****
1773 Sun Nov 30 17:57:26 2014
new/usr/src/cmd/acct/utmp2wtmp.c
3124 Remove any existing references to utmp, use utmpx instead
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"
/*
 *      create entries for users who are still logged on when accounting
 *      is being run. Look at utmpx, and update the time stamp. New info
 *      goes to wtmpx. Called by runacct.
 */

#include <stdio.h>
#include <sys/types.h>
#include <utmpx.h>
#include <time.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
        struct utmpx *utmpx;
        FILE *fp;

        fp = fopen(WTMPX_FILE, "a+");
        if (fp == NULL) {
                fprintf(stderr, "%s: %s: %s\n", argv[0],
                    WTMPX_FILE, strerror(errno));
                exit(1);
        }

        while ((utmpx = getutxent()) != NULL) {
                if ((utmpx->ut_type == USER_PROCESS) && !(nonuserx(*utmpx))) {
                if ((utmpx->ut_type == USER_PROCESS) && !(nonuser(*utmpx))) {
                        time(&utmpx->ut_xtime);
```



```
                        fwrite(utmpx, sizeof (*utmpx), 1, fp);
                        fwrite(utmpx, sizeof(*utmpx), 1, fp);
                }
        }
        fclose(fp);
        exit(0);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   25245 Sun Nov 30 17:57:26 2014
new/usr/src/cmd/utmpd/utmpd.c
3124 Remove any existing references to utmp, use utmpx instead
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  */

  26 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
  27 /*        All Rights Reserved   */

  29 /*
  30  * Portions of such source code were derived from Berkeley 4.3 BSD
  31  * under license from the Regents of the University of California.
  32  */

  34 #pragma ident   "%Z%%M% %I%     %E% SMI"

  34 /*
  35  * utmpd        - utmp daemon
  36  *
  37  *              This program receives requests from  pututxline(3)
  38  *              via a named pipe to watch the process to make sure it cleans up
  39  *              its utmpx entry on termination.
  40  *              The program keeps a list of procs
  41  *              and uses poll() on their /proc files to detect termination.
  42  *              Also the  program periodically scans the /etc/utmpx file for
  43  *              processes that aren't in the table so they can be watched.
  44  *
  45  *              If utmpd doesn't hear back over the pipe from pututline(3) that
  46  *              the process has removed its entry it cleans the entry when the
  47  *              the process terminates.
  48  *              The AT&T Copyright above is there since we borrowed the pipe
  49  *              mechanism from init(1m).
  50  */


  53 #include        <sys/types.h>
  54 #include        <signal.h>
  55 #include        <stdio.h>
  56 #include        <stdio_ext.h>
  57 #include        <unistd.h>
  58 #include        <utmpx.h>
  59 #include        <errno.h>
```



```
  60 #include        <termio.h>
  61 #include        <sys/termios.h>
  62 #include        <sys/tty.h>
  63 #include        <ctype.h>
  64 #include        <sys/stat.h>
  65 #include        <sys/statvfs.h>
  66 #include        <fcntl.h>
  67 #include        <time.h>
  68 #include        <sys/stropts.h>
  69 #include        <wait.h>
  70 #include        <syslog.h>
  71 #include        <stdlib.h>
  72 #include        <string.h>
  73 #include        <poll.h>
  74 #include        <deflt.h>
  75 #include        <procfs.h>
  76 #include        <sys/resource.h>

  78 #define dprintf(x)      if (Debug) (void) printf x

  80 /*
  81  * Memory allocation keyed off MAX_FDS
  82  */
  83 #define MAX_FDS         4064    /* Maximum # file descriptors */
  84 #define EXTRA_MARGIN    32      /* Allocate this many more FDS over Max_Fds */
  85 /*
  86  * MAX_POLLNV & RESETS - paranoia to cover an error case that might not exist
  87  */
  88 #define MAX_POLL_ERRS   1024    /* Count of bad errors */
  89 #define MAX_RESETS      1024    /* Maximum times to reload tables */
  90 #define POLL_TIMEOUT    300     /* Default Timeout for poll() in seconds */
  91 #define CLEANIT         1       /* Used by rem_pid() */
  92 #define DONT_CLEAN      0       /* Used by rem_pid() */
  93 #define UTMP_DEFAULT    "/etc/default/utmpd"
  94 #define WARN_TIME       3600    /* seconds between utmp checks */
  95 #define WTMPX_UFREQ     60      /* seconds between updating WTMPX's atime */


  98 /*
  99  * The pidrec structure describes the data shipped down the pipe to
 100  * us from the pututxline() library in
 101  * lib/libc/port/gen/getutx.c
 102  */

