40

LOGIN ACCOUNTING

Login accounting is concerned with recording which users are currently logged in to the system, and recording past logins and logouts. This chapter looks at the login accounting files and the library functions used to retrieve and update the information they contain. We also describe the steps that an application providing a login service should perform in order to update these files when a user logs in and out.

40.1 Overview of the `utmp` and `wtmp` Files

UNIX systems maintain two data files containing information about users logging in and out of the system:

- The `utmp` file maintains a record of users currently logged in to the system (as well as certain other information that we describe later). As each user logs in, a record is written to the `utmp` file. One of the fields in this record, *ut_user*, records the login name of the user. This record is later erased on logout. Programs such as *who(1)* use the information in the `utmp` file to display a list of currently logged-in users.
- The `wtmp` file is an audit trail of all user logins and logouts (as well as certain other information that we describe later). On each login, a record containing the same information as is written to the `utmp` file is appended to the `wtmp` file. On logout, a further record is appended to the file. This record contains the same information, except that the *ut_user* field is zeroed out. The *last(1)* command can be used to display and filter the contents of the `wtmp` file.

On Linux, the utmp file resides at /var/run/utmp, and the wtmp file resides at /var/log/wtmp. In general, applications don't need to know about these pathnames, since they are compiled into *glibc*. Programs that do need to refer to the locations of these files should use the _PATH_UTMP and _PATH_WTMP pathname constants, defined in <paths.h> (and <utmpx.h>), rather than explicitly coding pathnames into the program.

> SUSv3 doesn't standardize any symbolic names for the pathnames of the utmp and wtmp files. The names _PATH_UTMP and _PATH_WTMP are used on Linux and the BSDs. Many other UNIX implementations instead define the constants UTMP_FILE and WTMP_FILE for these pathnames. Linux also defines these names in <utmp.h>, but doesn't define them in <utmpx.h> or <paths.h>.

40.2 The *utmpx* API

The utmp and wtmp files have been present in the UNIX system since early times, but underwent steady evolution and divergence across various UNIX implementations, especially BSD versus System V. System V Release 4 greatly extended the API, in the process creating a new (parallel) *utmpx* structure and associated utmpx and wtmpx files. The letter *x* was likewise included in the names of header files and additional functions for processing these new files. Many other UNIX implementations also added their own extensions to the API.

In this chapter, we describe the Linux *utmpx* API, which is a hybrid of the BSD and System V implementations. Linux doesn't follow System V in creating parallel utmpx and wtmpx files; instead, the utmp and wtmp files contain all of the required information. However, for compatibility with other UNIX implementations, Linux provides both the traditional *utmp* and the System V–derived *utmpx* APIs for accessing the contents of these files. On Linux, these two APIs return exactly the same information. (One of the few differences between the two APIs is that the *utmp* API contains reentrant versions of a few functions, while the *utmpx* API does not.) However, we confine our discussion to the *utmpx* interface, since that is the API specified in SUSv3 and is thus preferred for portability to other UNIX implementations.

The SUSv3 specification doesn't cover all aspects of the *utmpx* API (e.g., the locations of the utmp and wtmp files are not specified). The precise contents of the login accounting files differ somewhat across implementations, and various implementations provide additional login accounting functions that are not specified in SUSv3.

> Chapter 17 of [Frisch, 2002] summarizes some of the variations in the location and use of the wtmp and utmp files across different UNIX implementations. It also describes the use of the *ac(1)* command, which can be used to summarize login information from the wtmp file.

40.3 The *utmpx* Structure

The utmp and wtmp files consist of *utmpx* records. The *utmpx* structure is defined in <utmpx.h>, as shown in Listing 40-1.

The SUSv3 specification of the *utmpx* structure doesn't include the *ut_host*, *ut_exit*, *ut_session*, or *ut_addr_v6* fields. The *ut_host* and *ut_exit* fields are present on most other implementations; *ut_session* is present on a few other implementations; and *ut_addr_v6* is Linux-specific. SUSv3 specifies the *ut_line* and *ut_user* fields, but leaves their lengths unspecified.

The *int32_t* data type used to define the *ut_addr_v6* field of the *utmpx* structure is a 32-bit integer.

