Lecture 8. System Data Files

The Password File

Traditionally, on UNIX systems the password file was in:

`/etc/passwd`.

On MacOS, it is in:

`/var/db/dslocal/nodes/Default/users/<accountname>.plist`.

Header file pwd.h (in `/usr/include/pwd.h`) contains the structure of entries from the password file:

```
struct passwd
{
        char    *pw_name;               /* user name */
        char    *pw_passwd;             /* encrypted password */
        uid_t   pw_uid;                 /* user uid */
        gid_t   pw_gid;                 /* user gid */
        __darwin_time_t pw_change;      /* password change time */
        char    *pw_class;              /* user access class */
        char    *pw_gecos;              /* Honeywell login info */
        char    *pw_dir;                /* home directory */
        char    *pw_shell;              /* default shell */
        __darwin_time_t pw_expire;      /* account expiration */
};
```

GECOS

The original version of GCOS was developed by General Electric from 1962; originally called **GECOS** (the **G**eneral **E**lectric **C**omprehensive **O**perating **S**upervisor). The operating system is still used today in its most recent versions (GCOS 7 and GCOS 8) on servers and mainframes produced by Honeywell (GCOS 8) and Groupe Bull (GCOS 7), primarily through emulation, to provide continuity with legacy mainframe environments.

Program languages available for GCOS included GCOS Algol, Algol-68, COBOL, SNOBOL, JOVIAL, APL, FORTRAN 68, CORAL 66 and FORTRAN 77. (from Wikipedia).

The entries to the `passwd` struct correspond to the POSIX.1 specification as follows:

Description	`struct passwd` member	POSIX.1	FreeBSD 5.2.1	Linux 2.4.22	Mac OS X 10.3	Solaris 9
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 directory	`char *pw_dir`	•	•	•	•	•
initial shell (user program)	`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`		•		•	

There are two functions providing entries from the passwd file, returning a user name and a user ID:

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

struct passwd *getpwuid(uid_t uid);

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

Both return pointer to struct if OK, `NULL` if error.

`getpwuid()` is used by `ls`, to get the user ID from the i-node translated into a user's login name. `getpwnam()` is used by `login`, when a user enters his login.

```
#include <unistd.h>

uid_t getuid(void);
```

Returns real user ID of calling process.

```
#include <unistd.h>

uid_t geteuid(void);
```

Returns effective user ID of calling process.

The following program will display the values of the `passwd` structure for me:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <pwd.h>

int main(void)
{
     struct passwd *pwd;

     uid_t UID=getuid();
     printf("UID=%d\n", UID);

     uid_t eUID=geteuid();
     printf("UID=%d\n", eUID);

     pwd=getpwuid(UID);


printf("-------------------------------------------------------
-\n");
     printf("|\t user name\t\t| %20s \t|\n", pwd->pw_name);
     printf("|\t encrypted password\t| %20s \t|\n", pwd-
>pw_passwd);
     printf("|\t numerical user ID\t| %20u \t|\n", pwd->pw_uid);
     printf("|\t numerical group ID\t| %20u \t|\n", pwd-
>pw_gid);
     printf("|\t comment field\t\t| %20s \t|\n", pwd->pw_gecos);
     printf("|\t home directory\t\t| %20s \t|\n", pwd->pw_dir);
     printf("|\t default shell\t\t| %20s \t|\n", pwd->pw_shell);
     printf("|\t user class\t\t| %20s \t|\n", pwd->pw_class);

printf("-------------------------------------------------------
-\n");

return 0;
}
```

The output for user awise and user root is:

```
Adrianas-MacBook-Pro:Lecture8 awise$ ./mypwd2
UID=501
UID=501
-----------------------------------------------------
|    user name            |                  awise   |
|    encrypted password   |               ********   |
|    numerical user ID    |                    501   |
|    numerical group ID   |                     20   |
|    comment field        |          Adriana Wise    |
|    home directory       |          /Users/awise    |
|    default shell        |             /bin/bash    |
|    user class           |                          |
-----------------------------------------------------
Adrianas-MacBook-Pro:Lecture8 awise$ sudo ./mypwd2
Password:
UID=0
UID=0
-----------------------------------------------------
|    user name            |                   root   |
|    encrypted password   |                      *   |
|    numerical user ID    |                      0   |
|    numerical group ID   |                      0   |
|    comment field        | System Administrator     |
|    home directory       |              /var/root   |
|    default shell        |                /bin/sh   |
|    user class           |                          |
-----------------------------------------------------
```

When a program needs to go through the entire password file, the following password entry (hence `pwent`) functions can be used:

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

struct passwd *getpwent(void);
```

Returns pointer to struct passwd if OK, `NULL` if error.

```
void setpwent(void);

void endpwent(void);
```

`getpwent()` returns the next entry into the passwd file. `setpwent()` rewinds the file and `endpwent()` closes it.

