

# System Data Files and Information

---

Advanced Programming in the UNIX Environment

Chun-Ying Huang <chuang@cs.nctu.edu.tw>

# Outline

---

Overview

Password

Group

Accounting

System Identification

Time and Date Routines

# Overview

---

A UNIX system requires numerous data files for normal operation

- Password file: `/etc/passwd`
- Group file: `/etc/group`
- Many other system configuration files are placed in `/etc` as well

These data files are usually ASCII text files, and can be accessed with standard I/O library

A number of portable interfaces to these files are also provided for application programs

- Developers does not need to handle the underlying design and implementations
- Increase the program portability

# Who Am I?

---

The `id` program

```
$ id
uid=1000(chuang) gid=1000(chuang) groups=4(adm), 20(dialout),
24(cdrom), 46(plugdev), 108(lpadmin), 123(admin),
124(sambashare), 1000(chuang)
```

# The /etc/passwd File

---

Description	struct passwd member	POSIX.1	FreeBSD 8.0	Linux 3.2	Mac OS X 10.6.8	Solaris 10
User name	char *pw_name	•	•	•	•	•
Encrypted password	char *pw_passwd		•	•	•	•
Numerical user ID	uid_t pw_uid	•	•	•	•	•
Numerical group ID	gid_t pw_gid	•	•	•	•	•
Comment field	char *pw_gecos		•	•	•	•
Initial working dir	char *pw_dir	•	•	•	•	•
Initial shell	char *pw_shell	•	•	•	•	•
User access class	char *pw_class		•		•	
Next time to change password	time_t pw_change		•		•	
Account expiration time	time_t pw_expire		•		•	

# The /etc/passwd File (Cont'd)

---

```
root:x:0:0:root:/root:/bin/bash
squid:x:23:23::/var/spool/squid:/dev/null
nobody:x:65534:65534:Nobody:/home:/bin/sh
sar:x:205:105:Stephen Rago:/home/sar:/bin/bash
```

A simple example from Linux system

Password fields are separated by colons (:)

Fields can be empty

Many UNIX services have their own user id and group id

The shell is the first process that a user logs in

- Use something like /dev/null or /bin/false to prevent users from logging into the system

Valid shells (for users to choose from) are listed in /etc/shells

# The finger Command

---

Read user information from password database

finger command may be not available in your system

You can still read passwd files to obtain the information

```
sar:x:205:105:Steve Rago, SF 5-121, 555-1111, 555-2222:/home/sar:/bin/sh
```

```
$ finger -p sar
```

```
Login: sar
```

```
Name: Steve Rago
```

```
Directory: /home/sar
```

```
Shell: /bin/sh
```

```
Office: SF 5-121, 555-1111
```

```
Home Phone: 555-2222
```

```
On since Mon Jan 19 03:57 (EST) on ttyv0 (messages off) No Mail.
```

# Functions to Retrieve Password Information

---

Get password information for a specific user

```
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
struct passwd *getpwnam(const char *name);
```

Iteratively retrieve all password information

```
#include <pwd.h>
struct passwd *getpwent(void);
void setpwent(void);           // rewind
void endpwent(void);          // close the password file
```



# User and Shadow Passwords

---

Historically, encrypted passwords are stored in `/etc/passwd` file

But modern UNIX systems move the encrypted password into another secret file, which is only readable by system administrators

- FreeBSD: `/etc/master.passwd`
- Linux: `/etc/shadow`

Only a few programs need to access encrypted passwords, for example, `login(1)` and `passwd(1)`

There is not a shadow structure for FreeBSD and Mac OS X

# Linux's Shadow Structure

---

```
#include <shadow.h>
struct spwd *getspnam(const char *name);
struct spwd *getspent(void);
void setspent(void);
void endspent(void);
```

Description	struct spwd member
User login name	char *sp_namp
Encrypted password	char *sp_pwdp
Date of last password change ( <b>Days since Epoch</b> )	long sp_lastchg
Min # of days between changes	long sp_min
Max # of days between changes	long sp_max
# of days before password expires to warn user to change it	long sp_warn
# of days after password expires until account is disabled	long sp_inact
Date when account expires ( <b>Days since Epoch</b> )	long sp_expire
Reserved	long sp_flag

# Example: Linux Username and Password

---

## Files

- /etc/passwd and /etc/shadow

```
root:x:0:0:root:/root:/bin/bash
chuang:x:1000:1000:Chun-Ying Huang,,,:/home/chuang:/bin/bash
```



```
root!:14265:0:99999:7:::
chuang:$1$.oJz4T5J$XmVX77wOdYDHxyvTEcaLc1:14265:0:99999:7:::
```