Listing 40-1: Definition of the *utmpx* structure

```
#define _GNU_SOURCE             /* Without _GNU_SOURCE the two field
struct exit_status {               names below are prepended by "__" */
    short e_termination;        /* Process termination status (signal) */
    short e_exit;               /* Process exit status */
};

#define __UT_LINESIZE    32
#define __UT_NAMESIZE    32
#define __UT_HOSTSIZE   256

struct utmpx {
    short ut_type;                      /* Type of record */
    pid_t ut_pid;                       /* PID of login process */
    char  ut_line[__UT_LINESIZE];       /* Terminal device name */
    char  ut_id[4];                     /* Suffix from terminal name, or
                                           ID field from inittab(5) */
    char  ut_user[__UT_NAMESIZE];       /* Username */
    char  ut_host[__UT_HOSTSIZE];       /* Hostname for remote login, or kernel
                                           version for run-level messages */
    struct exit_status ut_exit;         /* Exit status of process marked
                                           as DEAD_PROCESS (not filled
                                           in by init(8) on Linux) */
    long  ut_session;                   /* Session ID */
    struct timeval ut_tv;               /* Time when entry was made */
    int32_t ut_addr_v6[4];              /* IP address of remote host (IPv4
                                           address uses just ut_addr_v6[0],
                                           with other elements set to 0) */
    char __unused[20];                  /* Reserved for future use */
};
```

Each of the string fields in the *utmpx* structure is null-terminated unless it completely fills the corresponding array.

For login processes, the information stored in the *ut_line* and *ut_id* fields is derived from the name of the terminal device. The *ut_line* field contains the complete filename of the terminal device. The *ut_id* field contains the suffix part of the filename—that is, the string following *tty*, *pts*, or *pty* (the last two are for System-V and BSD-style pseudoterminals, respectively). Thus, for the terminal `/dev/tty2`, *ut_line* would be *tty2* and *ut_id* would be *2*.

In a windowing environment, some terminal emulators use the *ut_session* field to record the session ID for the terminal window. (Refer to Section 34.3 for an explanation of session IDs.)

The *ut_type* field is an integer defining the type of record being written to the file. The following set of constants (with their corresponding numeric values shown in parentheses) can be used as values for this field:

`EMPTY` (0)
: This record doesn't contain valid accounting information.

`RUN_LVL` (1)
: This record indicates a change in the system's run-level during system startup or shutdown. (Information about run-levels can be found in the *init(8)* manual page.) The `_GNU_SOURCE` feature test macro must be defined in order to obtain the definition of this constant from `<utmpx.h>`.

`BOOT_TIME` (2)
: This record contains the time of system boot in the *ut_tv* field. The usual author of `RUN_LVL` and `BOOT_TIME` records is *init*. These records are written to both the `utmp` file and the `wtmp` file.

`NEW_TIME` (3)
: This record contains the new time after a system clock change, recorded in the *ut_tv* field.

`OLD_TIME` (4)
: This record contains the old time before a system clock change, recorded in the *ut_tv* field. Records of type `OLD_TIME` and `NEW_TIME` are written to the `utmp` and `wtmp` files by the NTP (or a similar) daemon when it makes changes to the system clock.

`INIT_PROCESS` (5)
: This is a record for a process spawned by *init*, such as a *getty* process. Refer to the *inittab(5)* manual page for details.

`LOGIN_PROCESS` (6)
: This is a record for a session leader process for a user login, such as a *login(1)* process.

`USER_PROCESS` (7)
: This is a record for a user process, usually a login session, with the username appearing in the *ut_user* field. The login session may have been started by *login(1)* or by some application offering a remote login facility, such as *ftp* or *ssh*.

`DEAD_PROCESS` (8)
: This record identifies a process that has exited.

We show the numeric values of these constants because various applications depend on the constants having the above numerical order. For example, in the source code of the *agetty* program, we find checks such as the following:

```
utp->ut_type >= INIT_PROCESS && utp->ut_type <= DEAD_PROCESS
```

Records of the type `INIT_PROCESS` usually correspond to invocations of *getty(8)* (or a similar program, such as *agetty(8)* or *mingetty(8)*). On system boot, the *init* process

creates a child for each terminal line and virtual console, and each child execs the *getty* program. The *getty* program opens the terminal, prompts the user for a login name, and then execs *login(1)*. After successfully validating the user and performing various other steps, *login* forks a child that execs the user's login shell. The complete life of such a login session is represented by four records written to the `wtmp` file in the following order:

- an `INIT_PROCESS` record, written by *init*;
- a `LOGIN_PROCESS` record, written by *getty*;
- a `USER_PROCESS` record, written by *login*; and
- a `DEAD_PROCESS` record, written by *init* when it detects the death of the child *login* process (which occurs on user logout).