 104 /*
 105  * pd_type's
 106  */
 107 #define ADDPID  1
 108 #define REMPID  2

 110 struct  pidrec {
 111         int     pd_type;                /* Command type */
 112         pid_t   pd_pid;                 /* pid to add or remove */
 113 };
_____unchanged_portion_omitted_


 946 /*
 947  * clean_utmpx_ent      - Clean a utmpx entry
 948  */

 950 static void
 951 clean_utmpx_ent(u)
 952         struct utmpx *u;
 953 {
 954         dprintf(("      clean_utmpx_ent: %d\n", (int)u->ut_pid));
```

```
 955         u->ut_type = DEAD_PROCESS;
 956         (void) time(&u->ut_xtime);
 957         (void) pututxline(u);
 958         updwtmpx(WTMPX_FILE, u);
 959         /*
 960          * XXX update wtmp for ! nonuserx entries?
 962          * XXX update wtmp for ! nonuser entries?
 961          */
 962 }
```

unchanged_portion_omitted_

```
**********************************************************
   18949 Sun Nov 30 17:57:26 2014
new/usr/src/cmd/w/w.c
3124 Remove any existing references to utmp, use utmpx instead
**********************************************************
______unchanged_portion_omitted_

 117 /*
 118  *      define hash table for struct uproc
 119  *      Hash function uses process id
 120  *      and the size of the hash table(HSIZE)
 121  *      to determine process index into the table.
 122  */
 123 static struct uproc     pr_htbl[HSIZE];

 125 static struct   uproc   *findhash(pid_t);
 126 static time_t   findidle(char *);
 127 static void     clnarglist(char *);
 128 static void     showtotals(struct uproc *);
 129 static void     calctotals(struct uproc *);
 130 static void     prttime(time_t, int);
 131 static void     prtat(time_t *time);

 133 static char     *prog;          /* pointer to invocation name */
 134 static int      header = 1;     /* true if -h flag: don't print heading */
 135 static int      lflag = 1;      /* set if -l flag; 0 for -s flag: short form */
 136 static char     *sel_user;      /* login of particular user selected */
 137 static char     firstchar;      /* first char of name of prog invoked as */
 138 static int      login;          /* true if invoked as login shell */
 139 static time_t   now;            /* current time of day */
 140 static time_t   uptime;         /* time of last reboot & elapsed time since */
 141 static int      nusers;         /* number of users logged in now */
 142 static time_t   idle;           /* number of minutes user is idle */
 143 static time_t   jobtime;        /* total cpu time visible */
 144 static char     doing[520];     /* process attached to terminal */
 145 static time_t   proctime;       /* cpu time of process in doing */
 146 static pid_t    curpid, empty;
 147 static int      add_times;      /* boolean: add the cpu times or not */

 149 #if SIGQUIT > SIGINT
 150 #define ACTSIZE SIGQUIT
 151 #else
 152 #define ACTSIZE SIGINT
 153 #endif

 155 int
 156 main(int argc, char *argv[])
 157 {
 158         struct utmpx    *ut;
 159         struct utmpx    *utmpbegin;
 160         struct utmpx    *utmpend;
 161         struct utmpx    *utp;
 162         struct uproc    *up, *parent, *pgrp;
 163         struct psinfo   info;
 164         struct sigaction actinfo[ACTSIZE];
 165         struct pstatus  statinfo;
 166         size_t          size;
 167         struct stat     sbuf;
 168         DIR             *dirp;
 169         struct  dirent  *dp;
 170         char            pname[64];
 171         char            *fname;
 172         int             procfd;
 173         char            *cp;
 174         int             i;
 175         int             days, hrs, mins;
```



```
 176         int             entries;
 177         double          loadavg[3];

 179         /*
 180          * This program needs the proc_owner privilege
 181          */
 182         (void) __init_suid_priv(PU_CLEARLIMITSET, PRIV_PROC_OWNER,
 183             (char *)NULL);

 185         (void) setlocale(LC_ALL, "");
 186 #if !defined(TEXT_DOMAIN)
 187 #define TEXT_DOMAIN "SYS_TEST"
 188 #endif
 189         (void) textdomain(TEXT_DOMAIN);

 191         login = (argv[0][0] == '-');
 192         cp = strrchr(argv[0], '/');
 193         firstchar = login ? argv[0][1] : (cp == 0) ? argv[0][0] : cp[1];
 194         prog = argv[0];

 196         while (argc > 1) {
 197                 if (argv[1][0] == '-') {
 198                         for (i = 1; argv[1][i]; i++) {
 199                                 switch (argv[1][i]) {

 201                                 case 'h':
 202                                         header = 0;
 203                                         break;

 205                                 case 'l':
 206                                         lflag++;
 207                                         break;
 208                                 case 's':
 209                                         lflag = 0;
 210                                         break;