The following returns the user info for a given user name (e.g. "awise"). The difference from the previous example is that the user name is supplied as a string to function `getpwnam()`:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <pwd.h>

int main(void)
{
     struct passwd *pwd;
     char *Uname="awise";

     pwd=getpwnam(Uname);

printf("-----------------------------------------------------
-\n");
     printf("|\t user name\t\t| %20s \t|\n", pwd->pw_name);
     printf("|\t encrypted password\t| %20s \t|\n", pwd-
>pw_passwd);
     printf("|\t numerical user ID\t| %20u \t|\n", pwd->pw_uid);
     printf("|\t numerical group ID\t| %20u \t|\n", pwd-
>pw_gid);
     printf("|\t comment field\t\t| %20s \t|\n", pwd->pw_gecos);
     printf("|\t home directory\t\t| %20s \t|\n", pwd->pw_dir);
     printf("|\t default shell\t\t| %20s \t|\n", pwd->pw_shell);
     printf("|\t user class\t\t| %20s \t|\n", pwd->pw_class);

printf("-----------------------------------------------------
-\n");
}
```

The `getpwent()` function uses the `/etc/passwd` file. The following program will output the entries in the `/etc/passwd` file:

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

int main(void)
{
     struct passwd *ptr;
     setpwent();
```

```
     while ((ptr=getpwent())!=NULL)
     {
          printf("user %s name %s\n", ptr->pw_name,
                    ptr>pw_gecos);
     }
     endpwent();
return 0;
}
```

These entries are utility programs, not users in the traditional sense. This is because no program is allowed to list all users on a system, as a security feature. Rather, the `getpwuid()` and family will return information on a given user.

Shadow Passwords

The encryption algorithm used to encrypt the plaintext user password into the encrypted version stored in the passwd file is one-way, i.e. the encryption function is not invertible. However, forward guesses are possible, knowing that users will choose non-random passwords, since they are easier to remember. To make it harder to decrypt using a forward guess, some systems store the encrypted password in a different file, the shadow password file. The shadow password file contains the user ID, the encrypted password, and the periodicity of need for password renewal.

The shadow password file is not world-readable. The few programs that need to access it, such as login, passwd, are set-user-ID root (have effective user ID root).

Description	`struct spwd` member
user login name	`char *sp_namp`
encrypted password	`char *sp_pwdp`
days since Epoch of last password change	`int sp_lstchg`
days until change allowed	`int sp_min`
days before change required	`int sp_max`
days warning for expiration	`int sp_warn`
days before account inactive	`int sp_inact`
days since Epoch when account expires	`int sp_expire.`
reserved	`unsigned int sp_flag`

Group File

The group file, similar to the passwd file, contains fields corresponding to the following entries in the `grp.h` file (`/usr/include/grp.h`):

```
struct group
{
        char    *gr_name;           /* [XBD] group name */
        char    *gr_passwd;         /* [???] group password */
        gid_t   gr_gid;             /* [XBD] group id */
        char    **gr_mem;           /* [XBD] group members */
};
```

Similar to the `getpwuid()` function, which works on a given UID, the following two functions work on a given GID, returning the relevant information:

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

struct group *getgrgid(gid_t gid);

struct group *getgrnam(const char *name);
```

Both return pointer to `struct group` if OK, `NULL` if error.

The arguments can be either a known GID, or a known group name. Let's list all the values for the fields in the group structure, using the following program:

```
#include <sys/types.h>
#include <grp.h> /* for getgrgid() and getgrnam() */
#include <unistd.h> /* for getgid() */
#include <stdio.h> /* for printf() */

int main(void)
{
     struct group *ptrID, *ptrName;
     char **userIterator;

     char *uName="awise";
     char *gName="staff";
     gid_t GID;

     GID=getgid();
     ptrID=getgrgid(GID);
```

```
     ptrName=getgrnam(gName);

     printf("User %s belongs to group %s, with GID=%u\n", uName,
gName, GID);

     printf("-----------------------------------------------
\n");
     printf("|\tgroup name\t| %20s |\n", ptrID->gr_name);
     printf("|\tgroup password\t| %20s |\n", ptrID->gr_passwd);
     printf("|\tgroup ID\t| %20d |\n", ptrID->gr_gid);

     for (userIterator=ptrID->gr_mem; *userIterator!=NULL;
userIterator++)
     {
          printf("|\tgroup members\t| %20s |\n", *userIterator);
     }
     printf("-----------------------------------------------
\n");

return 0;
}
```

The output is:

```
Adrianas-MacBook-Pro:Lecture8 awise$ ./mygrpid
User awise belongs to group staff, with GID=20
-------------------------------------------
|     group name      |                staff |
|     group password  |                    * |
|     group ID        |                   20 |
|     group members   |                 root |
-------------------------------------------
```

To search the entire group file, as opposed to do a search on a give GID or group name, we use the following functions:

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

struct group *getgrent(void);
```

Returns pointer if OK, `NULL` if error.

```
void setgrent(void);

void endgrent(void);
```

`setgrent()` opens the group file and rewinds it. `getgrent()` reads the next entry, returning a pointer to a group structure (opening the group file first, if not already open). `endgrent()` closes the group file.

The following program illustrates the use of `getgrent()` on my system, obtaining all the groups with their information from the group structure, as well as all the users belonging to each group.

The output is too long to include in these lecture notes, and will be shown in class in full. An excerpt including the members of group admin is included at the end.

```
#include <unistd.h>
#include <grp.h> /* for setgrent(), getgrent(), endgrent() */
#include <stdio.h>

int main(void)
{
     struct group *ptr;
     char **userIterator;

     setgrent();
     while ((ptr=getgrent())!=NULL)
     {
          //printf("group %s name %s\n", ptr->gr_name,
               ptr->gr_passwd);

          printf("-------------------------------------\n");
          printf("|\tgroup name\t| %20s |\n", ptr->gr_name);
          printf("|\tgroup password\t| %20s |\n",
               ptr->gr_passwd);
          printf("|\tgroup ID\t| %20d |\n", ptr->gr_gid);

          for (userIterator=ptr->gr_mem; *userIterator!=NULL;
               userIterator++)
          {
                    printf("|\tgroup members\t| %20s |\n",
                         *userIterator);
          }

     }
     endgrent();
return 0;
}
```

An excerpt of the output:

```
-----------------------------------------
|    group name     |              admin |
|    group password |                  * |
|    group ID       |                 80 |
|    group members  |               root |
|    group members  |              awise |
-----------------------------------------
```

Of course, to output information relevant to the current user, we needed the following two utility functions:

```
#include <unistd.h>

gid_t getgid(void);

gid_t getegid(void);
```

Return the real GID, and effective GID of calling process, respectively.

Supplementary Group IDs

4.2BSD added the concept of supplementary group IDs, meaning a user could belong to more than one group (the one listed in the passwd entry), but to 16 additional groups. The checks for file access permissions for a user were changed to not only compare the user effective group ID against a file's group ID, but also all the supplementary group IDs. This way, a user may belong to multiple groups at the same time, and in order to obtain permissions on a file, his/her group ID needs not be changed explicitly. The following functions return and set the supplementary group IDs:

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

int getgroups(int gidsetsize, gid_t grouplist[]);
```

Returns # of supplementary group IDs if OK, -1 if error.

```
int setgroups(int ngroups, const gid_t grouplist[]);

int initgroups(const char *username, gid_t basegid);
```

Both return 0 if OK, -1 if error.

Arguments:

- *gidsetsize* — max # of elements storable in the grouplist array, NGROUPS_MAX. If `gidsetsize=0`, `getgroups()` returns # of existent supplementary group IDs;
- *grouplist*`[]` — array of supplementary group IDs, gets filled in by `getgroups()`. If specified with `setgroups()`, provides the supplementary group IDs to add;
- *ngroups* — # of groups in array *grouplist*`[]`;
- *basegid* — the default user group from the `/etc/passwd` file.

`initgroups()` returns all groups as we implemented before.

The following program returns all the groups the current user (me) is part of:

```
#include <sys/types.h>
#include <sys/param.h> /* for NGROUPS_MAX */
#include <limits.h>
#include <unistd.h>
#include <stdio.h>