Further details on the operation of *getty* and *login* during user login can be found in Chapter 9 of [Stevens & Rago, 2005].

> Some versions of *init* spawn the *getty* process before updating the `wtmp` file. Consequently, *init* and *getty* race with each other to update the `wtmp` file, with the result that the `INIT_PROCESS` and `LOGIN_PROCESS` records may be written in the opposite order from that described in the main text.

40.4 Retrieving Information from the `utmp` and `wtmp` Files

The functions described in this section retrieve records from files containing *utmpx*-format records. By default, these functions use the standard `utmp` file, but this can be changed using the *utmpxname()* function (described below).

These functions employ the notion of a *current location* within the file from which they are retrieving records. This location is updated by each function.

The *setutxent()* function rewinds the `utmp` file to the beginning.

```
#include <utmpx.h>

void setutxent(void);
```

Normally, we should call *setutxent()* before employing any of the *getutx*()* functions (described below). This prevents possible confusion that might result if some third-party function that we have called has previously made use of these functions. Depending on the task being performed, it may also be necessary to call *setutxent()* again at appropriate points later in a program.

The *setutxent()* function and the *getutx*()* functions open the `utmp` file if it is not already open. When we have finished using the file, we can close it with the *endutxent()* function.

```
#include <utmpx.h>

void endutxent(void);
```

The *getutxent()*, *getutxid()*, and *getutxline()* functions read a record from the `utmp` file and return a pointer to a (statically allocated) *utmpx* structure.

```
#include <utmpx.h>

struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *ut);
struct utmpx *getutxline(const struct utmpx *ut);
```

All return a pointer to a statically allocated *utmpx* structure, or `NULL` if no matching record or EOF was encountered

The *getutxent()* function retrieves the next sequential record from the `utmp` file. The *getutxid()* and *getutxline()* functions perform searches, starting from the current file location, for a record matching the criteria specified in the *utmpx* structure pointed to by the *ut* argument.

The *getutxid()* function searches the `utmp` file for a record based on the values specified in the *ut_type* and *ut_id* fields of the *ut* argument:

- If the *ut_type* field is `RUN_LVL`, `BOOT_TIME`, `NEW_TIME`, or `OLD_TIME`, then *getutxid()* finds the next record whose *ut_type* field matches the specified value. (Records of these types don't correspond to user logins.) This permits searches for records of changes to the system time and run-level.
- If the *ut_type* field is one of the remaining valid values (`INIT_PROCESS`, `LOGIN_PROCESS`, `USER_PROCESS`, or `DEAD_PROCESS`), then *getutxent()* finds the next record whose *ut_type* field matches *any* of these values and whose *ut_id* field matches that specified in its *ut* argument. This permits scanning the file for records corresponding to a particular terminal.

The *getutxline()* function searches forward for a record whose *ut_type* field is either `LOGIN_PROCESS` or `USER_PROCESS`, and whose *ut_line* field matches that specified in the *ut* argument. This is useful for finding records corresponding to user logins.

Both *getutxid()* and *getutxline()* return `NULL` if the search fails (i.e., end-of-file is encountered without finding a matching record).

On some UNIX implementations, *getutxline()* and *getutxid()* treat the static area used for returning the *utmpx* structure as a kind of cache. If they determine that the record placed in this cache by a previous *getutx*()* call matches the criteria specified in *ut*, then no file read is performed; the call simply returns the same record once more (SUSv3 permits this behavior). Therefore, to prevent the same record from being repeatedly returned when calling *getutxline()* and *getutxid()* within a loop, we must zero out this static data structure, using code such as the following:

```
struct utmpx *res = NULL;

/* Other code omitted */

if (res != NULL)            /* If 'res' was set via a previous call */
    memset(res, 0, sizeof(struct utmpx));
res = getutxline(&ut);
```

The *glibc* implementation doesn't perform this type of caching, but we should nevertheless employ this technique for the sake of portability.

