

# File di Sistema ed Informazioni

# Introduzione

---

Il sistema operativo ha bisogno di conservare numerose informazioni relative ai servizi che mette a disposizione.

Ad esempio, ogni volta che si richiede di visualizzare il contenuto di una cartella, il sistema ritorna il nome ed il gruppo del proprietario di ciascun file o cartella in essa contenuto.

Ancora, ogni volta che ci autenticiamo su di un sistema, esso controlla che la password sia corretta.

## Dove si trovano queste informazioni?

# Introduzione

---

Tutte le **informazioni** relative al sistema operativo si trovano nei **file di sistema**.

Essi intervengono durante tutte le **normali operazioni** su di un sistema Unix.

Ad esempio:

- ogni volta che si esegue **ls -l** viene utilizzato il file **/etc/passwd** per **convertire** lo **user ID** nel **nome dell'utente**;
- all'atto del **login** viene utilizzato il **file delle password** per verificarne la correttezza;
- ...

# File `/etc/passwd`

---

Il file `/etc/passwd` contiene il **database** degli **utenti**. Ciascun record ha i seguenti campi, contenuti nella struttura **passwd** definita in `<pwd.h>`:

Campo	Descrizione
<code>char *pw_name</code>	user name
<code>char *pw_passwd</code>	user password (*)
<code>uid_t pw_uid</code>	user id
<code>gid_t pw_gid</code>	group id
<code>char *pw_gecos</code>	real name (*)
<code>char *pw_dir</code>	home directory
<code>char *pw_shell</code>	shell program

# File /etc/passwd

---

Ciascun **record** del database degli **utenti** contiene i sette campi descritti nella tabella **separati** da ":".

Un esempio potrebbe essere:

**root:!:0:0:The superuser:/root:/bin/sh**

**nobody\*:65534:65534::/:**

**mariog:!:100:100:Mario Guarracino:/users/mariog:/bin/bash**

C'è di solito almeno una riga per l'utente **root**, il **superutente**.

## File /etc/passwd (cont.)

Il campo `password` può contenere una *copia crittografata* della *password dell'utente*. La codifica avviene mediante un algoritmo one way e restituisce un numero fissato di caratteri.

Se un utente ha una password di un solo carattere, nessuna password può coincidere con questo carattere.

## File /etc/passwd (cont.)

Alcuni **campi** possono essere **vuoti**.

Se il campo di shell è vuoto, di solito, viene attivata **/bin/sh**, oppure viene negato l'accesso al sistema, se manca la home directory la directory iniziale può essere /

Per i sistemi che supportano **finger(1)** è possibile inserire nel campo di commento il nome dell'utente, l'indirizzo, il numero di telefono dell'ufficio e quello privato, mediante il comando **chfn(1)**

## File /etc/passwd (cont.)

Per **accedere** alle informazioni contenute nel **database** degli **utenti** abbiamo bisogno di opportune **chiamate di sistema**:

- una funzione **get** per leggere il record successivo, che apre il file se necessario; tale funzione ritorna un puntatore ad una struttura, nullo quando si è raggiunta la fine del file
- una funzione **set** che apre il file (se non è già aperto) e si posiziona sul primo record
- una funzione **end** che chiude il file, da chiamare sempre quando sono finite le operazioni

# Funzioni **getpwent**, **setpwent** e **endpwent**

```
# include <pwd.h>
# include <sys/types.h>

struct passwd *getpwent(void);
void setpwent(void);
void endpwent(void);
```

La funzione **getpwent** ritorna uno alla volta i record contenuti nel file di password.; la funzione apre il file, se necessario.

Per ritornare al primo record si utilizza **setpwent** e per chiudere il file **endpwent**

# Funzioni `getpwuid` e `getpwnam`

```
# include <pwd.h>
# include <sys/types.h>

struct passwd *getpwuid(uid_t uid);
struct passwd *getpwnam(const char * name);
```

La funzione **getpwuid** ritorna un puntatore ad una struttura di tipo **passwd** per l'utente con user ID *uid*

La funzione **getpwnam** ritorna un puntatore ad una struttura di tipo **passwd** per l'utente di nome *name*

# Esempio: getpwnam.c

```
# include <sys/types.h>
```

```
# include <pwd.h>
```

```
# include <stddef.h>
```

```
# include <string.h>
```

```
struct passwd *getpwnam(const char *name)
```

```
{
```

```
    struct passwd *ptr;
```

```
    setpwent();
```

```
    while ( (ptr = getpwent()) != NULL) {
```

```
        if (strcmp(name, ptr->pw_name) == 0)
```

```
            break;          /* found a match */
```

```
    }
```

```
    endpwent();
```

```
    return(ptr);          /* ptr is NULL if no match
```

```
found */
```

```
}
```

## Esempio: getpwnam.c (cont.)

L'esempio mostra una possibile implementazione della funzione **getpwnam**.