# More on User Passwords

```
root:!:14265:0:99999:7:::  
chuang:$1$.oJz4T5J$XmVX77WodYDHxyvTEcaLc1:14265:0:99999:7:::
```



Passwords are *encrypted*, see `crypt(3)`

Binary data are converted to ASCII texts

- Each byte represents 6-bit data
- `const char map[] =  
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789./";`



**Salt** + **Encrypted password**

**Algorithm id** + **Salt** + **Encrypted password**

Common password algorithms

- DES, MD5, SHA256, SHA512

# Example: Password Algorithms

---

The password is 'hello'

DES: enc(salt, key = password):

- 56-bit key, 64-bit input, 64-bit output (11 bytes)
- mOeHQ3re1ro8s

MD5: hash(salt, password)

- 128-bit output (22 bytes)
- \$1\$18g4a5kf\$.92G5ZJvtDF/.WJbM.ef31

SHA256: hash(salt, password)

- 256-bit output (43 bytes)
- \$5\$aKM2MaGt\$NmJn1xu8kup8jl5SxJPBDhmFLH50nwQATB/72zVuH5

SHA512: hash(salt, password)

- 512-bit output (86 bytes)
- \$6\$CaP7vQ/f\$Puo5/OmR7P21D0BvqE15ZW4bqW4wPNKBGhj.kTUSwcfqj18wMdl36h2smX0ZUaT6buYKSeXhw13RR6oBpIfZv0

# The crypt(3) Function

---

The built-in password encryption function

```
#include <unistd.h>
char *crypt(const char *key, const char *salt);
```

Your program may have to link with the crypt library (-lcrypt)

```
printf("%s\n", crypt("password", "ab"));
printf("%s\n", crypt("password", "$1$abcdefgh$"));
printf("%s\n", crypt("password", "$5$abcdefgh$"));
printf("%s\n", crypt("password", "$6$abcdefgh$"));
```

# The /etc/group File

---

Description	struct group member	POSIX.1	FreeBSD 8.0	Linux 3.2	Mac OS X 10.6.8	Solaris 10
Group name	char *gr_name	•	•	•	•	•
Encrypted password	char *gr_passwd		•	•	•	•
Numerical group ID	gid_t gr_gid	•	•	•	•	•
Array of pointers to individual user names	char **gr_mem	•	•	•	•	•

An example: /etc/group

```
adm:x:4:chuang
dialout:x:20:chuang
cdrom:x:24:chuang
plugdev:x:46:chuang
lpadmin:x:108:chuang
admin:x:123:chuang
chuang:x:1000:chuang
sambashare:x:124:chuang
```

# Functions to Retrieve Group Information

---

Get group information for a specific group

```
#include <grp.h>
struct group *getgrgid(gid_t gid);
struct group *getgrnam(const char *name);
```

Iteratively retrieve all group information

```
#include <grp.h>
struct group *getgrent(void);
void setgrent(void);           // rewind
void endgrent(void);           // close the group file
```



# Supplement Group IDs

---

In the past, a UNIX user is belong to a single group at any time

Use the `newgrp(1)` command to switch between allowed groups

- Change the current effective GID

Later we have the concept of "supplement group IDs"

- A user has a default group ID (given in the `/etc/passwd` file)
- A user is also belong to a number of additional groups – the supplement group IDs
- The permission check are performed based on all the group IDs that the user belongs to
- The number of additional groups has a limit (`NGROUPS_MAX`), and a common value is 16

# Group Setup Functions

---

Standard POSIX.1 functions

```
#include <unistd.h>
int getgroups(int gidsetsize, gid_t grouplist[]);
```

Not in POSIX.1, but very common to most platforms

- `initgroups(3)` setup gid and supplement group IDs for a user based on `/etc/group`
- `setgroups(2)` setup supplement group IDs, usually called by `initgroups(3)`

```
#include <grp.h> /* on Linux */
#include <unistd.h> /* on FreeBSD, Mac OS X, and Solaris */
int setgroups(int ngroups, const gid_t grouplist[]);
```

```
#include <grp.h> /* on Linux and Solaris */
#include <unistd.h> /* on FreeBSD and Mac OS X */
int initgroups(const char *username, gid_t basegid);
```

# Implementation Differences

Information	FreeBSD 8.0	Linux 3.2.0	Mac OS 10.6.8	Solaris 10
Account information	/etc/passwd	/etc/passwd	Directory Services	/etc/passwd
Encrypted passwords	/etc/master.passwd	/etc/shadow	Directory Services	/etc/shadow
Hashed password files?	yes	no	no	no
Group information	/etc/group	/etc/group	Directory Services	/etc/group