> Because the *getutx*()* functions return a pointer to a statically allocated structure, they are not reentrant. The GNU C library provides reentrant versions of the traditional *utmp* functions (*getutent_r()*, *getutid_r()*, and *getutline_r()*), but doesn't provide reentrant versions of their *utmpx* counterparts. (SUSv3 doesn't specify the reentrant versions.)

By default, all of the *getutx*()* functions work on the standard `utmp` file. If we want to use another file, such as the `wtmp` file, then we must first call *utmpxname()*, specifying the desired pathname.

```
#define _GNU_SOURCE
#include <utmpx.h>

int utmpxname(const char *file);
```

Returns 0 on success, or −1 on error

The *utmpxname()* function merely records a copy of the pathname given to it. It doesn't open the file, but does close any file previously opened by one of the other calls. This means that *utmpxname()* doesn't return an error if an invalid pathname is specified. Instead, when one of the *getutx*()* functions is later called, it will return an error (i.e., `NULL`, with *errno* set to `ENOENT`) when it fails to open the file.

> Although not specified in SUSv3, most UNIX implementations provide *utmpxname()* or the analogous *utmpname()* function.

Example program

The program in Listing 40-2 uses some of the functions described in this section to dump the contents of a *utmpx*-format file. The following shell session log demonstrates the results when we use this program to dump the contents of `/var/run/utmp` (the default used by these functions if *utmpxname()* is not called):

```
$ ./dump_utmpx
user     type         PID line   id  host      date/time
LOGIN    LOGIN_PR    1761 tty1   1             Sat Oct 23 09:29:37 2010
LOGIN    LOGIN_PR    1762 tty2   2             Sat Oct 23 09:29:37 2010
lynley   USER_PR    10482 tty3   3             Sat Oct 23 10:19:43 2010
david    USER_PR     9664 tty4   4             Sat Oct 23 10:07:50 2010
liz      USER_PR     1985 tty5   5             Sat Oct 23 10:50:12 2010
mtk      USER_PR    10111 pts/0  /0            Sat Oct 23 09:30:57 2010
```

For brevity, we edited out much of the output produced by the program. The lines matching `tty1` to `tty5` are for logins on virtual consoles (`/dev/tty[1-6]`). The last line of output is for an *xterm* session on a pseudoterminal.

The following output produced by dumping `/var/log/wtmp` shows that when a user logs in and out, two records are written to the `wtmp` file. (We edited out all of

the other output produced by the program.) By searching sequentially through the `wtmp` file (using *getutxline()*), these records can be matched via the *ut_line* field.

```
$ ./dump_utmpx /var/log/wtmp
user     type        PID line   id  host      date/time
lynley   USER_PR   10482 tty3   3             Sat Oct 23 10:19:43 2010
         DEAD_PR   10482 tty3   3   2.4.20-4G Sat Oct 23 10:32:54 2010
```

Listing 40-2: Displaying the contents of a *utmpx*-format file

loginacct/dump_utmpx.c

```
#define _GNU_SOURCE
#include <time.h>
#include <utmpx.h>
#include <paths.h>
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    struct utmpx *ut;

    if (argc > 1 && strcmp(argv[1], "--help") == 0)
        usageErr("%s [utmp-pathname]\n", argv[0]);

    if (argc > 1)               /* Use alternate file if supplied */
        if (utmpxname(argv[1]) == -1)
            errExit("utmpxname");

    setutxent();

    printf("user     type        PID line   id  host      date/time\n");

    while ((ut = getutxent()) != NULL) {        /* Sequential scan to EOF */
        printf("%-8s ", ut->ut_user);
        printf("%-9.9s ",
                (ut->ut_type == EMPTY) ?         "EMPTY" :
                (ut->ut_type == RUN_LVL) ?       "RUN_LVL" :
                (ut->ut_type == BOOT_TIME) ?     "BOOT_TIME" :
                (ut->ut_type == NEW_TIME) ?      "NEW_TIME" :
                (ut->ut_type == OLD_TIME) ?      "OLD_TIME" :
                (ut->ut_type == INIT_PROCESS) ?  "INIT_PR" :
                (ut->ut_type == LOGIN_PROCESS) ? "LOGIN_PR" :
                (ut->ut_type == USER_PROCESS) ?  "USER_PR" :
                (ut->ut_type == DEAD_PROCESS) ?  "DEAD_PR" : "???");
        printf("%5ld %-6.6s %-3.5s %-9.9s ", (long) ut->ut_pid,
                ut->ut_line, ut->ut_id, ut->ut_host);
        printf("%s", ctime((time_t *) &(ut->ut_tv.tv_sec)));
    }

    endutxent();
    exit(EXIT_SUCCESS);
}
```

loginacct/dump_utmpx.c

40.5 Retrieving the Login Name: *getlogin()*

The *getlogin()* function returns the name of the user logged in on the controlling terminal of the calling process. This function uses the information maintained in the utmp file.

```
#include <unistd.h>

char *getlogin(void);
```