 212                                 case 'u':
 213                                 case 'w':
 214                                         firstchar = argv[1][i];
 215                                         break;

 217                                 default:
 218                                         (void) fprintf(stderr, gettext(
 219                                             "%s: bad flag %s\n"),
 220                                             prog, argv[1]);
 221                                         exit(1);
 222                                 }
 223                         }
 224                 } else {
 225                         if (!isalnum(argv[1][0]) || argc > 2) {
 226                                 (void) fprintf(stderr, gettext(
 227                                     "usage: %s [ -hlsuw ] [ user ]\n"), prog);
 228                                 exit(1);
 229                         } else
 230                                 sel_user = argv[1];
 231                 }
 232                 argc--; argv++;
 233         }

 235         /*
 236          * read the UTMPX_FILE (contains information about each logged in user)
 236          * read the UTMP_FILE (contains information about each logged in user)
 237          */
 238         if (stat(UTMPX_FILE, &sbuf) == ERR) {
 239                 (void) fprintf(stderr, gettext("%s: stat error of %s: %s\n"),
 240                     prog, UTMPX_FILE, strerror(errno));
```

```
241                 exit(1);
242         }
243         entries = sbuf.st_size / sizeof (struct futmpx);
244         size = sizeof (struct utmpx) * entries;
245         if ((ut = malloc(size)) == NULL) {
246                 (void) fprintf(stderr, gettext("%s: malloc error of %s: %s\n"),
247                     prog, UTMPX_FILE, strerror(errno));
248                 exit(1);
249         }

251         (void) utmpxname(UTMPX_FILE);

253         utmpbegin = ut;
254         utmpend = (struct utmpx *)((char *)utmpbegin + size);

256         setutxent();
257         while ((ut < utmpend) && ((utp = getutxent()) != NULL))
258                 (void) memcpy(ut++, utp, sizeof (*ut));
259         endutxent();

261         (void) time(&now);      /* get current time */

263         if (header) {   /* print a header */
264                 prtat(&now);
265                 for (ut = utmpbegin; ut < utmpend; ut++) {
266                         if (ut->ut_type == USER_PROCESS) {
267                                 if (!nonuserx(*ut))
267                                 if (!nonuser(*ut))
268                                         nusers++;
269                         } else if (ut->ut_type == BOOT_TIME) {
270                                 uptime = now - ut->ut_xtime;
271                                 uptime += 30;
272                                 days = uptime / (60*60*24);
273                                 uptime %= (60*60*24);
274                                 hrs = uptime / (60*60);
275                                 uptime %= (60*60);
276                                 mins = uptime / 60;

278                                 PRINTF((gettext("up")));
279                                 if (days > 0)
280                                         PRINTF((gettext(
281                                             " %d day(s),"), days));
282                                 if (hrs > 0 && mins > 0) {
283                                         PRINTF((" %2d:%02d,", hrs, mins));
284                                 } else {
285                                         if (hrs > 0)
286                                                 PRINTF((gettext(
287                                                     " %d hr(s),"), hrs));
288                                         if (mins > 0)
289                                                 PRINTF((gettext(
290                                                     " %d min(s),"), mins));
291                                 }
292                         }
293                 }

295                 ut = utmpbegin; /* rewind utmp data */
296                 PRINTF((((nusers == 1) ?
297                     gettext("  %d user") : gettext("  %d users")), nusers));
298                 /*
299                  * Print 1, 5, and 15 minute load averages.
300                  */
301                 (void) getloadavg(loadavg, 3);
302                 PRINTF((gettext(",  load average: %.2f, %.2f, %.2f\n"),
303                     loadavg[LOADAVG_1MIN], loadavg[LOADAVG_5MIN],
304                     loadavg[LOADAVG_15MIN]));
```



```
306                 if (firstchar == 'u')   /* uptime command */
307                         exit(0);

309                 if (lflag) {
310                         PRINTF((dcgettext(NULL, "User     tty      "
311                             "login@         idle    JCPU    PCPU what\n",
312                             LC_TIME)));
313                 } else {
314                         PRINTF((dcgettext(NULL,
315                             "User     tty         idle what\n",
316                             LC_TIME)));
317                 }

319                 if (fflush(stdout) == EOF) {
320                         perror((gettext("%s: fflush failed\n"), prog));
321                         exit(1);
322                 }
323         }

325         /*
326          * loop through /proc, reading info about each process
327          * and build the parent/child tree
328          */
329         if (!(dirp = opendir(PROCDIR))) {
330                 (void) fprintf(stderr, gettext("%s: could not open %s: %s\n"),
331                     prog, PROCDIR, strerror(errno));
332                 exit(1);
333         }

