system security
unix/linux – access control

controllo di accesso in unix

- i processi accedono alle risorse (file e altro) tramite il kernel
  - mediante system call
- il controllo di accesso viene eseguito dal kernel
- in unix il “soggetto” del controllo di accesso è un processo ed è identificato dalle sue **credenziali**

credenziali di un processo unix

a ciascun processo unix è associato…

- UID: id utente “reale”, usato per tracciare chi ha lanciato il processo (es. dal comando ps)
- **EUID: “effective” UID, usato per access control**

- GID: gruppo principale “reale”, per tracciamento
- **EGID: gruppo principale “effettivo”, usato nell’access control**
  - ...e come gruppo quando si crea un file
- **supplementary groups: usati nell’access control**

real vs. effective

per quasi tutti i processi UID=EUID e GID=EGID

ma non sempre...

- vedi suid/sgid più avanti

processi privilegiati e non

- processi privilegiati
  - **EUID=0**, cioè processi che girano “come root”
  - il kernel non limita tali processi
- processi non privilegiati
  - **EUID≠0**
  - le operazioni ammesse dal kernel sono alcune

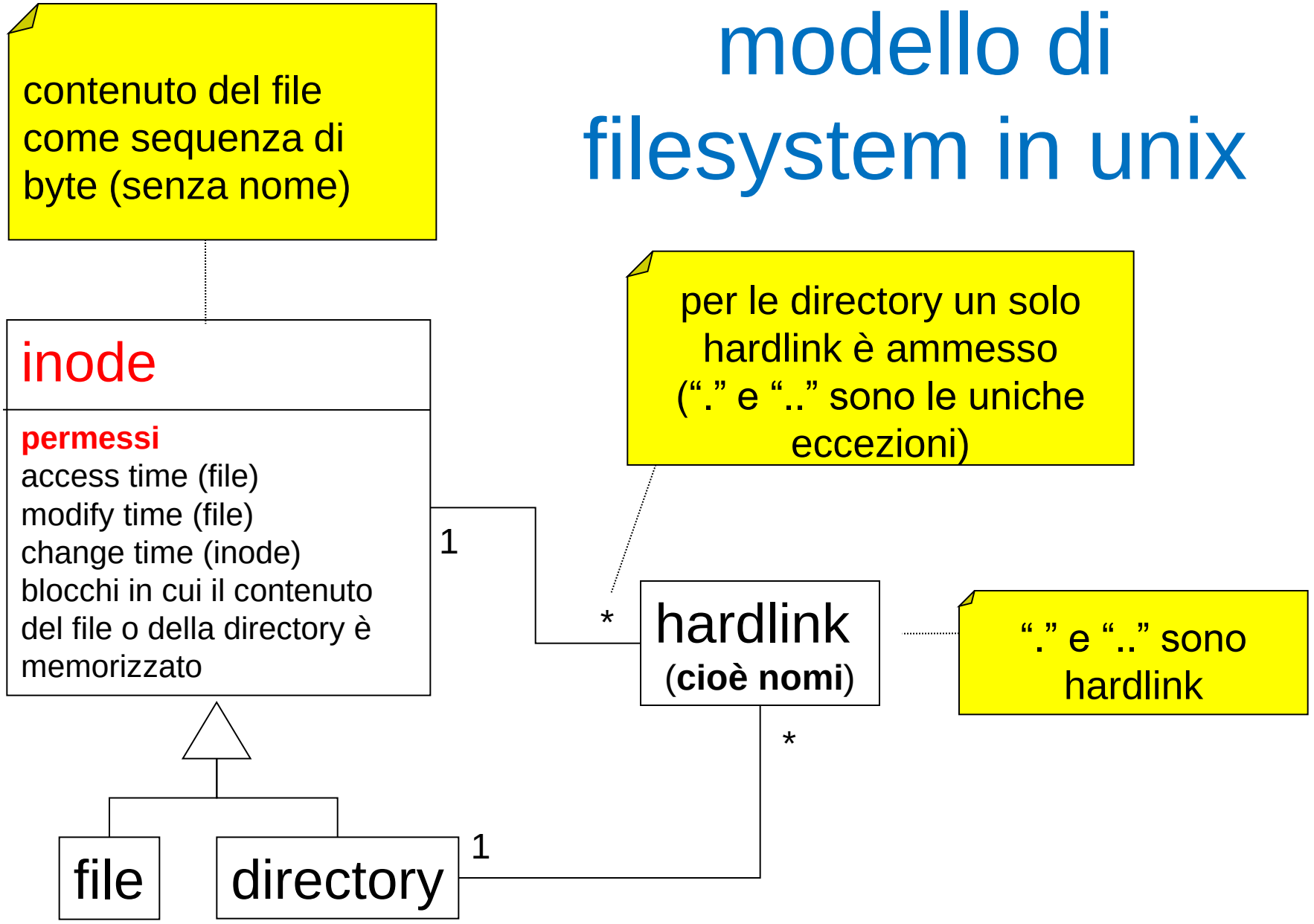
diritti dei processi **non** privilegiati

