```
**********************************************************
   18947 Thu Dec 19 12:20:25 2013
new/usr/src/cmd/w/w.c
2849 uptime should use locale settings for current time
**********************************************************
```

```
   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 (c) 2013 Gary Mills
  23  *
  24  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  25  * Use is subject to license terms.
  26  */

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

  31 /*
  32  * University Copyright- Copyright (c) 1982, 1986, 1988
  33  * The Regents of the University of California
  34  * All Rights Reserved
  35  *
  36  * University Acknowledgment- Portions of this document are derived from
  37  * software developed by the University of California, Berkeley, and its
  38  * contributors.
  39  */

  41 /*
  42  * This is the new w command which takes advantage of
  43  * the /proc interface to gain access to the information
  44  * of all the processes currently on the system.
  45  *
  46  * This program also implements 'uptime'.
  47  *
  48  * Maintenance note:
  49  *
  50  * Much of this code is replicated in whodo.c.  If you're
  51  * fixing bugs here, then you should probably fix 'em there too.
  52  */

  54 #include <stdio.h>
  55 #include <string.h>
  56 #include <stdarg.h>
  57 #include <stdlib.h>
  58 #include <ctype.h>
  59 #include <fcntl.h>
  60 #include <time.h>
  61 #include <errno.h>
```



```
  62 #include <sys/types.h>
  63 #include <utmpx.h>
  64 #include <sys/stat.h>
  65 #include <dirent.h>
  66 #include <procfs.h>             /* /proc header file */
  67 #include <locale.h>
  68 #include <unistd.h>
  69 #include <sys/loadavg.h>
  70 #include <limits.h>
  71 #include <priv_utils.h>

  73 /*
  74  * Use the full lengths from utmpx for user and line.
  75  */
  76 static struct utmpx dummy;
  77 #define NMAX            (sizeof (dummy.ut_user))
  78 #define LMAX            (sizeof (dummy.ut_line))

  80 /* Print minimum field widths. */
  81 #define LOGIN_WIDTH     8
  82 #define LINE_WIDTH      8
  82 #define LINE_WIDTH      12

  84 #define DIV60(t)        ((t+30)/60)     /* x/60 rounded */

  86 #ifdef ERR
  87 #undef ERR
  88 #endif
  89 #define ERR             (-1)

  91 #define HSIZE           256             /* size of process hash table   */
  92 #define PROCDIR         "/proc"
  93 #define INITPROCESS     (pid_t)1        /* init process pid */
  94 #define NONE            'n'             /* no state */
  95 #define RUNNING         'r'             /* runnable process */
  96 #define ZOMBIE          'z'             /* zombie process */
  97 #define VISITED         'v'             /* marked node as visited */
  98 #define PRINTF(a)       if (printf a < 0) { \
  99                 perror((gettext("%s: printf failed"), prog)); \
 100                 exit(1); }

 102 struct uproc {
 103         pid_t   p_upid;                 /* process id */
 104         char    p_state;                /* numeric value of process state */
 105         dev_t   p_ttyd;                 /* controlling tty of process */
 106         time_t  p_time;                 /* seconds of user & system time */
 107         time_t  p_ctime;                /* seconds of child user & sys time */
 108         int     p_igintr;               /* 1 = ignores SIGQUIT and SIGINT */
 109         char    p_comm[PRARGSZ+1];      /* command */
 110         char    p_args[PRARGSZ+1];      /* command line arguments */
 111         struct uproc    *p_child,       /* first child pointer */
 112                         *p_sibling,     /* sibling pointer */
 113                         *p_pgrpl,       /* pgrp link */
 114                         *p_link;        /* hash table chain pointer */
 115 };

 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);
130 static void     prttime(time_t, char *);
131 static void     prtat(time_t *time);
132 static void     checkampm(char *str);

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 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 (!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                                 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)));
312                             "login@  idle   JCPU   PCPU  what\n", LC_TIME)));
313                 } else {
314                         PRINTF((dcgettext(NULL,
315                             "User     tty         idle what\n",
316                             LC_TIME)));
315                             "User     tty           idle   what\n", 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));
488                     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]));
492                             PRINTF(("%-*.3s", LMAX, &ut->ut_line[4]));
495                     } else {
496                             PRINTF(("%-*.*s ", LINE_WIDTH, LMAX,
494                             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);
507             if (idle >= 36 * 60) {
508                     PRINTF((dcgettext(NULL, "%2ddays ", LC_TIME),
509                         (idle + 12 * 60) / (24 * 60)));
510             } else
511                     prttime(idle, " ");
```

```
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 }

519 /*
520  *  Prints the CPU time for all processes & children,
521  *  and the cpu time for interesting process,
522  *  and what the user is doing.
523  */
524 static void
525 showtotals(struct uproc *up)
526 {
527         jobtime = 0;
528         proctime = 0;
529         empty = 1;
530         curpid = -1;
531         add_times = 1;

533         calctotals(up);

535         if (lflag) {
536                 /* print CPU time for all processes & children */
537                 /* and need to convert clock ticks to seconds first */
538                 prttime((time_t)jobtime, 8);
540                 prttime((time_t)jobtime, " ");

540                 /* print cpu time for interesting process */
541                 /* and need to convert clock ticks to seconds first */
542                 prttime((time_t)proctime, 8);
544                 prttime((time_t)proctime, " ");
543         }
544         /* what user is doing, current process */
545         PRINTF(("%-.32s\n", doing));
547         PRINTF((" %-.32s\n", doing));
546 }
```

_____*unchanged_portion_omitted_*

```
642 #define HR      (60 * 60)
643 #define DAY     (24 * HR)
644 #define MON     (30 * DAY)

646 /*
647  * Prttime prints an elapsed time in hours, minutes, or seconds,
648  * right-justified with the rightmost column always blank.