Si noti la chiamata iniziale a **setpwent**, in modo da partire sempre dall'inizio del file, nel caso il chiamante avesse già eseguito una chiamata a **getpwent**, e chiude il file con **endpwent**.

# Shadow password

---

Il file `/etc/passwd` è un file leggibile da tutti gli utenti, in quanto sono conservate le informazioni su ciascuno di essi.

Storicamente in questo file si trovavano le password di ciascun utente, codificate mediante un algoritmo *one-way*.

Tale codifica non mette al riparo dall'uso di dizionari per cercare di carpire la password di un utente.

## Shadow password (cont.)

Per rendere più difficile il carpire una password, alcuni sistemi le conservano in un file non leggibile dagli utenti, detto file delle *shadow password*.

In questo file vengono conservate almeno il nome dell'utente e la sua password.

Alcuni sistemi richiedono che tale password sia cambiata ad intervalli regolari di tempo, che essa sia non troppo simile ad una utilizzata in precedenza, che non sia una parola di dizionario...

# File dei gruppi

---

Il file dei gruppi contiene il database dei gruppi supplementari. Ciascun record ha i seguenti campi, contenuti nella struttura **group** definita in **<grp.h>**:

Campo	Descrizione
<code>char *gr_name</code>	group name
<code>char *gr_passwd</code>	group password (*)
<code>gid_t gr_gid</code>	group id
<code>char **gr_mem</code>	group members

Il campo `gr_mem` è un array di puntatori ai nomi degli utenti che appartengono al gruppo, il cui ultimo elemento è un puntatore nullo.

## File dei gruppi (cont.)

---

Funzioni analoghe a quelle per gli utenti esistono per i gruppi.

```
# include <grp.h>
# include <sys/types.h>

struct group *getgrent(void);
void setgrent(void);
void endgrent(void);

struct group *getgrnam(const char *name);
struct group *getgrgid(gid_t gid);
```

# Gruppi supplementari

---

Ciascun **utente** appartiene al **gruppo** che corrisponde al **group ID** che si trova nel proprio record nel file di password.

Ciascun **utente** può però appartenere al più ad **altri** **NGROUPS\_MAX gruppi**, detti **gruppi supplementari**.

Il controllo dei **permessi** di accesso ad un file avviene quindi non solo rispetto al **gruppo** di appartenenza dell'utente, ma anche rispetto ai suoi **gruppi supplementari**.

## Gruppi supplementari (cont.)

Un programma può ottenere i gruppi supplementari utilizzando **getgroups**, che ritorna il numero di gruppi supplementari che ha messo nell'array *list*.

*size* indica il numero massimo di group ID da mettere in *list*.

Nel caso *size* è nullo, la funzione ritorna solo il numero di gruppi supplementari.

```
# include <unistd.h>
# include <sys/types.h>
int getgroups(int size, gid_t list[ ]);
```

# Altri file di dati

---

Il file delle password ed il file dei gruppi non sono gli unici file di dati di sistema.

Altri file sono utilizzati dal sistema Unix per gestire le sue operazioni.

Ad esempio il file di dati per i servizi forniti dai server di rete è `/etc/services`, per i protocolli è `/etc/protocols` e per gli host di una rete locale `/etc/hosts`.

## Altri file di dati (cont.)

---

Fortunatamente le **interfacce** con questi file sono **simili** a quelle descritte per i **file** delle **password** e dei gruppi:

- una funzione **get** per leggere il record successivo, che apre il file se necessario; tale funzione ritorna un puntatore ad una struttura, nullo quando si è raggiunta la fine del file
- una funzione **set** che apre il file (se non è già aperto) e si posiziona sul primo record
- una funzione **end** che chiude il file, da chiamare sempre quando sono finite le operazioni

## Altri file di dati (cont.)

---

Così come si può cercare un utente nel file di password facendo riferimento al suo user ID o al suo nome utente, ci sono funzioni che permettono la ricerca di un particolare record mediante una chiave.

<b>File</b>	<b>Header</b>	<b>Struttura</b>	<b>Funzioni</b>
<b>/etc/passwd</b>	<b>&lt;pwd.h&gt;</b>	<b>passwd</b>	<b>getpwnam getpwuid</b>
<b>/etc/group</b>	<b>&lt;grp.h&gt;</b>	<b>group</b>	<b>getgrnam getgrgid</b>
<b>/etc/hosts</b>	<b>&lt;netdb.h&gt;</b>	<b>hostent</b>	<b>gethostbyname gethostbyaddr</b>
<b>/etc/services</b>	<b>&lt;netdb.h&gt;</b>	<b>servent</b>	<b>getservbyname getservbyport</b>

# Log accounting

---

Alcuni sistemi mantengono traccia degli utenti collegati e di tutti i login e logout.

Ciascuna informazione viene memorizzata in un record:

```
struct utmp {  
    char  ut_user[8];      /* user login name */  
    char  ut_line[12];    /* device name (console, lnx) */  
    short ut_pid;        /* process id */  
    time_t ut_time;      /* time entry was made */  
};
```

Il comando **who**(1) legge i dati da **utmp**, mentre **last**(1) da **wtmp**

La collocazione dei file e la stessa struttura **utmp** dipendono dalla particolare implementazione.

