NAME

getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname, getutmp, getutmpx, updwtmp, updwtmpx – access utmpx file entry

SYNOPSIS

#include <utmpx.h>

```
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
struct utmpx *pututxline(const struct utmpx *utmpx);
void setutxent(void);
void endutxent(void);
int utmpxname(const char *file);
void getutmp(struct utmpx *utmpx, struct utmp *utmp);
void getutmpx(struct utmp *utmp, struct utmpx *utmpx);
void updwtmp(char *wfile, struct utmp *utmp);
void updwtmpx(char *wfilex, struct utmpx *utmpx);
```

MT-LEVEL

Unsafe

DESCRIPTION

getutxent( ), **getutxid( )**, and **getutxline( )** each return a pointer to a **utmpx** structure with the following members:

```
char    ut_user[32];           /* user login name */
char    ut_id[4];              /* /etc/inittab id */
                               /* (usually line #) */
char    ut_line[32];           /* device name (console, lnxx) */
pid_t   ut_pid;                /* process id */
short   ut_type;               /* type of entry */
struct  exit_status ut_exit;   /* exit status of a process */
                               /* marked as DEAD_PROCESS */
struct  timeval ut_tv;         /* time entry was made */
long    ut_session;            /* session ID, used for windowing */
long    pad[5];                /* reserved for future use */
short   ut_syslen;             /* significant length of ut_host */
                               /* including terminating null */
char    ut_host[257];          /* host name, if remote */
```

The structure exit status includes the following members:

```
short   e_termination;   /* termination status */
short   e_exit;          /* exit status */
```

getutxent( )

Reads in the next entry from a **utmpx**-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.

getutxid( )

Searches forward from the current point in the **utmpx** file until it finds an entry with a **ut_type** matching *id*→**ut_type** if the type specified is **RUN_LVL**, **BOOT_TIME**, **OLD_TIME**, or **NEW_TIME**. If the type specified in *id* is **INIT_PROCESS**, **LOGIN_PROCESS**,

USER_PROCESS, or **DEAD_PROCESS**, then **getutxid()** will return a pointer to the first entry whose type is one of these four and whose *ut_id* field matches *id*→**ut_id**. If the end of file is reached without a match, it fails.

getutxline() Searches forward from the current point in the **utmpx** file until it finds an entry of the type **LOGIN_PROCESS** or **USER_PROCESS** which also has a *ut_line* string matching the *line*→**ut_line** string. If the end of file is reached without a match, it fails.

pututxline() Writes out the supplied **utmpx** structure into the **utmpx** file. It uses **getutxid()** to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of **pututxline()** will have searched for the proper entry using one of the **getutx()** routines. If so, **pututxline()** will not search. If **pututxline()** does not find a matching slot for the new entry, it will add a new entry to the end of the file. It returns a pointer to the **utmpx** structure. When called by a non-root user, **pututxline()** invokes a **setuid()** root program to verify and write the entry, since **/etc/utmpx** is normally writable only by root. In this event, the *ut_name* field must correspond to the actual user name associated with the process; the *ut_type* field must be either **USER_PROCESS** or **DEAD_PROCESS**; and the *ut_line* field must be a device special file and be writable by the user.

setutxent() Resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.

endutxent() Closes the currently open file.

utmpxname() Allows the user to change the name of the file examined, from **/var/adm/utmpx** to any other file. It is most often expected that this other file will be **/var/adm/wtmpx**. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. **utmpxname()** does not open the file. It just closes the old file if it is currently open and saves the new file name. The new file name must end with the ''**x**'' character to allow the name of the corresponding **utmp** file to be easily obtainable; otherwise, an error code of **1** is returned.

getutmp() Copies the information stored in the fields of the **utmpx** structure to the corresponding fields of the **utmp** structure. If the information in any field of **utmpx** does not fit in the corresponding **utmp** field, the data is truncated. (See **getutent**(3C) for **utmp** structure)

getutmpx() Copies the information stored in the fields of the **utmp** structure to the corresponding fields of the **utmpx** structure. (See **getutent**(3C) for **utmp** structure)

updwtmp() Checks the existence of *wfile* and its parallel file, whose name is obtained by appending an ''**x**'' to *wfile*. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. *utmp* is written to *wfile* and the corresponding **utmpx** structure is written to the parallel file.

updwtmpx( ) Checks the existence of *wfilex* and its parallel file, whose name is obtained by truncating the final ''**x**'' from *wfilex*. If only one of them exists, the second one is created and initialized to reflect the state of the existing file. *utmpx* is written to *wfilex*, and the corresponding **utmp** structure is written to the parallel file.

RETURN VALUES

A null pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

FILES

/var/adm/utmp	contains current user access and adminstrative information (old format)
/var/adm/utmpx	contains current user access and adminstration information (new format)
/var/adm/wtmp	contains a history of user access and adminstrative information.
/var/adm/wtmpx	contains a history of user access and adminstrative information.

SEE ALSO

getutent(3C), **ttyslot**(3C), **utmp**(4), **utmpx**(4)

NOTES

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. On each call to either **getutxid( )** or **getutxline( )**, the routine examines the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use **getutxline( )** to search for multiple occurrences it would be necessary to zero out the static after each success, or **getutxline( )** would just return the same structure over and over again. There is one exception to the rule about emptying the structure before further reads are done. The implicit read done by **pututxline( )** (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the **getutxent( )**, **getutxid( )**, or **getutxline( )** routines, if the user has just modified those contents and passed the pointer back to **pututxline( )**.

These routines use buffered standard I/O for input, but **pututxline( )** uses an unbuffered write to avoid race conditions between processes trying to modify the **utmpx** and **wtmpx** files.