649  * The second argument is the minimum field width.
649  * prttime prints a time in hours and minutes or minutes and seconds.
650  * The character string tail is printed at the end, obvious
651  * strings to pass are "", " ", or "am".
650  */
651 static void
652 prttime(time_t tim, int width)
654 prttime(time_t tim, char *tail)
653 {
654         char value[36];

656         if (tim >= 36 * 60) {
657                 (void) snprintf(value, sizeof (value), "%d:%02d:%02d",
658                     (int)tim / HR, (int)(tim % HR) / 60, (int)tim % 60);
659         } else if (tim >= 60) {
660                 (void) snprintf(value, sizeof (value), "%d:%02d",
661                     (int)tim / 60, (int)tim % 60);
```



```
656         if (tim >= 60) {
657                 PRINTF((dcgettext(NULL, "%3d:%02d", LC_TIME),
658                     (int)tim/60, (int)tim%60));
662         } else if (tim > 0) {
663                 (void) snprintf(value, sizeof (value), "%d", (int)tim);
660                 PRINTF((dcgettext(NULL, "    %2d", LC_TIME), (int)tim));
664         } else {
665                 (void) strcpy(value, "0");
662                 PRINTF(("      "));
666         }
667         width = (width > 2) ? width - 1 : 1;
668         PRINTF(("%*s ", width, value));
664         PRINTF(("%s", tail));
669 }

671 /*
672  * Prints the ISO date or time given a pointer to a time of day,
673  * left-justfied in a 12-character expanding field with the
674  * rightmost column always blank.
675  * Includes a dcgettext() override in case a message catalog is needed.
668  * prints a 12 hour time given a pointer to a time of day
676  */
677 static void
678 prtat(time_t *time)
679 {
680         struct tm       *p;

682         p = localtime(time);
683         if (now - *time <= 18 * HR) {
684                 char timestr[50];

686                 (void) strftime(timestr, sizeof (timestr),
687                     dcgettext(NULL, "%T", LC_TIME), p);
688                 PRINTF(("%-11s ", timestr));
679                     dcgettext(NULL, "%l:%M""%p", LC_TIME), p);
680                 checkampm(timestr);
681                 PRINTF((" %s", timestr));
689         } else if (now - *time <= 7 * DAY) {
690                 char weekdaytime[20];

692                 (void) strftime(weekdaytime, sizeof (weekdaytime),
693                     dcgettext(NULL, "%a %H:%M", LC_TIME), p);
694                 PRINTF(("%-11s ", weekdaytime));
686                     dcgettext(NULL, "%a%l%p", LC_TIME), p);
687                 checkampm(weekdaytime);
688                 PRINTF((" %s", weekdaytime));
695         } else {
696                 char monthtime[20];

698                 (void) strftime(monthtime, sizeof (monthtime),
699                     dcgettext(NULL, "%F", LC_TIME), p);
700                 PRINTF(("%-11s ", monthtime));
693                     dcgettext(NULL, "%e%b%y", LC_TIME), p);
694                 PRINTF((" %s", monthtime));
701         }
702 }
```

_____*unchanged_portion_omitted_*

```
742 /* replaces all occurences of AM/PM with am/pm */
743 static void
744 checkampm(char *str)
745 {
746         char *ampm;
747         while ((ampm = strstr(str, "AM")) != NULL ||
748             (ampm = strstr(str, "PM")) != NULL) {
749                 *ampm = tolower(*ampm);
```

```
750                 *(ampm+1) = tolower(*(ampm+1));
751         }
752 }
```

```
**********************************************************
   20943 Thu Dec 19 12:20:25 2013
new/usr/src/cmd/whodo/whodo.c
2849 uptime should use locale settings for current time
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (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) 2013 Gary Mills
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * This is the new whodo command which takes advantage of
 * the /proc interface to gain access to the information
 * of all the processes currently on the system.
 *
 * Maintenance note:
 *
 * Much of this code is replicated in w.c.  If you're
 * fixing bugs here, then you should probably fix 'em there too.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <time.h>
#include <errno.h>
#include <sys/types.h>
#include <utmpx.h>
#include <sys/utsname.h>
```



```
#include <sys/stat.h>
#include <sys/mkdev.h>
#include <dirent.h>
#include <procfs.h>             /* /proc header file */
#include <sys/wait.h>
#include <locale.h>
#include <unistd.h>
#include <limits.h>
#include <priv_utils.h>

/*
 * Use the full lengths from utmpx for user and line.