335         while ((dp = readdir(dirp)) != NULL) {
336                 if (dp->d_name[0] == '.')
337                         continue;
338 retry:
339                 (void) sprintf(pname, "%s/%s/", PROCDIR, dp->d_name);
340                 fname = pname + strlen(pname);
341                 (void) strcpy(fname, "psinfo");
342                 if ((procfd = open(pname, O_RDONLY)) < 0)
343                         continue;
344                 if (read(procfd, &info, sizeof (info)) != sizeof (info)) {
345                         int err = errno;
346                         (void) close(procfd);
347                         if (err == EAGAIN)
348                                 goto retry;
349                         if (err != ENOENT)
350                                 (void) fprintf(stderr, gettext(
351                                     "%s: read() failed on %s: %s \n"),
352                                     prog, pname, strerror(err));
353                         continue;
354                 }
355                 (void) close(procfd);

357                 up = findhash(info.pr_pid);
358                 up->p_ttyd = info.pr_ttydev;
359                 up->p_state = (info.pr_nlwp == 0? ZOMBIE : RUNNING);
360                 up->p_time = 0;
361                 up->p_ctime = 0;
362                 up->p_igintr = 0;
363                 (void) strncpy(up->p_comm, info.pr_fname,
364                     sizeof (info.pr_fname));
365                 up->p_args[0] = 0;

367                 if (up->p_state != NONE && up->p_state != ZOMBIE) {
368                         (void) strcpy(fname, "status");

370                         /* now we need the proc_owner privilege */
371                         (void) __priv_bracket(PRIV_ON);
```

```
373                     procfd = open(pname, O_RDONLY);

375                     /* drop proc_owner privilege after open */
376                     (void) __priv_bracket(PRIV_OFF);

378                     if (procfd < 0)
379                             continue;

381                     if (read(procfd, &statinfo, sizeof (statinfo))
382                         != sizeof (statinfo)) {
383                             int err = errno;
384                             (void) close(procfd);
385                             if (err == EAGAIN)
386                                     goto retry;
387                             if (err != ENOENT)
388                                     (void) fprintf(stderr, gettext(
389                                         "%s: read() failed on %s: %s \n"),
390                                         prog, pname, strerror(err));
391                             continue;
392                     }
393                     (void) close(procfd);

395                     up->p_time = statinfo.pr_utime.tv_sec +
396                         statinfo.pr_stime.tv_sec;   /* seconds */
397                     up->p_ctime = statinfo.pr_cutime.tv_sec +
398                         statinfo.pr_cstime.tv_sec;

400                     (void) strcpy(fname, "sigact");

402                     /* now we need the proc_owner privilege */
403                     (void) __priv_bracket(PRIV_ON);

405                     procfd = open(pname, O_RDONLY);

407                     /* drop proc_owner privilege after open */
408                     (void) __priv_bracket(PRIV_OFF);

410                     if (procfd < 0)
411                             continue;

413                     if (read(procfd, actinfo, sizeof (actinfo))
414                         != sizeof (actinfo)) {
415                             int err = errno;
416                             (void) close(procfd);
417                             if (err == EAGAIN)
418                                     goto retry;
419                             if (err != ENOENT)
420                                     (void) fprintf(stderr, gettext(
421                                         "%s: read() failed on %s: %s \n"),
422                                         prog, pname, strerror(err));
423                             continue;
424                     }
425                     (void) close(procfd);

427                     up->p_igintr =
428                         actinfo[SIGINT-1].sa_handler == SIG_IGN &&
429                         actinfo[SIGQUIT-1].sa_handler == SIG_IGN;

431                     /*
432                      * Process args.
433                      */
434                     up->p_args[0] = 0;
435                     clnarglist(info.pr_psargs);
436                     (void) strcat(up->p_args, info.pr_psargs);
437                     if (up->p_args[0] == 0 ||
```



```
438                         up->p_args[0] == '-' && up->p_args[1] <= ' ' ||
439                         up->p_args[0] == '?') {
440                             (void) strcat(up->p_args, " (");
441                             (void) strcat(up->p_args, up->p_comm);
442                             (void) strcat(up->p_args, ")");
443                     }
444             }

446             /*
447              * link pgrp together in case parents go away
448              * Pgrp chain is a single linked list originating
449              * from the pgrp leader to its group member.
450              */
451             if (info.pr_pgid != info.pr_pid) {      /* not pgrp leader */
452                     pgrp = findhash(info.pr_pgid);
453                     up->p_pgrpl = pgrp->p_pgrpl;
454                     pgrp->p_pgrpl = up;
455             }
456             parent = findhash(info.pr_ppid);