Returns pointer to username string, or NULL on error

The *getlogin()* function calls *ttyname()* (Section 62.10) to find the name of the terminal associated with the calling process's standard input. It then searches the utmp file for a record whose *ut_line* value matches this terminal name. If a matching record is found, then *getlogin()* returns the *ut_user* string from that record.

If a match is not found or an error occurs, then *getlogin()* returns NULL and sets *errno* to indicate the error. One reason *getlogin()* may fail is that the process doesn't have a terminal associated with its standard input (ENOTTY), perhaps because it is daemon. Another possibility is that this terminal session is not recorded in utmp; for example, some software terminal emulators don't create entries in the utmp file.

Even in the (unusual) case where a user ID has multiple login names in /etc/passwd, *getlogin()* is able to return the actual username that was used to log in on this terminal because it relies on the utmp file. By contrast, using *getpwuid(getuid())* always retrieves the first matching record from /etc/passwd, regardless of the name that was used to log in.

> A reentrant version of *getlogin()* is specified by SUSv3, in the form of *getlogin_r()*, and this function is provided by *glibc*.
>
> The LOGNAME environment variable can also be used to find a user's login name. However, the value of this variable can be changed by the user, which means that it can't be used to securely identify a user.

40.6 Updating the utmp and wtmp Files for a Login Session

When writing an application that creates a login session (in the manner of, say, *login* or *sshd*), we should update the utmp and wtmp files as follows:

- On login, a record should be written to the utmp file to indicate that this user logged in. The application must check whether a record for this terminal already exists in the utmp file. If a previous record exists, it is overwritten; otherwise, a new record is appended to the file. Often, calling *pututxline()* (described shortly) is enough to ensure that these steps are correctly performed (see Listing 40-3 for an example). The output *utmpx* record should have at least the *ut_type*, *ut_user*, *ut_tv*, *ut_pid*, *ut_id*, and *ut_line* fields filled in. The *ut_type* field should be set to USER_PROCESS. The *ut_id* field should contain the suffix of the name of the device (i.e., the terminal or pseudoterminal) on which the user is logging in, and the *ut_line* field should contain the name of the login device, with the leading /dev/ string removed. (Examples of the contents of these two fields are shown in the

sample runs of the program in Listing 40-2.) A record containing exactly the same information is appended to the wtmp file.

> The terminal name acts (via the *ut_line* and *ut_id* fields) as a unique key for records in the utmp file.

- On logout, the record previously written to the utmp file should be erased. This is done by creating a record with *ut_type* set to DEAD_PROCESS, and with the same *ut_id* and *ut_line* values as the record written during login, but with the *ut_user* field zeroed out. This record is written over the earlier record. A copy of the same record is appended to the wtmp file.

  > If we fail to clean up the *utmp* record on logout, perhaps because of a program crash, then, on the next reboot, *init* automatically cleans up the record, setting its *ut_type* to DEAD_PROCESS and zeroing out various other fields of the record.

The utmp and wtmp files are normally protected so that only privileged users can perform updates on these files. The accuracy of *getlogin()* depends on the integrity of the utmp file. For this, as well as other reasons, the permissions on the utmp and wtmp files should never be set to allow writing by unprivileged users.

What qualifies as a login session? As we might expect, logins via *login*, *telnet*, and *ssh* are recorded in the login accounting files. Most *ftp* implementations also create login accounting records. However, are login accounting records created for each terminal window started on the system or for invocations of *su*, for example? The answer to that question varies across UNIX implementations.

> Under some terminal emulator programs (e.g., *xterm*), command-line options or other mechanisms can be used to determine whether the program updates the login accounting files.