 */
#define NMAX    (sizeof (((struct utmpx *)0)->ut_user))
#define LMAX    (sizeof (((struct utmpx *)0)->ut_line))

/* Print minimum field widths. */
#define LOGIN_WIDTH     8
#define LINE_WIDTH      8
#define LINE_WIDTH      12

#define DIV60(t)        ((t+30)/60)    /* x/60 rounded */

#ifdef ERR
#undef ERR
#endif
#define ERR             (-1)

#define DEVNAMELEN      14
#define HSIZE           256             /* size of process hash table */
#define PROCDIR         "/proc"
#define INITPROCESS     (pid_t)1        /* init process pid */
#define NONE            'n'             /* no state */
#define RUNNING         'r'             /* runnable process */
#define ZOMBIE          'z'             /* zombie process */
#define VISITED         'v'             /* marked node as visited */

static int      ndevs;                  /* number of configured devices */
static int      maxdev;                 /* slots for configured devices */
#define DNINCR  100
static struct devl {                    /* device list   */
        char    dname[DEVNAMELEN];      /* device name   */
        dev_t   ddev;                   /* device number */
} *devl;
______unchanged_portion_omitted_

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

static struct   uproc   *findhash(pid_t);
static time_t   findidle(char *);
static void     clnarglist(char *);
static void     showproc(struct uproc *);
static void     showtotals(struct uproc *);
static void     calctotals(struct uproc *);
static char     *getty(dev_t);
static void     prttime(time_t, int);
static void     prttime(time_t, char *);
static void     prtat(time_t *);
static void     checkampm(char *);
```

```
static char     *prog;
static int      header = 1;     /* true if -h flag: don't print heading */
static int      lflag = 0;      /* true if -l flag: w command format */
static char     *sel_user;      /* login of particular user selected */
static time_t   now;            /* current time of day */
static time_t   uptime;         /* time of last reboot & elapsed time since */
static int      nusers;         /* number of users logged in now */
static time_t   idle;           /* number of minutes user is idle */
static time_t   jobtime;        /* total cpu time visible */
static char     doing[520];     /* process attached to terminal */
static time_t   proctime;       /* cpu time of process in doing */
static int      empty;
static pid_t    curpid;

#if SIGQUIT > SIGINT
#define ACTSIZE SIGQUIT
#else
#define ACTSIZE SIGINT
#endif

int
main(int argc, char *argv[])
{
        struct utmpx    *ut;
        struct utmpx    *utmpbegin;
        struct utmpx    *utmpend;
        struct utmpx    *utp;
        struct tm               *tm;
        struct uproc    *up, *parent, *pgrp;
        struct psinfo   info;
        struct sigaction actinfo[ACTSIZE];
        struct pstatus  statinfo;
        size_t          size;
        struct stat     sbuf;
        struct utsname  uts;
        DIR             *dirp;
        struct  dirent  *dp;
        char            pname[64];
        char            *fname;
        int             procfd;
        int             i;
        int             days, hrs, mins;
        int             entries;

        /*
         * This program needs the proc_owner privilege
         */
        (void) __init_suid_priv(PU_CLEARLIMITSET, PRIV_PROC_OWNER,
            (char *)NULL);

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)
#define TEXT_DOMAIN "SYS_TEST"
#endif
        (void) textdomain(TEXT_DOMAIN);

        prog = argv[0];

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

                                case 'h':
                                        header = 0;
                                        break;
```



```
                                case 'l':
                                        lflag++;
                                        break;

                                default:
                                        (void) printf(gettext(
                                            "usage: %s [ -hl ] [ user ]\n"),
                                            prog);
                                        exit(1);
                                }
                        }
                } else {
                        if (!isalnum(argv[1][0]) || argc > 2) {
                                (void) printf(gettext(
                                    "usage: %s [ -hl ] [ user ]\n"), prog);
                                exit(1);
                        } else
                                sel_user = argv[1];
                }
                argc--; argv++;
        }

        /*
         * read the UTMPX_FILE (contains information about
         * each logged in user)
         */
        if (stat(UTMPX_FILE, &sbuf) == ERR) {
                (void) fprintf(stderr, gettext("%s: stat error of %s: %s\n"),
                    prog, UTMPX_FILE, strerror(errno));
                exit(1);
        }
        entries = sbuf.st_size / sizeof (struct futmpx);
        size = sizeof (struct utmpx) * entries;

        if ((ut = malloc(size)) == NULL) {
                (void) fprintf(stderr, gettext("%s: malloc error of %s: %s\n"),
                    prog, UTMPX_FILE, strerror(errno));
                exit(1);
        }

        (void) utmpxname(UTMPX_FILE);

        utmpbegin = ut;
        /* LINTED pointer cast may result in improper alignment */
        utmpend = (struct utmpx *)((char *)utmpbegin + size);

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

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

        if (header) {   /* print a header */
                if (lflag) {    /* w command format header */
                        prtat(&now);
                        for (ut = utmpbegin; ut < utmpend; ut++) {
                                if (ut->ut_type == USER_PROCESS) {
                                        nusers++;
                                } else if (ut->ut_type == BOOT_TIME) {
                                        uptime = now - ut->ut_xtime;
                                        uptime += 30;
                                        days = uptime / (60*60*24);
                                        uptime %= (60*60*24);
                                        hrs = uptime / (60*60);
```

```
271                                     uptime %= (60*60);
272                                     mins = uptime / 60;

274                                     (void) printf(dcgettext(NULL,
275                                         "up %d day(s), %d hr(s), "
276                                         "  up %d day(s), %d hr(s), "
276                                         "%d min(s)", LC_TIME),
277                                         days, hrs, mins);
278                                 }
279                         }

281                         ut = utmpbegin; /* rewind utmp data */
282                         (void) printf(dcgettext(NULL,
283                             "  %d user(s)\n", LC_TIME), nusers);
284                         (void) printf(dcgettext(NULL, "User     tty      "
285                             "login@         idle    JCPU    PCPU what\n",
286                             LC_TIME));
286                             "login@  idle   JCPU   PCPU  what\n", LC_TIME));
287                 } else {        /* standard whodo header */
288                         char date_buf[100];

290                         /*
291                          * print current time and date
292                          */
293                         (void) strftime(date_buf, sizeof (date_buf),
294                             "%c", localtime(&now));
294                             dcgettext(NULL, "%C", LC_TIME), localtime(&now));
295                         (void) printf("%s\n", date_buf);

297                         /*
298                          * print system name
299                          */
300                         (void) uname(&uts);
301                         (void) printf("%s\n", uts.nodename);
302                 }
303         }

305         /*
306          * loop through /proc, reading info about each process
307          * and build the parent/child tree
308          */
309         if (!(dirp = opendir(PROCDIR))) {
310                 (void) fprintf(stderr, gettext("%s: could not open %s: %s\n"),
311                     prog, PROCDIR, strerror(errno));
312                 exit(1);
313         }

315         while ((dp = readdir(dirp)) != NULL) {
316                 if (dp->d_name[0] == '.')