458             /* if this is the new member, link it in */
459             if (parent->p_upid != INITPROCESS) {
460                     if (parent->p_child) {
461                             up->p_sibling = parent->p_child;
462                             up->p_child = 0;
463                     }
464                     parent->p_child = up;
465             }
466     }

468     /* revert to non-privileged user after opening */
469     (void) __priv_relinquish();

471     (void) closedir(dirp);
472     (void) time(&now);      /* get current time */

474     /*
475      * loop through utmpx file, printing process info
476      * about each logged in user
477      */
478     for (ut = utmpbegin; ut < utmpend; ut++) {
479             if (ut->ut_type != USER_PROCESS)
480                     continue;
481             if (sel_user && strncmp(ut->ut_name, sel_user, NMAX) != 0)
482                     continue;       /* we're looking for somebody else */

484             /* print login name of the user */
485             PRINTF(("%-*.*s ", LOGIN_WIDTH, NMAX, ut->ut_name));

487             /* print tty user is on */
488             if (lflag) {
489                     PRINTF(("%-*.*s ", LINE_WIDTH, LMAX, ut->ut_line));
490             } else {
491                     if (ut->ut_line[0] == 'p' && ut->ut_line[1] == 't' &&
492                         ut->ut_line[2] == 's' && ut->ut_line[3] == '/') {
493                             PRINTF(("%-*.*s ", LINE_WIDTH, LMAX,
494                                 &ut->ut_line[4]));
495                     } else {
496                             PRINTF(("%-*.*s ", LINE_WIDTH, LMAX,
497                                 ut->ut_line));
498                     }
499             }

501             /* print when the user logged in */
502             if (lflag) {
503                     time_t tim = ut->ut_xtime;
```

```
504                         prtat(&tim);
505                 }

507                 /* print idle time */
508                 idle = findidle(ut->ut_line);
509                 prttime(idle, 8);
510                 showtotals(findhash(ut->ut_pid));
511         }
512         if (fclose(stdout) == EOF) {
513                 perror((gettext("%s: fclose failed"), prog));
514                 exit(1);
515         }
516         return (0);
517 }
```

_____***unchanged_portion_omitted_***

****
11097 Sun Nov 30 17:57:26 2014
new/usr/src/cmd/wall/wall.c
3124 Remove any existing references to utmp, use utmpx instead
****
_____***unchanged_portion_omitted_***

```
 292 static void
 293 sendmes_tozone(zoneid_t zid, int aflag) {
 294         int i = 0;
 295         char zonename[ZONENAME_MAX], root[MAXPATHLEN];
 296         struct utmpx *p;

 298         if (zid != getzoneid()) {
 299                 root[0] = '\0';
 300                 (void) getzonenamebyid(zid, zonename, ZONENAME_MAX);
 301                 (void) zone_get_rootpath(zonename, root, sizeof (root));
 302                 (void) strlcat(root, UTMPX_FILE, sizeof (root));
 303                 if (!utmpxname(root)) {
 304                         (void) fprintf(stderr, "Cannot open %s\n", root);
 305                         return;
 306                 }
 307         } else {
 308                 (void) utmpxname(UTMPX_FILE);
 309         }
 310         setutxent();
 311         while ((p = getutxent()) != NULL) {
 312                 if (p->ut_type != USER_PROCESS)
 313                         continue;
 314                 /*
 315                  * if (-a option OR NOT pty window login), send the message
 316                  */
 317                 if (aflag || !nonuserx(*p))
 317                 if (aflag || !nonuser(*p))
 318                         sendmes(p, zid);
 319         }
 320         endutxent();

 322         (void) alarm(60);
 323         do {
 324                 i = (int)wait((int *)0);
 325         } while (i != -1 || errno != ECHILD);

 327 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   12634 Sun Nov 30 17:57:26 2014
new/usr/src/man/man3c/getutxent.3c
3124 Remove any existing references to utmp, use utmpx instead
**********************************************************
   1 '\" te
   2 .\" Copyright 1989 AT&T.  Copyright (c) 2004 Sun Microsystems, Inc.  All Rights
   3 .\" The contents of this file are subject to the terms of the Common Development
   4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
   5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
   6 .TH GETUTXENT 3C "Nov 21, 2014"
   6 .TH GETUTXENT 3C "Jul 27, 2004"
   7 .SH NAME
   8 getutxent, getutxid, getutxline, pututxline, setutxent, endutxent, utmpxname,
   9 getutmp, getutmpx, updwtmp, updwtmpx \- user accounting database functions
  10 .SH SYNOPSIS
  11 .LP
  12 .nf
  13 #include <utmpx.h>

  15 \fBstruct utmpx *\fR\fBgetutxent\fR(\fBvoid\fR);
  16 .fi

  18 .LP
  19 .nf
  20 \fBstruct utmpx *\fR\fBgetutxid\fR(\fBconst struct utmpx *\fR\fIid\fR);
  21 .fi