The storage of users, groups, and passwords could be different on different platforms

Some UNIX systems implements user and group database using network information service (NIS) or lightweight directory access protocol (LDAP)

- You may have a look at /etc/nsswitch.conf

# Other Data Files

---

We have a number of other data files in the system

They have similar lookup functions to passwords and groups

Description	Data file	Header	Structure	Lookup functions
passwords	/etc/passwd	<pwd.h>	passwd	getpwnam, getpwuid
groups	/etc/group	<grp.h>	group	getgrnam, getgrgid
shadow	/etc/shadow	<shadow.h>	spwd	getspnam
hosts	/etc/hosts	<netdb.h>	hostent	gethostbyname, gethostbyaddr
networks	/etc/networks	<netdb.h>	netent	getnetbyname, getnetbyaddr
protocols	/etc/protocols	<netdb.h>	protoent	getprotobyname, getprotobynumber
services	/etc/services	<netdb.h>	servent	getservbyname, getservbyport

# Login Accounting

---

utmp: Record the currently logged in users

wtmp: Record the history of user login, logout, and system (up, down, or reboot) activities

The format and the location of utmp and wtmp records are diverse

- See utmp(5)
- Parts of a utmp record (textbook example)

```
struct utmp {  
    char ut_line[8]; /* tty line: "ttyh0", "ttyd0", "ttyp0", ... */  
    char ut_user[8]; /* login name */  
    long ut_time;    /* seconds since Epoch */  
};
```

Relevant commands

- w(1) and who(1)
- last(1)

# System Identification

---

The uname function

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

The utsname structure

```
struct utsname {
    char sysname[]; /* name of the operating system */
    char nodename[]; /* name of this node */
    char release[]; /* current release of operating system */
    char version[]; /* current version of this release */
    char machine[]; /* name of hardware type */
};
```

# System Identification

---

The uname command

- System
- Node
- Release
- Version
- Machine
- Processor
- Hardware platform
- Operating system

```
$ uname -a
```

```
Linux ubuntu-vm 3.16.0-62-generic #83~14.04.1-Ubuntu SMP  
Fri Feb 26 22:52:39 UTC 2016 x86_64 x86_64 x86_64  
GNU/Linux
```

# Time and Date Routines

---

We have introduced several time and date time routines

- `time(2)` get current time, in the unit of seconds from Epoch
- `gettimeofday(2)` get current time, in the unit of milliseconds from Epoch

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

```
#include <sys/time.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
```



# The tm Structure (1/4)

---

`gmtime(3)` and `localtime(3)` break down a `time_t` value into a structure

`mktime(3)` does the reverse


```
struct tm {
    int tm_sec;           /* seconds */
    int tm_min;           /* minutes */
    int tm_hour;          /* hours */
    int tm_mday;          /* day of the month */
    int tm_mon;           /* month */
    int tm_year;          /* year */
    int tm_wday;          /* day of the week */
    int tm_yday;          /* day in the year */
    int tm_isdst;         /* daylight saving time */
};
```

# The tm Structure (2/4)

---

Break down time\_t value

```
#include <time.h>
struct tm *gmtime(const time_t *calptr);
struct tm *gmtime_r(const time_t *calptr, struct *tm result);
struct tm *localtime(const time_t *calptr);
struct tm *localtime_r(const time_t *calptr, struct *tm result);
```



Print out time in a string

```
#include <time.h>
char *asctime(const struct tm *tmptr);
char *asctime_r(const struct tm *tmptr, char *buf);
char *ctime(const time_t *calptr);
char *ctime_r(const time_t *calptr, char *buf);
```

mktime

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

# The tm Structure (3/4)

Formatted output using strftime(3)

```
#include <time.h>
size_t strftime(char *buf, size_t maxsize, const char *format,
               const struct tm *tmptr);
```

The format string

Fmt	Description	Example	Fmt	Description	Example
%a	Abbr. weekday name	Tue	%d	Day of the month	10
%A	Full weekday name	Tuesday	%D	Date [MM/DD/YY]	02/10/04
%b	Abbr. month name	Feb	%e	Day of month	10
%B	Full month name	February	%F	ISO 8601: YYYY-MM-DD	2004-02-10
%c	Date and time	Tue Feb 10 18:27:38 2004	%g	Last two digits of a year	04
%C	Year/100	20	%G	ISO 8601 year	2004
			%h	Same as %b	Feb