317                         continue;
318 retry:
319                 (void) snprintf(pname, sizeof (pname),
320                     "%s/%s/", PROCDIR, dp->d_name);
321                 fname = pname + strlen(pname);
322                 (void) strcpy(fname, "psinfo");
323                 if ((procfd = open(pname, O_RDONLY)) < 0)
324                         continue;
325                 if (read(procfd, &info, sizeof (info)) != sizeof (info)) {
326                         int err = errno;
327                         (void) close(procfd);
328                         if (err == EAGAIN)
329                                 goto retry;
330                         if (err != ENOENT)
331                                 (void) fprintf(stderr, gettext(
332                                     "%s: read() failed on %s: %s\n"),
333                                     prog, pname, strerror(err));
```



```
334                         continue;
335                 }
336                 (void) close(procfd);

338                 up = findhash(info.pr_pid);
339                 up->p_ttyd = info.pr_ttydev;
340                 up->p_state = (info.pr_nlwp == 0? ZOMBIE : RUNNING);
341                 up->p_time = 0;
342                 up->p_ctime = 0;
343                 up->p_igintr = 0;
344                 (void) strncpy(up->p_comm, info.pr_fname,
345                     sizeof (info.pr_fname));
346                 up->p_args[0] = 0;

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

351                         /* now we need the proc_owner privilege */
352                         (void) __priv_bracket(PRIV_ON);

354                         procfd = open(pname, O_RDONLY);

356                         /* drop proc_owner privilege after open */
357                         (void) __priv_bracket(PRIV_OFF);

359                         if (procfd  < 0)
360                                 continue;

362                         if (read(procfd, &statinfo, sizeof (statinfo))
363                             != sizeof (statinfo)) {
364                                 int err = errno;
365                                 (void) close(procfd);
366                                 if (err == EAGAIN)
367                                         goto retry;
368                                 if (err != ENOENT)
369                                         (void) fprintf(stderr, gettext(
370                                             "%s: read() failed on %s: %s \n"),
371                                             prog, pname, strerror(err));
372                                 continue;
373                         }
374                         (void) close(procfd);

376                         up->p_time = statinfo.pr_utime.tv_sec +
377                             statinfo.pr_stime.tv_sec;
378                         up->p_ctime = statinfo.pr_cutime.tv_sec +
379                             statinfo.pr_cstime.tv_sec;

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

383                         /* now we need the proc_owner privilege */
384                         (void) __priv_bracket(PRIV_ON);

386                         procfd = open(pname, O_RDONLY);

388                         /* drop proc_owner privilege after open */
389                         (void) __priv_bracket(PRIV_OFF);

391                         if (procfd  < 0)
392                                 continue;
393                         if (read(procfd, actinfo, sizeof (actinfo))
394                             != sizeof (actinfo)) {
395                                 int err = errno;
396                                 (void) close(procfd);
397                                 if (err == EAGAIN)
398                                         goto retry;
399                                 if (err != ENOENT)
```

```
400                                     (void) fprintf(stderr, gettext(
401                                         "%s: read() failed on %s: %s \n"),
402                                         prog, pname, strerror(err));
403                             continue;
404                     }
405                     (void) close(procfd);

407                     up->p_igintr =
408                         actinfo[SIGINT-1].sa_handler == SIG_IGN &&
409                         actinfo[SIGQUIT-1].sa_handler == SIG_IGN;

411                     up->p_args[0] = 0;

413                     /*
414                      * Process args if there's a chance we'll print it.