  23 .LP
  24 .nf
  25 \fBstruct utmpx *\fR\fBgetutxline\fR(\fBconst struct utmpx *\fR\fIline\fR);
  26 .fi

  28 .LP
  29 .nf
  30 \fBstruct utmpx *\fR\fBpututxline\fR(\fBconst struct utmpx *\fR\fIutmpx\fR);
  31 .fi

  33 .LP
  34 .nf
  35 \fBvoid\fR \fBsetutxent\fR(\fBvoid\fR);
  36 .fi

  38 .LP
  39 .nf
  40 \fBvoid\fR \fBendutxent\fR(\fBvoid\fR);
  41 .fi

  43 .LP
  44 .nf
  45 \fBint\fR \fButmpxname\fR(\fBconst char *\fR\fIfile\fR);
  46 .fi

  48 .LP
  49 .nf
  50 \fBvoid\fR \fBgetutmp\fR(\fBstruct utmpx *\fR\fIutmpx\fR, \fBstruct utmp *\fR\fI
  51 .fi

  53 .LP
  54 .nf
  55 \fBvoid\fR \fBgetutmpx\fR(\fBstruct utmp *\fR\fIutmp\fR, \fBstruct utmpx *\fR\fI
  56 .fi

  58 .LP
  59 .nf
  60 \fBvoid\fR \fBupdwtmp\fR(\fBchar *\fR\fIwfile\fR, \fBstruct utmp *\fR\fIutmp\fR)
```



```
  61 .fi

  63 .LP
  64 .nf
  65 \fBvoid\fR \fBupdwtmpx\fR(\fBchar *\fR\fIwfilex\fR, \fBstruct utmpx *\fR\fIutmpx
  66 .fi

  68 .SH DESCRIPTION
  69 .sp
  70 .LP
  71 These functions provide access to the user accounting database, \fButmpx\fR
  72 (see \fButmpx\fR(4)). Entries in the database are described by the definitions
  73 and data structures in \fB<utmpx.h>\fR\&.
  74 .sp
  75 .LP
  76 The \fButmpx\fR structure contains the following members:
  77 .sp
  78 .in +2
  79 .nf
  80 char                ut_user[32];   /* user login name */
  81 char                ut_id[4];      /* /etc/inittab id */
  82                                    /* (usually line #) */
  83 char                ut_line[32];   /* device name */
  84                                    /* (console, lnxx) */
  85 pid_t               ut_pid;        /* process id */
  86 short               ut_type;       /* type of entry */
  87 struct exit_status  ut_exit;       /* exit status of a process */
  88                                    /* marked as DEAD_PROCESS */
  89 struct timeval      ut_tv;         /* time entry was made */
  90 int                 ut_session;    /* session ID, used for */
  91                                    /* windowing */
  92 short               ut_syslen;     /* significant length of */
  93                                    /* ut_host */
  94                                    /* including terminating null */
  95 char                ut_host[257];  /* host name, if remote */
  96 .fi
  97 .in -2

  99 .sp
 100 .LP
 101 The \fBexit_status\fR structure includes the following members:
 102 .sp
 103 .in +2
 104 .nf
 105 short   e_termination;   /* termination status */
 106 short   e_exit;          /* exit status */
 107 .fi
 108 .in -2

