in modo incrementale per adattare semplicemente la sua implementazione in contesti di sicurezza differenziati