# The tm Structure (4/4)

Fmt	Description	Example	Fmt	Description	Example
%H	Hour of the day (24-hr)	18	%u	ISO 8601 weekday: 1-7	2
%I	Hour of the day (12-hr)	06	%U	Sunday week number: 00-53	06
%j	Day of the year : 001-366	041	%V	ISO 8601 week number	07
%m	Month: 01-12	02	%w	Weekday: 0-6	2
%M	Minute: 00-59	27	%W	Monday week number: 0-53	06
%n	Newline character		%x	date	02/10/04
%p	AM/PM	PM	%X	time	18:27:38
%r	Locale's time (12-hr)	06:27:38 PM	%y	Last two digits of a year	04
%R	Same as "%H:%M"	18:27	%Y	Year	2004
%S	Second: 00-60	38	%z	Offset from UTC	-0500
%t	Horizontal tab character		%Z	Time zone name	EST
%T	Same as "%H:%M:%S"	18:27:38	%%	Percent sign	%

# Time Function Examples

---

(times.c)

```
time_t t = time(0);
struct tm tm1, tm2;
char buf[256];

gmtime_r(&t, &tm1);
localtime_r(&t, &tm2);

printf("                time: %ld\n", t);
printf("                ctime: %s", ctime_r(&t, buf));
printf("    gmtime -> asctime: %s", asctime(&tm1));
printf("    localtime -> asctime: %s", asctime(&tm2));
strftime(buf, sizeof(buf), "%c %Z (%z)", &tm1);
printf("    gmtime -> strftime: %s\n", buf);
strftime(buf, sizeof(buf), "%c %Z (%z)", &tm2);
printf("localtime -> strftime: %s\n", buf);
```

# Running the Example

---

```
$ ./times
```

```
        time: 1457769624
        ctime: Sat Mar 12 16:00:24 2016
    gmtime -> asctime: Sat Mar 12 08:00:24 2016
localtime -> asctime: Sat Mar 12 16:00:24 2016
    gmtime -> strftime: Sat Mar 12 08:00:24 2016 GMT (+0000)
localtime -> strftime: Sat Mar 12 16:00:24 2016 CST (+0800)
```

# Time Zone (1/3)

---

The TZ environment variable

Formats: from `tzset(3)`

Standard:

- `std offset`
- Example: `CST-08:00:00`, `PST08:00:00`, `NSDT-13:00:00`

Daylight saving time:

- `std offset dst [offset],start[/time],end[/time]`
- Example: `NZST-12:00:00NZDT-13:00:00,M10.1.0,M3.3.0`

Predefined: files stored in `/usr/share/zoneinfo`

- `:filename` (relative path to `/usr/share/zoneinfo`)
- Example: `:Asia/Taipei`, `:America/Vancouver`, `:NZ`

# Time Zone (2/3)

---

The times.c example with time zone settings (New Zealand)

```
$ TZ=":NZ" ./times
      time: 1457771092
      ctime: Sat Mar 12 21:24:52 2016
    gmtime -> asctime: Sat Mar 12 08:24:52 2016
    localtime -> asctime: Sat Mar 12 21:24:52 2016
      gmtime -> strftime: Sat Mar 12 08:24:52 2016 GMT (+0000)
    localtime -> strftime: Sat Mar 12 21:24:52 2016 NZDT (+1300)
```

```
$ TZ="NZST-12:00:00NZDT-13:00:00,M10.1.0,M3.3.0" ./times      # Mmonth.week.weekday
      time: 1457771128
      ctime: Sat Mar 12 21:25:28 2016      # /time: default is 02:00:00
    gmtime -> asctime: Sat Mar 12 08:25:28 2016
    localtime -> asctime: Sat Mar 12 21:25:28 2016
      gmtime -> strftime: Sat Mar 12 08:25:28 2016 GMT (+0000)
    localtime -> strftime: Sat Mar 12 21:25:28 2016 NZDT (+1300)
```



# Time Zone (3/3)

---

```
$ TZ="NSDT-13:00:00" ./times
      time: 1457771162
      ctime: Sat Mar 12 21:26:02 2016
    gmtime -> asctime: Sat Mar 12 08:26:02 2016
  localtime -> asctime: Sat Mar 12 21:26:02 2016
    gmtime -> strftime: Sat Mar 12 08:26:02 2016 GMT (+0000)
  localtime -> strftime: Sat Mar 12 21:26:02 2016 NSDT (+1300)
```

```
$ TZ="ABC-13:00:00" ./times
      time: 1457771300
      ctime: Sat Mar 12 21:28:20 2016
    gmtime -> asctime: Sat Mar 12 08:28:20 2016
  localtime -> asctime: Sat Mar 12 21:28:20 2016
    gmtime -> strftime: Sat Mar 12 08:28:20 2016 GMT (+0000)
  localtime -> strftime: Sat Mar 12 21:28:20 2016 ABC (+1300)
```

# Q & A

---