 110 .SS "\fBgetutxent()\fR"
 111 .sp
 112 .LP
 113 The \fBgetutxent()\fR function reads in the next entry from a \fButmpx\fR
 114 database. If the database is not already open, it opens it. If it reaches the
 115 end of the database, it fails.
 116 .SS "\fBgetutxid()\fR"
 117 .sp
 118 .LP
 119 The \fBgetutxid()\fR function searches forward from the current point in the
 120 \fButmpx\fR database until it finds an entry with a \fBut_type\fR matching
 121 \fIid\fR->\fBut_type\fR, if the type specified is \fBRUN_LVL\fR,
 122 \fBBOOT_TIME\fR, \fBDOWN_TIME\fR, \fBOLD_TIME\fR, or \fBNEW_TIME\fR. If the
 123 type specified in \fIid\fR is \fBINIT_PROCESS\fR, \fBLOGIN_PROCESS\fR,
 124 \fBUSER_PROCESS\fR, or \fBDEAD_PROCESS\fR, then \fBgetutxid()\fR will return a
 125 pointer to the first entry whose type is one of these four and whose
 126 \fBut_id\fR member matches \fIid\fR->\fBut_id\fR. If the end of database is
```

reached without a match, it fails.
.SS "\fBgetutxline()\fR"
.sp
.LP
The \fBgetutxline()\fR function searches forward from the current point in the \fButmpx\fR database until it finds an entry of the type \fBLOGIN_PROCESS\fR or \fBUSER_PROCESS\fR which also has a \fIut_line\fR string matching the \fIline\fR->\fBut_line\fR string. If the end of the database is reached without a match, it fails.
.SS "\fBpututxline()\fR"
.sp
.LP
The \fBpututxline()\fR function writes the supplied \fButmpx\fR structure into the \fButmpx\fR database. It uses \fBgetutxid()\fR 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 \fBpututxline()\fR will have searched for the proper entry using one of the \fBgetutx()\fR routines. If so, \fBpututxline()\fR will not search. If \fBpututxline()\fR does not find a matching slot for the new entry, it will add a new entry to the end of the database. It returns a pointer to the \fButmpx\fR structure. When called by a non-root user, \fBpututxline()\fR invokes a \fBsetuid()\fR root program to verify and write the entry, since the \fButmpx\fR database is normally writable only by root. In this event, the \fBut_name\fR member must correspond to the actual user name associated with the process; the \fBut_type\fR member must be either \fBUSER_PROCESS\fR or \fBDEAD_PROCESS\fR; and the \fBut_line\fR member must be a device special file and be writable by the user.
.SS "\fBsetutxent()\fR"
.sp
.LP
The \fBsetutxent()\fR function resets the input stream to the beginning. This should be done before each search for a new entry if it is desired that the entire database be examined.
.SS "\fBendutxent()\fR"
.sp
.LP
The \fBendutxent()\fR function closes the currently open database.
.SS "\fButmpxname()\fR"
.sp
.LP
The \fButmpxname()\fR function allows the user to change the name of the database file examined from \fB/var/adm/utmpx\fR to any other file, most often \fB/var/adm/wtmpx\fR. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. The \fButmpxname()\fR function does not open the file, but 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 \fButmp\fR file to be easily obtainable.; otherwise, an error value of \fB0\fR is returned. The function returns \fB1\fR on success.
.SS "\fBgetutmp()\fR"
.sp
.LP
The \fBgetutmp()\fR function copies the information stored in the members of the \fButmpx\fR structure to the corresponding members of the \fButmp\fR structure. If the information in any member of \fButmpx\fR does not fit in the corresponding \fButmp\fR member, the data is silently truncated. (See \fBgetutent\fR(3C) for \fButmp\fR structure)
.SS "\fBgetutmpx()\fR"
.sp
.LP
The \fBgetutmpx()\fR function copies the information stored in the members of the \fButmp\fR structure to the corresponding members of the \fButmpx\fR structure. (See \fBgetutent\fR(3C) for \fButmp\fR structure)
.SS "\fBupdwtmp()\fR"
.sp
.LP
The \fBupdwtmp()\fR function can be used in two ways.

.sp
.LP
If \fIwfile\fR is \fB/var/adm/wtmp\fR, the \fButmp\fR format record supplied by the caller is converted to a \fButmpx \fRformat record and the \fB/var/adm/wtmpx\fR file is updated (because the \fB/var/adm/wtmp\fR file no longer exists, operations on \fBwtmp\fR are converted to operations on \fBwtmpx\fR by the library functions.
.sp
.LP
If \fIwfile\fR is a file other than \fB/var/adm/wtmp\fR, it is assumed to be an old file in \fButmp\fR format and is updated directly with the \fButmp\fR format record supplied by the caller.
.SS "\fBupdwtmpx()\fR"
.sp
.LP
The \fBupdwtmpx()\fR function writes the contents of the \fButmpx\fR structure pointed to by \fIutmpx\fR to the database.
.SS "\fButmpx\fR structure"
.sp
.LP
The values of the \fBe_termination\fR and \fBe_exit\fR members of the \fBut_exit\fR structure are valid only for records of type \fBDEAD_PROCESS\fR. For \fButmpx\fR entries created by \fBinit\fR(1M), these values are set according to the result of the \fBwait()\fR call that \fBinit\fR performs on the process when the process exits. See the \fBwait\fR(3C), manual page for the values \fBinit\fR uses. Applications creating \fButmpx\fR entries can set \fBut_exit\fR values using the following code example:
.sp
.in +2
.nf
u->ut_exit.e_termination = WTERMSIG(process->p_exit)
u->ut_exit.e_exit = WEXITSTATUS(process->p_exit)
.fi
.in -2

.sp
.LP
See \fBwait.h\fR(3HEAD) for descriptions of the \fBWTERMSIG\fR and \fBWEXITSTATUS\fR macros.
.sp
.LP
The \fBut_session\fR member is not acted upon by the operating system. It is used by applications interested in creating \fButmpx\fR entries.
.sp
.LP
For records of type \fBUSER_PROCESS\fR, the \fBnonuserx()\fR macro uses value of the \fBut_exit.e_exit\fR member to mark \fButmpx\fR entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each \fBpty\fR to have a \fButmpx\fR record (as most applications expect). The \fBNONROOT_USRX\fR macro defines the value that \fBlogin\fR places in the \fBut_exit.e_exit\fR member.
For records of type \fBUSER_PROCESS\fR, the \fBnonuser()\fR and \fBnonuserx()\fR macros use the value of the \fBut_exit.e_exit\fR member to mark \fButmpx\fR entries as real logins (as opposed to multiple xterms started by the same user on a window system). This allows the system utilities that display users to obtain an accurate indication of the number of actual users, while still permitting each \fBpty\fR to have a \fButmpx\fR record (as most applications expect.). The \fBNONROOT_USER\fR macro defines the value that \fBlogin\fR places in the \fBut_exit.e_exit\fR member.
.SH RETURN VALUES
.sp
.LP
Upon successful completion, \fBgetutxent()\fR, \fBgetutxid()\fR, and \fBgetutxline()\fR each return a pointer to a \fButmpx\fR structure containing