415                      */
416                     if (lflag) { /* w command needs args */
417                             clnarglist(info.pr_psargs);
418                             (void) strcpy(up->p_args, info.pr_psargs);
419                             if (up->p_args[0] == 0 ||
420                                 up->p_args[0] == '-' &&
421                                 up->p_args[1] <= ' ' ||
422                                 up->p_args[0] == '?') {
423                                     (void) strcat(up->p_args, " (");
424                                     (void) strcat(up->p_args, up->p_comm);
425                                     (void) strcat(up->p_args, ")");
426                             }
427                     }

429             }

431             /*
432              * link pgrp together in case parents go away
433              * Pgrp chain is a single linked list originating
434              * from the pgrp leader to its group member.
435              */
436             if (info.pr_pgid != info.pr_pid) {      /* not pgrp leader */
437                     pgrp = findhash(info.pr_pgid);
438                     up->p_pgrplink = pgrp->p_pgrplink;
439                     pgrp->p_pgrplink = up;
440             }
441             parent = findhash(info.pr_ppid);

443             /* if this is the new member, link it in */
444             if (parent->p_upid != INITPROCESS) {
445                     if (parent->p_child) {
446                             up->p_sibling = parent->p_child;
447                             up->p_child = 0;
448                     }
449                     parent->p_child = up;
450             }

452         }

454         /* revert to non-privileged user */
455         (void) __priv_relinquish();

457         (void) closedir(dirp);
458         (void) time(&now);      /* get current time */

460         /*
461          * loop through utmpx file, printing process info
462          * about each logged in user
463          */
464         for (ut = utmpbegin; ut < utmpend; ut++) {
465                 time_t tim;
```



```
467                 if (ut->ut_type != USER_PROCESS)
468                         continue;
469                 if (sel_user && strncmp(ut->ut_name, sel_user, NMAX) != 0)
470                         continue;       /* we're looking for somebody else */
471                 if (lflag) {    /* -l flag format (w command) */
472                         /* print login name of the user */
473                         (void) printf("%-*.*s ", LOGIN_WIDTH, (int)NMAX,
474                             ut->ut_name);

476                         /* print tty user is on */
477                         (void) printf("%-*.*s ", LINE_WIDTH, (int)LMAX,
477                         (void) printf("%-*.*s", LINE_WIDTH, (int)LMAX,
478                             ut->ut_line);

480                         /* print when the user logged in */
481                         tim = ut->ut_xtime;
482                         (void) prtat(&tim);

484                         /* print idle time */
485                         idle = findidle(ut->ut_line);
486                         prttime(idle, 8);
486                         if (idle >= 36 * 60)
487                                 (void) printf(dcgettext(NULL, "%2ddays ",
488                                     LC_TIME), (idle + 12 * 60) / (24 * 60));
489                         else
490                                 prttime(idle, " ");
487                         showtotals(findhash((pid_t)ut->ut_pid));
488                 } else {        /* standard whodo format */
489                         tim = ut->ut_xtime;
490                         tm = localtime(&tim);
491                         (void) printf("\n%-*.*s %-*.*s %2.1d:%2.2d\n",
492                             LINE_WIDTH, (int)LMAX, ut->ut_line,
493                             LOGIN_WIDTH, (int)NMAX, ut->ut_name, tm->tm_hour,
494                             tm->tm_min);
495                         showproc(findhash((pid_t)ut->ut_pid));
496                 }
497         }

499         return (0);
500 }
_____unchanged_portion_omitted_


543 /*
544  * Used for -l flag (w command) format.
545  * Prints the CPU time for all processes & children,
546  * and the cpu time for interesting process,
547  * and what the user is doing.
548  */
549 static void
550 showtotals(struct uproc *up)
551 {
552         jobtime = 0;
553         proctime = 0;
554         empty = 1;
555         curpid = -1;
556         (void) strcpy(doing, "-"); /* default act: normally never prints */
557         calctotals(up);

559         /* print CPU time for all processes & children */
560         /* and need to convert clock ticks to seconds first */
561         prttime((time_t)jobtime, 8);
565         prttime((time_t)jobtime, " ");

563         /* print cpu time for interesting process */
```

```
564         /* and need to convert clock ticks to seconds first */
565         prttime((time_t)proctime, 8);
569         prttime((time_t)proctime, " ");

567         /* what user is doing, current process */
568         (void) printf("%-.32s\n", doing);
572         (void) printf(" %-.32s\n", doing);
569 }
______unchanged_portion_omitted_

732 #define HR      (60 * 60)
733 #define DAY     (24 * HR)
734 #define MON     (30 * DAY)
735 #define PRINTF(a)       (void) printf a

737 /*
738  * Prttime prints an elapsed time in hours, minutes, or seconds,
739  * right-justified with the rightmost column always blank.
740  * The second argument is the minimum field width.
741  * prints a time in hours and minutes or minutes and seconds.
742  * The character string 'tail' is printed at the end, obvious
743  * strings to pass are "", " ", or "am".
