

Sviluppo di software di rete con l'utilizzo di sistemi UNIX

Gestione dei processi

Claudio Vicari

Cambiare UID e GID

```
int setuid(uid_t uid);  
int setgid(gid_t gid);
```

- ➔ queste funzioni cambiano real user id e real group id di un processo
- ➔ come al solito, il valore di ritorno è 0 in caso di successo, -1 altrimenti (errno contiene i dettagli)
- ➔ oltre a UID, GID, EUID, EGID, negli ultimi sistemi Unix esistono anche i *saved uid*, *saved gid* per consentire ad un processo di cambiare momentaneamente permessi per poi tornare alla situazione originaria

Cambiare UID e GID: permessi

- ➔ non sempre si può cambiare UID, GID
- ➔ casi possibili (per `setuid`, `setgid` è analogo):
 - 1 se il processo ha diritti di root, `setuid` cambia il real uid, l'*effective uid*, e il *saved uid* al valore di *uid*
 - 2 se il processo non ha privilegi di root, ma *uid=real uid* o *uid=saved uid*, allora `setuid` cambia solo l'*effective uid*
 - 3 negli altri casi, non è permesso cambiare uid

Cambiare UID e GID: esempio

- ➡ un programma di u1 con setuid attivo viene eseguito (exec), dall'utente u2
- ➡ il processo ottiene *uid=u2*, *euid=u1*, *saved set-uid=u1*
- ➡ svolge alcuni lavori riservati a u1...
- ➡ chiama *setuid(u2)*, e diviene *euid=u2*
- ➡ esegue dei lavori con *euid=u2*, per esempio leggere file di u2
- ➡ esegue *setuid(u1)*, quindi *euid=u1*
- ➡ torna a lavorare con i privilegi di u1,

Cambiare UID e GID: conseguenze

➔ alcuni fatti:

- solo root può cambiare il real user id. Questo è quello che avviene quando facciamo login (lo fa proprio il programma login)
- lo effective uid può essere impostato da exec, solo nel caso in cui il programma sia set-user-id
- exec copia lo effective uid in saved uid
- un processo di root che faccia setuid, non può tornare indietro!
- grazie al saved uid, un processo set-user-id non root, può cambiare privilegi temporaneamente per poi tornare ai privilegi precedenti

UID e GID: riepilogo

exec

ID	set-user-id off	set-user-id on
real UID	unchanged	unchanged
effective UID	unchanged	set from program file uid
saved set-user-id	copied from euid	copied from euid

setuid(uid)

ID	superuser	non superuser
real UID	set to uid	unchanged
effective UID	set to uid	set to uid
saved set-user-id	set to uid	unchanged

setreuid, setregid

```
int setreuid(uid_t ruid, uid_t euid);  
int setregid(gid_t rgid, gid_t egid);
```

- ➔ funzioni per modificare sia real uid che effective uid
- ➔ processi non root possono fare solo *euid=euid*, *euid=ruid*, o *euid=saved-set-uid*
- ➔ root può fare tutto
- ➔ anche per questa funzione, il comportamento sui gruppi è analogo

File interpretati

- ➔ i file interpretati sono file che cominciano con:
 `#!pathname [arguments]`
- ➔ esempi:
 `#!/bin/sh`
 `#!/usr/bin/perl -w`
- ➔ pathname è un percorso assoluto
- ➔ chi riconosce questi file è il kernel stesso, durante la chiamata ad exec
- ➔ il processo che viene effettivamente eseguito (exec) è l'interprete fornito in pathname

File interpretati

➡ esempio:
`#!/usr/bin/perl`

`print "interpretato con perl";`

- ➡ rendiamo eseguibile il file
- ➡ se togliamo la prima riga, il file non può essere eseguito
- ➡ se eseguiamo
 - `perl interpretato.pl`
- ➡ il risultato è lo stesso

La funzione system

```
int system(const char *command);
```

- ➔ esegue il comando specificato, in una shell:
 - sh -c command
- ➔ blocca il processo finché command non termina
- ➔ system è implementata usando fork, exec, wait
 - restituisce il valore di uscita del processo (esattamente come per wait) se tutto è andato a buon fine
 - -1 se c'è stato un errore
- ➔ se command==NULL, allora restituisce:
 - 0 se la shell è disponibile
 - non zero altrimenti