- filesystem
  - **creare e cancellare file (links) in directory in accordo con i permessi configurati per la directory**
  - **aprire** file in accordo con i permessi configurati per il file
    - per qualsiasi operazione su un file gia' aperto il controllo di accesso è solo contro la “modalità di apertura” e non contro i permessi del file
  - **cambiare permessi** dei “propri” file/directory
  - **cambiare proprietario** dei “propri” file/directory
- rete
  - usare socket regolari (tcp, udp, unix, **no raw socket** i.e. no ping)
  - binding di port porte >=1024
- processi
  - kill o ptrace su processi dello stesso utente
    - “kill” comprende tutti i segnali come stop, cont, ecc.
  - possibili altri vincoli, es. ptrace solo sui figli (YAMA LSM)

diritti dei processi privilegiati

- un processo *privilegiato* può praticamente tutto, ecco alcuni esempi…
- **filesystem**
  - nessuna limitazione
  - **cambiare permessi** e **proprietario** di tutti i file
- rete
  - usare qualisiasi tipo di socket, anche raw
  - binding di well-known ports <1024
  - amministrazione: interfacce, tabella di routing
  - rete in modalità promiscua e generazione di pacchetti qualsasi
- processi
  - inviare qualsiasi segnale e ptrace su qualsiasi processo
  - renice
  - cambio delle credenziali (necessario per login degli utenti)
- varie
  - (u)mount, quota, swap
  - reboot, shutdown, chroot, kernel modules, system clock

modello di filesystem in unix

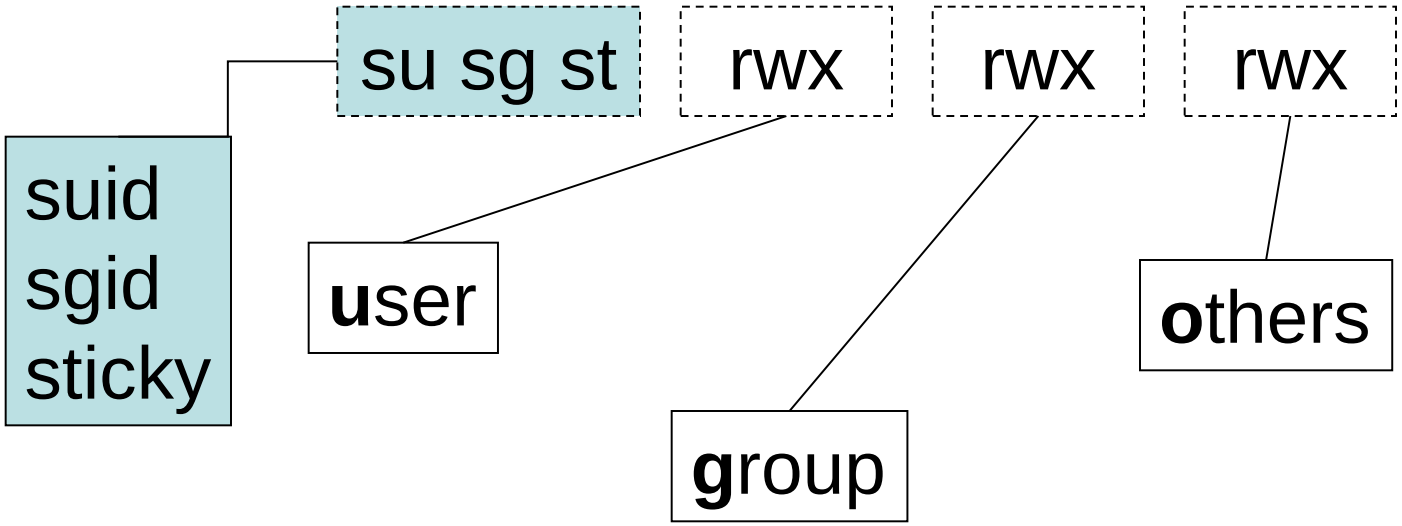


system calls per hardlink: link e unlink

- link(srg,dest): creazione di un hardlink dest che punta allo stesso inode di srg
  - da shell si usa il comando “ln srg dest”
- unlink(x): rimozione di un hardlink x
  - un inode viene rimosso quando non ha più alcun hardlink che lo punta
  - da shell si usa il comando “rm x”