741  */
742 static void
743 prttime(time_t tim, int width)
746 prttime(time_t tim, char *tail)
744 {
745         char value[36];

747         if (tim >= 36 * 60) {
748                 (void) snprintf(value, sizeof (value), "%d:%02d:%02d",
749                     (int)tim / HR, (int)(tim % HR) / 60, (int)tim % 60);
750         } else if (tim >= 60) {
751                 (void) snprintf(value, sizeof (value), "%d:%02d",
752                     (int)tim / 60, (int)tim % 60);
753         } else if (tim > 0) {
754                 (void) snprintf(value, sizeof (value), "%d", (int)tim);
755         } else {
756                 (void) strcpy(value, "0");
757         }
758         width = (width > 2) ? width - 1 : 1;
759         PRINTF(("%*s ", width, value));
748         if (tim >= 60)
749                 (void) printf(dcgettext(NULL, "%3d:%02d", LC_TIME),
750                     (int)tim/60, (int)tim%60);
751         else if (tim > 0)
752                 (void) printf(dcgettext(NULL, "    %2d", LC_TIME), (int)tim);
753         else
754                 (void) printf("      ");
755         (void) printf("%s", tail);
760 }


762 /*
763  * Prints the ISO date or time given a pointer to a time of day,
764  * left-justfied in a 12-character expanding field with the
765  * rightmost column always blank.
766  * Includes a dcgettext() override in case a message catalog is needed.
760  * prints a 12 hour time given a pointer to a time of day
767  */
768 static void
769 prtat(time_t *time)
770 {
771         struct tm       *p;

773         p = localtime(time);
```



```
774         if (now - *time <= 18 * HR) {
775                 char timestr[50];

777                 (void) strftime(timestr, sizeof (timestr),
778                     dcgettext(NULL, "%T", LC_TIME), p);
779                 PRINTF(("%-11s ", timestr));
771                     dcgettext(NULL, " %l:%M""%p", LC_TIME), p);
772                 checkampm(timestr);
773                 (void) printf("%s", timestr);
780         } else if (now - *time <= 7 * DAY) {
781                 char weekdaytime[20];

783                 (void) strftime(weekdaytime, sizeof (weekdaytime),
784                     dcgettext(NULL, "%a %H:%M", LC_TIME), p);
785                 PRINTF(("%-11s ", weekdaytime));
778                     dcgettext(NULL, "%a%l%p", LC_TIME), p);
779                 checkampm(weekdaytime);
780                 (void) printf(" %s", weekdaytime);
786         } else {
787                 char monthtime[20];

789                 (void) strftime(monthtime, sizeof (monthtime),
790                     dcgettext(NULL, "%F", LC_TIME), p);
791                 PRINTF(("%-11s ", monthtime));
785                     dcgettext(NULL, "%e%b%y", LC_TIME), p);
786                 (void) printf(" %s", monthtime);
792         }
793 }
______unchanged_portion_omitted_

834 /* replaces all occurences of AM/PM with am/pm */
835 static void
836 checkampm(char *str)
837 {
838         char *ampm;
839         while ((ampm = strstr(str, "AM")) != NULL ||
840             (ampm = strstr(str, "PM")) != NULL) {
841                 *ampm = tolower(*ampm);
842                 *(ampm+1) = tolower(*(ampm+1));
843         }
844 }
```

```
**********************************************************
    4757 Thu Dec 19 12:20:25 2013
new/usr/src/man/man1/w.1
2849 uptime should use locale settings for current time
**********************************************************
'\" te
.\" Copyright (c) 2013 Gary Mills
.\"  Copyright (c) 2004, Sun Microsystems, Inc. All Rights Reserved.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH W 1 "Dec 15, 2013"
.TH W 1 "Mar 19, 2004"
.SH NAME
w \- display information about currently logged-in users
.SH SYNOPSIS
.LP
.nf
\fBw\fR [\fB-hlsuw\fR] [\fIuser\fR]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBw\fR command displays a summary of the current activity on the system,
including what each user is doing. The heading line shows the current time, the
length of time the system has been up, the number of users logged into the
system, and the average number of jobs in the run queue over the last 1, 5 and
15 minutes.
.sp
.LP
The fields displayed are: the user's login name,
the name of the tty the user is on,
the time of day the user logged on (in ISO time format, weekday name
and \fIhours:minutes\fR, or ISO date format), the idle
time\(emthat is, the number of minutes since the user last typed anything
(in \fIhours:minutes:seconds\fR),
the \fBCPU\fR time used by all processes and their
children on that terminal (in \fIhours:minutes:seconds\fR),
the \fBCPU\fR time used
by the currently active processes (in \fIhours:minutes:seconds\fR),
and the name and
The fields displayed are: the user's login name, the name of the tty the user
is on, the time of day the user logged on (in \fIhours:minutes\fR), the idle
time\(emthat is, the number of minutes since the user last typed anything (in
\fIhours:minutes\fR), the \fBCPU\fR time used by all processes and their
children on that terminal (in \fIminutes:seconds\fR), the \fBCPU\fR time used
by the currently active processes (in \fIminutes:seconds\fR), and the name and
arguments of the current process.
.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-h\fR\fR
.ad
.RS 6n
Suppresses the heading.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR\fR
```



```
.ad
.RS 6n
Produces a long form of output, which is the default.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR\fR
.ad
.RS 6n
Produces a short form of output. In the short form, the tty is abbreviated, the
login time and \fBCPU\fR times are left off, as are the arguments to commands.
.RE

.sp
.ne 2
.na
\fB\fB-u\fR\fR
.ad
.RS 6n
Produces the heading line which shows the current time, the length of time the
system has been up, the number of users logged into the system, and the average
number of jobs in the run queue over the last 1, 5 and 15 minutes.