The *pututxline()* function writes the *utmpx* structure pointed to by *ut* into the /var/run/utmp file (or an alternative file if *utmpxname()* was previously called).

```
#include <utmpx.h>

struct utmpx *pututxline(const struct utmpx *ut);
```

Returns pointer to copy of successfully updated record on success, or NULL on error

Before writing the record, *pututxline()* first uses *getutxid()* to search forward for a record that may be overwritten. If such a record is found, it is overwritten; otherwise, a new record is appended to the end of the file. In many cases, an application precedes a call to *pututxline()* by a call to one of the *getutx*()* functions, which sets the current file location to the correct record—that is, one matching the *getutxid()*-style criteria in the *utmpx* structure pointed to by *ut*. If *pututxline()* determines that this has occurred, it doesn't call *getutxid()*.

> If *pututxline()* makes an internal call to *getutxid()*, this call doesn't change the static area used by the *getutx*()* functions to return the *utmpx* structure. SUSv3 requires this behavior from an implementation.

When updating the wtmp file, we simply open the file and append a record to it. Because this is a standard operation, *glibc* encapsulates it in the *updwtmpx()* function.

```
#define _GNU_SOURCE
#include <utmpx.h>

void updwtmpx(char *wtmpx_file, struct utmpx *ut);
```

The *updwtmpx()* function appends the *utmpx* record pointed to by *ut* to the file specified in *wtmpx_file*.

SUSv3 doesn't specify *updwtmpx()*, and it appears on only a few other UNIX implementations. Other implementations provide related functions—*login(3)*, *logout(3)*, and *logwtmp(3)*—which are also in *glibc* and described in the manual pages. If such functions are not present, we need to write our own equivalents. (The implementation of these functions is not complex.)

Example program

Listing 40-3 uses the functions described in this section to update the utmp and wtmp files. This program performs the required updates to utmp and wtmp in order to log in the user named on the command line, and then, after sleeping a few seconds, log them out again. Normally, such actions would be associated with the creation and termination of a login session for a user. This program uses *ttyname()* to retrieve the name of the terminal device associated with a file descriptor. We describe *ttyname()* in Section 62.10.

The following shell session log demonstrates the operation of the program in Listing 40-3. We assume privilege in order to be able to update the login accounting files, and then use the program to create a record for the user *mtk*:

```
$ su
Password:
# ./utmpx_login mtk
Creating login entries in utmp and wtmp
        using pid 1471, line pts/7, id /7
Type Control-Z to suspend program
[1]+  Stopped                 ./utmpx_login mtk
```

While the *utmpx_login* program was sleeping, we typed *Control-Z* to suspend the program and push it into the background. Next, we use the program in Listing 40-2 to examine the contents of the utmp file:

```
# ./dump_utmpx /var/run/utmp
user     type        PID line   id  host      date/time
cecilia  USER_PR     249 tty1   1             Fri Feb  1 21:39:07 2008
mtk      USER_PR    1471 pts/7  /7            Fri Feb  1 22:08:06 2008
# who
cecilia  tty1     Feb  1 21:39
mtk      pts/7    Feb  1 22:08
```

Above, we used the *who(1)* command to show that the output of *who* derives from utmp.

Next we use our program to examine the contents of the wtmp file:

```
# ./dump_utmpx /var/log/wtmp
user     type        PID line   id  host      date/time
cecilia  USER_PR     249 tty1   1             Fri Feb  1 21:39:07 2008
mtk      USER_PR    1471 pts/7  /7            Fri Feb  1 22:08:06 2008
# last mtk
mtk      pts/7                     Fri Feb  1 22:08   still logged in
```

Above, we used the *last(1)* command to show that the output of *last* derives from wtmp. (For brevity, we have edited the output of the *dump_utmpx* and *last* commands in this shell session log to remove lines of output that are irrelevant to our discussion.)

Next, we use the *fg* command to resume the *utmpx_login* program in the foreground. It subsequently writes logout records to the utmp and wtmp files.

```
# fg
./utmpx_login mtk
Creating logout entries in utmp and wtmp
```

We then once more examine the contents of the utmp file. We see that the utmp record was overwritten:

```
# ./dump_utmpx /var/run/utmp
user     type        PID line   id  host      date/time
cecilia  USER_PR     249 tty1   1             Fri Feb  1 21:39:07 2008
         DEAD_PR    1471 pts/7  /7            Fri Feb  1 22:09:09 2008
# who
cecilia  tty1     Feb  1 21:39
```

The final line of output shows that *who* ignored the DEAD_PROCESS record.