filesystem unix: permessi

- permessi su inode per file
  - read (r), write (w), execute (x)
- permessi su inode per directory
  - read (r), write (w), **search** (x)
- a ciascun inode sono associati tre gruppi (triplette) di permessi chiamati ***u**ser, **g**roup, **o**thers*
  - più altri



filesystem unix: chmod

- chmod <*chi*> +/-/= <*cosa*> pippo.txt
  - chi: u=user, g=gruppo, o=other
  - chmod ug+rw pippo.txt (set read e write per user e group
  - chmod o-x pippo.txt (unset execution per gli altri)
  - chmod ug=r pippo.txt (set r per user e group e unset il resto)
- sinstassi “ottale” ancora usata
  - “chmod 660 pippo.txt” configura rw- rw- ---

algoritmo per access control

- input: le syscall per aprire file (open) e directory (opendir) specificano l'oggetto da aprire come **pathname**
  - ciascun "name" del pathname ha associato dei permessi (in realtà li ha il relativo inode) che sono input all'algoritmo access control
  - di tali permessi l'algoritmo usa solo una tripletta (vedi slide successiva).

- **tutte le operazioni richiedono premesso search** (x) su tutte le directory nominate in tutti i pathname passati come parametro
  - compresa la directory corrente per pathname relativi
  - …**e in più richiedono permessi specifici**...

algoritmo per access control: selezione della tripletta

input:

- credenziali processo: EUID, EGID, supplementary groups
- UID e GID della risorsa, permessi (rxw) risorsa

1. **UID della risorsa = EUID del processo** allora si usa solo (!) la tripletta **“user”**
   altrimenti...

2. **GID della risorsa = EGID o in supplementary groups del processo**
   allora si usa solo (!) la tripletta **“group”**
   altrimenti...

3. si usa solo (!) la tripletta **“others”**

filesystem unix: permessi specifici richiesti per processi **non** privilegiati

- permessi specifici richiesti da operazioni su files
  - open create: w sulla directory (permessi del nuovo file stabiliti da parametro e umask)
  - open read: r sul file
  - open write append truncate: w sul file
  - execute: x sul file
  - link: w sulla directory destinazione
  - unlink: w sulla directory
- permessi specifici richiesti da operazioni su files e directory
  - chmod: uid del processo uguale a uid del inode
  - chown: uid del processo uguale a uid del inode
  - stat: -
- permessi specifici richiesti da operazioni su directory
  - mkdir: w sulla directory contenitore
  - rmdir: w sulla directory contenitore
  - readdir: r sulla directory da leggere

filesystem unix: permessi specifici richiesti processi privilegiati

nessuno

minimalità dei diritti e alternative

- questo sistema di diritti ha una limitata espressività
  - il principio della minimalità dei diritti è difficile da raggiungere con l'access control standard
- si possono altri strumenti come SELinux e AppAromor
- oppure si delega l'accesso ad un altro processo che esegue il controllo dei diritti, che può essere…
  - un server privilegiato + protocollo (usato molto in windows)
  - un processo lanciato sudo (adeguatamente configurato)
  - un processo il cui eseguibile ha suid o sgid bit configurato

set-user-id bit (suid)

- molti comandi hanno bisogno dei diritti di “root” per funzionare correttamente ma gli utenti senza specifici diritti devono poterli comunque eseguire
  - alcuni esempi: ping, sudo, passwd, e molti altri!
- tali comandi hanno il bit “set user id” settato

```
pizzonia@pisolo$ cd /bin
pizzonia@pisolo$ ls -l ping
-rwsr-xr-x  1 root root 30764 Dec 22  2003 ping
```

- **un eseguibile con tale bit settato viene eseguito con EUID pari al proprietario del file indipendentemente da chi ne ha richiesto l’esecuzione**

set-group-id bit (sgid)

- il bit set-group-id si comporta in maniera analoga per l'EGID

```
-rwxr-sr-x  1 root tty      /usr/bin/wall
-rwxr-sr-x  1 root crontab  /usr/bin/crontab
```

- **usando set-uid e set-gid non necessariamente si “diventa root”, può bastare un utente o un gruppo che è proprietario di certi file necessari all'applicazione in questione**
- **questo aumenta la sicurezza del sistema**

other important topics

- logging (syslog)
- user/password database
- login and Pluggable Authentication Modules (PAM)
- etc…