# Identificazione del sistema

---

```
# include <sys/utsname.h>  
int uname(struct utsname *buf);
```

La funzione **uname** ritorna informazioni sull'host ed il suo sistema operativo.

# Identificazione del sistema (cont.)

L'informazione è ritornata nella struttura **utsname**.

I membri della struttura sono tutti array di caratteri, la cui lunghezza dipende dall'implementazione.

```
struct utsname {  
    char sysname[9];    /* Implementation of the OS */  
    char nodename[9]; /* This node on the network */  
    char release[9];  /* Current release level */  
    char version[9];  /* Current version level */  
    char machine[9];  /* Hardware type */  
    };
```

## Identificazione del sistema (cont.)

---

Alcuni sistemi sono provvisti della funzione **gethostname**, che ritorna il nome (*fully qualified domain name*) dell'host su di una rete TCP/IP.

```
# include <unistd.h>

int gethostname(char *name, size_t len);
int sethostname(const char *name, size_t len);
```

Il nome dell'host viene normalmente inizializzato in fase di boot mediante il comando **hostname(1)**.

# Funzioni per data e tempo

Il kernel di Unix fornisce un servizio per la gestione del tempo contando il numero di secondi che sono passati dalla mezzanotte del 1° Gennaio 1970 (UTC) (*the epoch*).

Questi secondi sono rappresentati in un dato di tipo **time\_t** e vengono detti *calendar times*. (rappresentano quindi sia la data, sia l'ora).

La data e l'ora sono ritornati dalla funzione **time**; se l'argomento è non nullo, tale valore è conservato nella locazione puntata da *calptr*.

```
# include <time.h>
time_t time(time_t *calptr);
```

## Funzioni per data e tempo (cont.)

Una volta ottenuto tale **tempo in secondi** si possono utilizzare altre funzioni per **convertire** l'informazione in un **formato** leggibile.

```
# include <time.h>
struct tm *localtime(const time_t *timep);
```

Questa funzione converte un calendar time in una struttura di tipo **tm**.

**localtime** inizializza la variabile esterna **tzname** con l'informazione sul fuso orario

# Funzioni per data e tempo (cont.)

La struttura tm viene inizializzata da **localtime**.

```
struct tm { /* broken-down time */
    int  tm_sec;           /* seconds after the minute - [0, 61] */
                          /* for leap seconds */
    int  tm_min;          /* minutes after the hour - [0, 59] */
    int  tm_hour;         /* hour since midnight - [0, 23] */
    int  tm_mday;         /* day of the month - [1, 31] */
    int  tm_mon;          /* months since January - [0, 11] */
    int  tm_year;         /* years since 1900 */
    int  tm_wday;         /* days since Sunday - [0, 6] */
    int  tm_yday;         /* days since January 1 - [0, 365] */
    int  tm_isdst;        /* flag for alternate daylight savings time */
};
```

I secondi possono essere più di 59 perché bisogna tenere in conto i secondi degli anni bisestili.

# Funzioni per data e tempo (cont.)

Per convertire una data (espressa come local time) in un valore **t\_time** si utilizza **mktime**:

```
#include <time.h>  
time_t mktime(struct tm *timeptr);
```

# Funzioni per data e tempo (cont.)

Le funzioni **asctime** e **ctime** ritornano una stringa di 26 caratteri del tipo:

Sun Sept 16 01:03:52 1973\n\0

```
#include <time.h>
char *ctime(const time_t *clock);
char *asctime(const struct tm *tm);
```

L'argomento di **asctime** è un puntatore ad una struttura di tipo `tm`, mentre per **ctime** è un puntatore ad un calendar time.

# Sommario

---

Il file di password e quello dei gruppi vengono utilizzati da tutti i sistemi Unix.

Abbiamo visto alcune funzioni che permettono di leggere questi file.

I gruppi supplementari possono essere uno strumento utile per il lavoro in gruppi.

Abbiamo visto altre funzioni che permettono di accedere ad altri file di sistema.

# Avviso

---

- Leggere il capitolo 6 dello Stevens
- Venerdì 11.1.2002 h.9:00 lab

# Esercizi

---

1. Utilizzando le funzioni `getpwent`, `setpwent` ed `endpwent` si implementi una funzione `int ordpw(const char *field, const char *path)`; che presi in input il nome di un campo della struttura `passwd` ritorni in `path` il file di password ordinato rispetto a tale campo.
2. Si implementi il comando `myid` che, preso come argomento lo username di un utente, ritorni sullo standard output il suo user id, group id e supplementary groupid.
3. Implementare la funzione `mygetpwuid`.
4. Si scriva un programma che chiama `uname` e stampa tutti i campi della struttura `utsname`. Si confronti il risultato con quello del comando `uname(1)`.