.RE

.sp
.ne 2
.na
\fB\fB-w\fR\fR
.ad
.RS 6n
Produces a long form of output, which is also the same as the default.
.RE

.SH OPERANDS
.sp
.ne 2
.na
\fB\fIuser\fR\fR
.ad
.RS 8n
Name of a particular user for whom login information is displayed. If
specified, output is restricted to that user.
.RE

.SH EXAMPLES
.LP
\fBExample 1 \fRSample Output From the \fBw\fR Command
.sp
.in +2
.nf
example% \fBw\fR


 10:54am  up 27 day(s), 57 mins,  1 user,  load average: 0.28, 0.26, 0.22
User       tty           login@    idle         JCPU    PCPU     what
ralph    console  7:10am    1       10:05   4:31     w
.fi
.in -2
.sp

.SH ENVIRONMENT VARIABLES
.sp
.LP
See  \fBenviron\fR(5) for descriptions of the following environment variables
```

```
121 that affect the execution of \fBw\fR: \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, and
122 \fBLC_TIME\fR.
123 .SH FILES
124 .sp
125 .ne 2
126 .na
127 \fB\fB/var/adm/utmpx\fR\fR
128 .ad
129 .RS 18n
130 user and accounting information
131 .RE

133 .SH SEE ALSO
134 .sp
135 .LP
136 \fBps\fR(1), \fBwho\fR(1), \fBwhodo\fR(1M), \fButmpx\fR(4),
137 \fBattributes\fR(5), \fBenviron\fR(5)
138 .SH NOTES
139 .sp
140 .LP
141 The notion of the "current process" is unclear. The current algorithm is "the
142 highest numbered process on the terminal that is not ignoring interrupts, or,
143 if there is none, the highest numbered process on the terminal". This fails,
144 for example, in critical sections of programs like the shell and editor, or
145 when faulty programs running in the background fork and fail to ignore
146 interrupts. In cases where no process can be found, \fBw\fR prints
147 \fB\(mi\fR\&.
148 .sp
149 .LP
150 The \fBCPU\fR time is only an estimate, in particular, if someone leaves a
151 background process running after logging out, the person currently on that
152 terminal is ‘‘charged’’ with the time.
153 .sp
154 .LP
155 Background processes are not shown, even though they account for much of the
156 load on the system.
157 .sp
158 .LP
159 Sometimes processes, typically those in the background, are printed with null
160 or garbaged arguments. In these cases, the name of the command is printed in
161 parentheses.
162 .sp
163 .LP
164 \fBw\fR does not know about the conventions for detecting background jobs. It
165 will sometimes find a background job instead of the right one.
```

```
**********************************************************
    5224 Thu Dec 19 12:20:25 2013
new/usr/src/man/man1m/whodo.1m
2849 uptime should use locale settings for current time
**********************************************************
   1 '\" te
   2 .\" Copyright (c) 2013 Gary Mills
   3 .\" Copyright (c) 2001 Sun Microsystems, Inc. All Rights Reserved.
   4 .\" Copyright 1989 AT&T
   5 .\" The contents of this file are subject to the terms of the Common Development
   6 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
   7 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
   8 .TH WHODO 1M "Dec 15, 2013"
   7 .TH WHODO 1M "Jun 18, 2003"
   9 .SH NAME
  10 whodo \- who is doing what
  11 .SH SYNOPSIS
  12 .LP
  13 .nf
  14 \fB/usr/sbin/whodo\fR [\fB-h\fR] [\fB-l\fR] [\fIuser\fR]
  15 .fi

  17 .SH DESCRIPTION
  18 .sp
  19 .LP
  20 The \fBwhodo\fR command produces formatted and dated output from information in
  21 the \fB/var/adm/utmpx\fR and \fB/proc/pid\fR files.
  22 .sp
  23 .LP
  24 The display is headed by the date, time, and machine name. For each user logged
  25 in, device name, user-ID and login time is shown, followed by a list of active
  26 processes associated with the user-ID. The list includes the device name,
  27 process-ID, CPU minutes and seconds used, and process name.
  28 .sp
  29 .LP
  30 If \fIuser\fR is specified, output is restricted to all sessions pertaining to
  31 that user.
  32 .SH OPTIONS
  33 .sp
  34 .LP
  35 The following options are supported:
  36 .sp
  37 .ne 2
  38 .na
  39 \fB\fB-h\fR\fR
  40 .ad
  41 .RS 6n
  42 Suppress the heading.
  43 .RE

  45 .sp
  46 .ne 2
  47 .na
  48 \fB\fB-l\fR\fR
  49 .ad
  50 .RS 6n
  51 Produce a long form of output. The fields displayed are: the user's login name,
  52 the name of the tty the user is on, the time of day the user logged in
  53 (in ISO time format, weekday name and \fIhours\fR\fB:\fR\fIminutes\fR,
  54 or ISO date format),
  55 the idle time \(em that is, the time since the user last typed anything
  56 (in \fIhours\fR\fB:\fR\fIminutes\fR\fB:\fR\fIseconds\fR),
  57 the CPU time used by all processes and their children on that terminal
  58 (in \fIhours\fR\fB:\fR\fIminutes\fR\fB:\fR\fIseconds\fR),
  59 the CPU time used by the currently active processes
  60 (in \fIhours\fR\fB:\fR\fIminutes\fR\fB:\fR\fIseconds\fR),
```



```
  61 and the name and arguments of the current process.