When we examine the wtmp file, we see that the wtmp record was superseded:

```
# ./dump_utmpx /var/log/wtmp
user     type        PID line   id  host      date/time
cecilia  USER_PR     249 tty1   1             Fri Feb  1 21:39:07 2008
mtk      USER_PR    1471 pts/7  /7            Fri Feb  1 22:08:06 2008
         DEAD_PR    1471 pts/7  /7            Fri Feb  1 22:09:09 2008
# last mtk
mtk      pts/7                     Fri Feb  1 22:08 - 22:09  (00:01)
```

The final line of output above demonstrates that *last* matches the login and logout records in wtmp to show the starting and ending times of the completed login session.

Listing 40-3: Updating the utmp and wtmp files

loginacct/utmpx_login.c

```
#define _GNU_SOURCE
#include <time.h>
#include <utmpx.h>
#include <paths.h>              /* Definitions of _PATH_UTMP and _PATH_WTMP */
#include "tlpi_hdr.h"
```

```
int
main(int argc, char *argv[])
{
    struct utmpx ut;
    char *devName;

    if (argc < 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s username [sleep-time]\n", argv[0]);

    /* Initialize login record for utmp and wtmp files */

    memset(&ut, 0, sizeof(struct utmpx));
    ut.ut_type = USER_PROCESS;          /* This is a user login */
    strncpy(ut.ut_user, argv[1], sizeof(ut.ut_user));
    if (time((time_t *) &ut.ut_tv.tv_sec) == -1)
        errExit("time");                /* Stamp with current time */
    ut.ut_pid = getpid();

    /* Set ut_line and ut_id based on the terminal associated with
       'stdin'. This code assumes terminals named "/dev/[pt]t[sy]*".
       The "/dev/" dirname is 5 characters; the "[pt]t[sy]" filename
       prefix is 3 characters (making 8 characters in all). */

    devName = ttyname(STDIN_FILENO);
    if (devName == NULL)
        errExit("ttyname");
    if (strlen(devName) <= 8)           /* Should never happen */
        fatal("Terminal name is too short: %s", devName);

    strncpy(ut.ut_line, devName + 5, sizeof(ut.ut_line));
    strncpy(ut.ut_id, devName + 8, sizeof(ut.ut_id));

    printf("Creating login entries in utmp and wtmp\n");
    printf("        using pid %ld, line %.*s, id %.*s\n",
            (long) ut.ut_pid, (int) sizeof(ut.ut_line), ut.ut_line,
            (int) sizeof(ut.ut_id), ut.ut_id);

    setutxent();                        /* Rewind to start of utmp file */
    if (pututxline(&ut) == NULL)        /* Write login record to utmp */
        errExit("pututxline");
    updwtmpx(_PATH_WTMP, &ut);          /* Append login record to wtmp */

    /* Sleep a while, so we can examine utmp and wtmp files */

    sleep((argc > 2) ? getInt(argv[2], GN_NONNEG, "sleep-time") : 15);

    /* Now do a "logout"; use values from previously initialized 'ut',
       except for changes below */

    ut.ut_type = DEAD_PROCESS;          /* Required for logout record */
    time((time_t *) &ut.ut_tv.tv_sec);  /* Stamp with logout time */
    memset(&ut.ut_user, 0, sizeof(ut.ut_user));
                                        /* Logout record has null username */
```

```
    printf("Creating logout entries in utmp and wtmp\n");
    setutxent();                        /* Rewind to start of utmp file */
    if (pututxline(&ut) == NULL)        /* Overwrite previous utmp record */
        errExit("pututxline");
    updwtmpx(_PATH_WTMP, &ut);          /* Append logout record to wtmp */

    endutxent();
    exit(EXIT_SUCCESS);
}
```

loginacct/utmpx_login.c

40.7 The `lastlog` File

The `lastlog` file records the time each user last logged in to the system. (This is different from the `wtmp` file, which records all logins and logouts by all users.) Among other things, the `lastlog` file allows the *login* program to inform users (at the start of a new login session) when they last logged in. In addition to updating `utmp` and `wtmp`, applications providing login services should also update `lastlog`.

