Modulo 4: Sezione C
System call relative agli attributi dei processi

Laboratorio di Sistemi Operativi I
Anno Accademico 2007-2008

Francesco Pedullà
(Tecnologie Informatiche)

Massimo Verola
(Informatica)

Copyright © 2005-2007 Francesco Pedullà, Massimo Verola
Copyright © 2001-2005 Renzo Davoli, Alberto Montresor (Universitá di Bologna)
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license can be found at: http://www.gnu.org/licenses/fdl.html#TOC1

Sommario delle system call relative ai processi

- **`getpid, getppid, getgid,`** ..... : forniscono gli attributi dei processi (PID, PPID, gruppo, ecc.).
- **`fork`**: crea un processo figlio duplicando il processo chiamante.
- **`exec`**: trasforma un processo sostituendo un nuovo programma nello spazio di memoria del chiamante.
- **`wait`**: permette la sincronizzazione fra processi.
- **`exit`**: termina un processo.

© 2005'2007 F. Pedullà, M. Verola

Identificatore di processo

- **Identificatore di processo**
  - ogni processo ha un identificatore univoco (intero non negativo) detto *Process Identifier* o brevemente *PID*
  - Generalmente il PID è un numero a 16 bit (di tipo **`pid_t`**) assegnato sequenzialmente dal kernel ogni volta che un nuovo processo è creato
  - ogni processo ha anche un *Parent Process Identifier* (*PPID*) associato, cioè il pid del processo da cui è stato generato (vedremo a breve il meccanismo di creazione di un nuovo processo)
- **Identificatori standard:**
  - PID 0: Non assegnato o assegnato a un processo del kernel
  - PID 1: Processo **`init (/sbin/init)`**
    - viene creato dal kernel al termine della procedura di bootstrap
    - effettua tutta una serie di azioni per portare il sistema ad un certo stato (ad es. multiuser)

© 2005'2007 F. Pedullà, M. Verola

Funzioni relative agli identificatori di processo

- `pid_t getpid();  // Process ID of calling process`
- `pid_t getppid(); // Process ID of parent process`

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("Il PID del processo è %d.\n",(int)getpid());
    printf("Il PID del processo padre è %d.\n",(int)getppid());
    return(0);
}
```


© 2005'2007 F. Pedullà, M. Verola

User-ID e group-ID

- Ad ogni processo sono anche associati degli identificatori di utente (*user-IDs*) e di gruppo (*group-IDs*) che ne determinano i privilegi, ovvero quali system call ha il diritto di invocare e su quali risorse
- Gli identificatori sono sei (o più, considerando i *supplementary group IDs*):
  - *real user ID, real group ID*: utente che ha lanciato il processo e gruppo associato all'utente
  - *effective user ID, effective group ID, supplementary group IDs*: utente e gruppo che il kernel considera per determinare i privilegi per l'accesso ai file; potrebbero non coincidere con *real user ID, real group ID* nel caso in cui il file eseguibile ha il *set-use-id* o il *set-group-id* bit attivo (leggi **`info coreutils 'File permissions' 'Mode Structure' )`**
  - *saved set-user-ID, saved set-group-ID*: salvati dalla system call **`exec`**, sono i valori di *effective user ID* e *effective group ID* subito dopo una **`exec`** di un file eseguibile i cui *set-user-id* o *set-group-id* sono attivi

© 2005'2007 F. Pedullà, M. Verola

User-ID e group-ID

- Generalmente i valori del *real user-id* e *real group-id* non cambiano per tutta la sessione di login (solo superuser ha il potere di cambiarli)
- Nella maggior parte dei casi l'*effective user ID* e l'*effective group ID* coincidono rispettivamente con il *real user-id* ed il *real group-id*.
- Ogni file (e quindi ogni file di programma) ha un *owner* e un *group owner:* se il file di un programma ha attivo il bit dei permessi noto come *set-user-id* (*set-group-id*), allora, quando viene invocato con una chiamata **exec**, l'*effective user-id* (*effective group-id*) diventerà quello dell'*owner* del file e non quello dell'utente che lo ha lanciato.
- Ad esempio, se un file di programma appartiene al superuser ed ha il bit *set-user-ID* attivo, l'utente che lancia il programma ottiene i privilegi del superuser durante l'esecuzione del programma stesso
- Un caso tipico è l'eseguibile **passwd** per cambiare la propria password:

```
$ ls -l /usr/bin/passwd
-r-s--x--x 1 root root 19336 Sep   7    2004  /usr/bin/passwd
```


© 2005'2007 F. Pedullà, M. Verola

Funzioni di set dello user-ID e group-ID

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

- Cambiano real user/group ID ed effective user/group ID
- Esistono delle regole per permettere al programma di cambiare questi ID:
  - se il processo ha privilegi da superutente, la funzione **`setuid`** cambia real uid / effective uid / saved-set-uid con uid
  - se il processo non ha privilegi da superutente e uid è uguale a real uid o a saved-set-uid, la funzione **`setuid`** cambia effective uid
  - se nessuna di queste condizioni è vera, viene ritornato un errore ed **`errno`** è settato uguale a EPERM
- Per quanto riguarda group ID, le regole sono del tutto simili

© 2005'2007 F. Pedullà, M. Verola

Funzioni relative agli attributi dei processi

```
uid_t getuid();  // get real user id
gid_t getgid();  // get real group id
uid_t geteuid(); // get effective user id
gid_t getegid(); // get effective group id
int getresuid(uid_t *ruid, uid_t *euid, uid_t *suid);
    // get real, effective and saved user id
int getresgid(gid_t *rgid, gid_t *egid, gid_t *sgid);
    // get real, effective and saved group id

int setuid(uid_t uid);   // set real user id
int setgid(gid_t gid);   // set real group id
int seteuid(uid_t euid); // set effective user id
int setegid(gid_t egid); // set effective group id
int setresuid(uid_t ruid, uid_t euid, uid_t suid);
    // set real, effective and saved user id
int setresgid(gid_t rgid, gid_t egid, gid_t sgid);
    // set real, effective and saved group id
```


© 2005'2007 F. Pedullà, M. Verola

Esempio

```
#include <unistd.h>
main()
{
     uid_t uid, euid, newuid;
     gid_t gid, egid, newgid;
     int status;

     uid = getuid();
     euid = geteuid();
     gid = getgid();
     egid = getegid();
     printf("real uid: %d, effective uid: %d\n",(int)uid,(int)euid);
     printf("real gid: %d, effective gid: %d\n",(int)gid,(int)egid);
     if ((status = setuid(newuid))==0) /* cambio effective uid */
          printf("nuovo effective uid: %d\n", (int)newuid);
     if ((status = setgid(newgid))==0) /* cambio effective gid */
          printf("nuovo effective gid: %d\n", (int)newgid);
}
```


© 2005'2007 F. Pedullà, M. Verola