  51 the name of the tty the user is on, the time of day the user logged in (in
  52 \fIhours\fR\fB:\fR\fIminutes\fR), the idle time \(em that is, the time since
  53 the user last typed anything (in \fIhours\fR\fB:\fR\fIminutes\fR), the CPU time
  54 used by all processes and their children on that terminal (in
  55 \fIminutes\fR\fB:\fR\fIseconds\fR), the CPU time used by the currently active
  56 processes (in \fIminutes\fR\fB:\fR\fIseconds\fR), and the name and arguments of
  57 the current process.
  62 .RE

  64 .SH EXAMPLES
  65 .LP
  66 \fBExample 1 \fRUsing the whodo Command
  67 .sp
  68 .LP
  69 The command:

  71 .sp
  72 .in +2
  73 .nf
  74 example% whodo
  75 .fi
  76 .in -2
  77 .sp

  79 .sp
  80 .LP
  81 produces a display like this:

  83 .sp
  84 .in +2
  85 .nf
  86 Tue Mar 12 15:48:03 1985
  87 bailey
  88 tty09    mcn       8:51
  89     tty09   28158    0:29 sh

  91 tty52    bdr      15:23
  92     tty52   21688    0:05 sh
  93     tty52   22788    0:01 whodo
  94     tty52   22017    0:03 vi
  95     tty52   22549    0:01 sh

  97 xt162    lee      10:20
  98     tty08    6748    0:01 layers
  99     xt162    6751    0:01 sh
 100     xt163    6761    0:05 sh
 101     tty08    6536    0:05 sh
 102 .fi
 103 .in -2
 104 .sp

 106 .SH ENVIRONMENT VARIABLES
 107 .sp
 108 .LP
 109 If any of the \fBLC_*\fR variables ( \fB\fR\fBLC_CTYPE\fR\fB,\fR
 110 \fBLC_MESSAGES\fR\fB,\fR \fBLC_TIME\fR\fB,\fR \fBLC_COLLATE\fR\fB,\fR
 111 \fBLC_NUMERIC\fR\fB,\fR and \fBLC_MONETARY\fR ) (see \fBenviron\fR(5)) are not
 112 set in the environment, the operational behavior of \fBtar\fR(1) for each
 113 corresponding locale category is determined by the value of the \fBLANG\fR
 114 environment variable. If \fBLC_ALL\fR is set, its contents are used to override
 115 both the \fBLANG\fR and the other \fBLC_*\fR variables. If none of the above
 116 variables is set in the environment, the "C" (U.S. style) locale determines how
 117 \fBwhodo\fR behaves.
 118 .sp
 119 .ne 2
```

```
120 .na
121 \fB\fBLC_CTYPE\fR\fR
122 .ad
123 .RS 15n
124 Determines how \fBwhodo\fR handles characters. When \fBLC_CTYPE\fR is set to a
125 valid value, \fBwhodo\fR can display and handle text and filenames containing
126 valid characters for that locale. The \fBwhodo\fR command can display and
127 handle Extended Unix code (EUC) characters where any individual character can
128 be 1, 2, or 3 bytes wide. \fBwhodo\fR can also handle EUC characters of 1, 2,
129 or more column widths. In the "C" locale, only characters from ISO 8859-1 are
130 valid.
131 .RE

133 .sp
134 .ne 2
135 .na
136 \fB\fBLC_MESSAGES\fR\fR
137 .ad
138 .RS 15n
139 Determines how diagnostic and informative messages are presented. This includes
140 the language and style of the messages, and the correct form of affirmative and
141 negative responses. In the "C" locale, the messages are presented in the
142 default form found in the program itself (in most cases, U.S. English).
143 .RE

145 .sp
146 .ne 2
147 .na
148 \fB\fBLC_TIME\fR\fR
149 .ad
150 .RS 15n
151 Determines how \fBwhodo\fR handles date and time formats. In the "C" locale,
152 date and time handling follow the U.S. rules.
153 .RE

155 .SH EXIT STATUS
156 .sp
157 .LP
158 The following exit values are returned:
159 .sp
160 .ne 2
161 .na
162 \fB\fB0\fR\fR
163 .ad
164 .RS 12n
165 Successful completion.
166 .RE

168 .sp
169 .ne 2
170 .na
171 \fBnon-zero\fR
172 .ad
173 .RS 12n
174 An error occurred.
175 .RE

177 .SH FILES
178 .sp
179 .ne 2
180 .na
181 \fB\fB/etc/passwd\fR\fR
182 .ad
183 .RS 18n
184 System password file
185 .RE
```



```
187 .sp
188 .ne 2
189 .na
190 \fB\fB/var/adm/utmpx\fR\fR
191 .ad
192 .RS 18n
193 User access and administration information
194 .RE

196 .sp
197 .ne 2
198 .na
199 \fB\fB/proc/pid\fR\fR
200 .ad
201 .RS 18n
202 Contains PID
203 .RE

205 .SH SEE ALSO
206 .sp
207 .LP
208 \fBps\fR(1), \fBwho\fR(1), \fBattributes\fR(5), \fBenviron\fR(5)
```