As with the `utmp` and `wtmp` files, there is variation in the location and format of the `lastlog` file. (A few UNIX implementations don't provide this file.) On Linux, this file resides at `/var/log/lastlog`, and a constant, `_PATH_LASTLOG`, is defined in `<paths.h>` to point to this location. Like the `utmp` and `wtmp` files, the `lastlog` file is normally protected so that it can be read by all users but can be updated only by privileged processes.

The records in the `lastlog` file have the following format (defined in `<lastlog.h>`):

```
#define UT_NAMESIZE           32
#define UT_HOSTSIZE          256

struct lastlog {
    time_t ll_time;                     /* Time of last login */
    char   ll_line[UT_NAMESIZE];        /* Terminal for remote login */
    char   ll_host[UT_HOSTSIZE];        /* Hostname for remote login */
};
```

Note that these records don't include a username or user ID. Instead, the `lastlog` file consists of a series of records that are indexed by user ID. Thus, to find the `lastlog` record for user ID 1000, we would seek to byte *(1000 * sizeof(struct lastlog))* of the file. This is demonstrated in Listing 40-4, a program that allows us to view the `lastlog` records for the user(s) listed on its command line. This is similar to the functionality offered by the *lastlog(1)* command. Here is an example of the output produced by running this program:

```
$ ./view_lastlog annie paulh
annie    tty2                     Mon Jan 17 11:00:12 2011
paulh    pts/11                   Sat Aug 14 09:22:14 2010
```

Performing updates on `lastlog` is similarly a matter of opening the file, seeking to the correct location, and performing a write.

Since the lastlog file is indexed by user ID, it is not possible to distinguish logins under different usernames that have the same user ID. (In Section 8.1, we noted that it is possible, though unusual, to have multiple login names with the same user ID.)

Listing 40-4: Displaying information from the lastlog file

loginacct/view_lastlog.c

```
#include <time.h>
#include <lastlog.h>
#include <paths.h>                      /* Definition of _PATH_LASTLOG */
#include <fcntl.h>
#include "ugid_functions.h"             /* Declaration of userIdFromName() */
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    struct lastlog llog;
    int fd, j;
    uid_t uid;

    if (argc > 1 && strcmp(argv[1], "--help") == 0)
        usageErr("%s [username...]\n", argv[0]);

    fd = open(_PATH_LASTLOG, O_RDONLY);
    if (fd == -1)
        errExit("open");

    for (j = 1; j < argc; j++) {
        uid = userIdFromName(argv[j]);
        if (uid == -1) {
            printf("No such user: %s\n", argv[j]);
            continue;
        }

        if (lseek(fd, uid * sizeof(struct lastlog), SEEK_SET) == -1)
            errExit("lseek");

        if (read(fd, &llog, sizeof(struct lastlog)) <= 0) {
            printf("read failed for %s\n", argv[j]);    /* EOF or error */
            continue;
        }

        printf("%-8.8s %-6.6s %-20.20s %s", argv[j], llog.ll_line,
                llog.ll_host, ctime((time_t *) &llog.ll_time));
    }

    close(fd);
    exit(EXIT_SUCCESS);
}
```

loginacct/view_lastlog.c

40.8 Summary

Login accounting records the users currently logged in, as well as all past logins. This information is maintained in three files: the `utmp` file, which maintains a record of all currently logged-in users; the `wtmp` file, which is an audit trail of all logins and logouts; and the `lastlog` file, which records the time of last login for each user. Various commands, such as *who* and *last*, use the information in these files.

The C library provides functions to retrieve and update the information in the login accounting files. Applications providing login services should use these functions to update the login accounting files, so that commands depending on this information operate correctly.

Further information

Aside from the *utmp(5)* manual page, the most useful place to find further information about the login accounting functions is in the source code of the various applications that use these functions. See, for example, the sources of *mingetty* (or *agetty*), *login*, *init*, *telnet*, *ssh*, and *ftp*.

40.9 Exercises

40-1. Implement *getlogin()*. As noted in Section 40.5, *getlogin()* may not work correctly for processes running under some software terminal emulators; in that case, test from a virtual console instead.

40-2. Modify the program in Listing 40-3 (`utmpx_login.c`) so that it updates the `lastlog` file in addition to the `utmp` and `wtmp` files.

40-3. Read the manual pages for *login(3)*, *logout(3)*, and *logwtmp(3)*. Implement these functions.

40-4. Implement a simple version of *who(1)*.