//#define NGROUPS_MAX 16

int main(void)
{
     int num_groups;
     gid_t grouplist[NGROUPS_MAX];
     gid_t *gi;

     num_groups=getgroups(NGROUPS_MAX, grouplist);
     printf("There are %d supplementary groups\n", num_groups);
     /*
     for (gi=grouplist; gi!=NULL; gi++)
     {
          printf("group %d\n", *gi);
     }
     */
     for (int i=0; i<num_groups; i++)
     {
          printf("group %d\n", grouplist[i]);
     }
return 0;
}
```

Other Data Files

In addition to the passwd and group files, there are many other system data files, for example `/etc/services` (data file for the services provided by the various network servers), `/etc/protocols` (data file for the protocols provided by same), or `/etc/networks` (networks data file). The interface to accessing information from these files is the same as for passwd and group.

Each data file has 3 functions:

1. A `get()` function that reads the next record, opening the data file if necessary. The `get()` function returns a pointer to a structure containing the entries into the data file. The structure is static, meaning it retains value until the next call of the function, so it must be saved on each call.
2. A `set()` function that opens the file (if not already open) and rewinds the file. This is used when we need to traverse the file from the beginning.
3. An `end()` function that closes the file.

As with the password look-up function by user ID and user name (remember `getpwuid()` and `getpwnam()`?), system data files may support similar keyed look-up functions. The following table shows these functions:

Description	Data file	Header	Structure	Additional keyed 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

In particular, two data files keep track of users on a UNIX system:

- `utmp` — tracks users currently logged in
- `wtmp` — tracks all logins and logouts

Each of these files writes into the following structure:

```
struct utmp
{
     char ut_line[8]; /* tty line: “ttyh0”, “ttyd0”, “ttyp0”, …
*/
     char ut_name[8]; /* login name */
     long ut_time; /* seconds since Epoch */
};
```

On each login, a `utmp` structure is written into both the `utmp` file and the `wtmp` file by the login program. On logout, the entry into `utmp` is erased by the init process, and a logout entry is appended to `wtmp`. The logout entry has the `ut_name` field zeroed out. The `wtmp` file also logs system reboots, and system time changes. The `utmp` file is used by the `who` utility:

```
Adrianas-MacBook-Pro:Lecture8 awise$ who
awise    console  Feb 24 17:23
awise    ttys000  Feb 24 17:24
awise    ttys001  Feb 24 17:24
awise    ttys002  Feb 24 17:24
awise    ttys003  Feb 24 17:24
awise    ttys004  Feb 24 17:24
```

On FreeBSD, Linux, and MacOS utmp(5) gives the field formats of `utmp` and `wtmp`. On this system, these files are on `/var/run/utmps` and `/var/log/wtmp`.

System Identification

The `uname()` function returns information about the host and the OS:

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

int uname(struct utsname *name);
```

Returns a non-negative value if OK, -1 if error.

The argument name fills in the following 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 */
};
```

The following code lists information about my system, printing out values of the members of the utsname structure:

```
#include <sys/utsname.h> /* for uname() */
#include <stdio.h> /* for printf() */
#include <stdlib.h> /* for malloc() */

int main(void)
{
     struct utsname ptr;
     int n;

     n=uname(&ptr);

     printf("OS: %50s\n", ptr.sysname);
     printf("node: %50s\n", ptr.nodename);
     printf("release: %50s\n", ptr.release);
     printf("version: %50s\n", ptr.version);
     printf("machine: %50s\n", ptr.machine);
return 0;
}
```

The output is:

```
Adrianas-MacBook-Pro:Lecture8 awise$ ./mysysinfo2
OS:                                                 Darwin
node:                           Adrianas-MacBook-Pro.local
release:                                              13.4.0
version: Darwin Kernel Version 13.4.0: Sun Aug 17 19:50:11 PDT
2014; root:xnu-2422.115.4~1/RELEASE_X86_64
machine:                                              x86_64
```

Finally, `gethostname()` returns the hostname of the machine it's run on:

```
#include <unistd.h>

int gethostname(char *name, int namelen);
```

Returns 0 if OK, -1 if error.

The following code returns the name of my machine:

```
#include <unistd.h> /* for gethostname() */
#include <stdio.h> /* for printf() */
#include <limits.h> /* for _POSIX_HOST_NAME_MAX */

int main(void)
{
     int n;
     char *sys_name;

     n=gethostname(sys_name, _POSIX_HOST_NAME_MAX);
     printf("Hostname: %s\n", sys_name);
return 0;
}
```

The output is (duh):

```
Adrianas-MacBook-Pro:Lecture8 awise$ ./myhostname
Hostname: Adrianas-MacBook-Pro.local
```

Homework (due Monday, Mar-8-2016):

1. Write a program that finds all users on a system. You should test for a large number of UID numbers, since no function returns that information wholesale.

2. Modify the program that returns supplementary GIDs for a user to also list these groups' names.