```
251 a copy of the requested entry in the user accounting database.  Otherwise a
252 null pointer is returned.
253 .sp
254 .LP
255 The return value may point to a static area which is overwritten by a
256 subsequent call to \fBgetutxid ()\fR or \fBgetutxline()\fR.
257 .sp
258 .LP
259 Upon successful completion, \fBpututxline()\fR returns a pointer to a
260 \fButmpx\fR structure containing a copy of the entry added to the user
261 accounting database.  Otherwise a null pointer is returned.
262 .sp
263 .LP
264 The \fBendutxent()\fR and \fBsetutxent()\fR functions return no value.
265 .sp
266 .LP
267 A null pointer is returned upon failure to read, whether for permissions or
268 having reached the end of file, or upon failure to write.
269 .SH USAGE
270 .sp
271 .LP
272 These functions use buffered standard I/O for input, but \fBpututxline()\fR
273 uses an unbuffered write to avoid race conditions between processes trying to
274 modify the \fButmpx\fR and \fBwtmpx\fR files.
275 .sp
276 .LP
277 Applications should not access the \fButmpx\fR and \fBwtmpx\fR databases
278 directly, but should use these functions to ensure that these databases are
279 maintained consistently.
280 .SH FILES
281 .sp
282 .ne 2
283 .na
284 \fB\fB/var/adm/utmpx\fR\fR
285 .ad
286 .RS 18n
287 user access and accounting information
288 .RE

290 .sp
291 .ne 2
292 .na
293 \fB\fB/var/adm/wtmpx\fR\fR
294 .ad
295 .RS 18n
296 history of user access and accounting information
297 .RE

299 .SH ATTRIBUTES
300 .sp
301 .LP
302 See \fBattributes\fR(5) for descriptions of the following attributes:
303 .sp

305 .sp
306 .TS
307 box;
308 c | c
309 l | l .
310 ATTRIBUTE TYPE  ATTRIBUTE VALUE
311 _
312 Interface Stability     See below.
313 _
314 MT-Level        Unsafe
315 .TE
```



```
317 .sp
318 .LP
319 The \fBendutxent()\fR, \fBgetutxent()\fR, \fBgetutxid()\fR, \fBgetutxline()\fR,
320 \fBpututxline()\fR, and \fBsetutxent()\fR functions are Standard.
321 .SH SEE ALSO
322 .sp
323 .LP
324 \fBgetutent\fR(3C), \fBttyslot\fR(3C), \fBwait\fR(3C), \fBwait.h\fR(3HEAD),
325 \fButmpx\fR(4), \fBattributes\fR(5), \fBstandards\fR(5)
326 .SH NOTES
327 .sp
328 .LP
329 The most current entry is saved in a static structure.  Multiple accesses
330 require that it be copied before further accesses are made. On each call to
331 either \fBgetutxid()\fR or \fBgetutxline()\fR, the routine examines the static
332 structure before performing more I/O. If the contents of the static structure
333 match what it is searching for, it looks no further. For this reason, to use
334 \fBgetutxline()\fR to search for multiple occurrences it would be necessary to
335 zero out the static after each success, or \fBgetutxline()\fR would just return
336 the same structure over and over again. There is one exception to the rule
337 about emptying the structure before further reads are done.  The implicit read
338 done by \fBpututxline()\fR (if it finds that it is not already at the correct
339 place in the file) will not hurt the contents of the static structure returned
340 by the \fBgetutxent()\fR, \fBgetutxid()\fR, or \fBgetutxline()\fR routines, if
341 the user has just modified those contents and passed the pointer back to
342 \fBpututxline()\fR.
```