```
**********************************************************
   18927 Sat Nov 30 09:38:52 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                                 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);
394
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;
399
400                     (void) strcpy(fname, "sigact");
401
402                     /* now we need the proc_owner privilege */
403                     (void) __priv_bracket(PRIV_ON);
404
405                     procfd = open(pname, O_RDONLY);
406
407                     /* drop proc_owner privilege after open */
408                     (void) __priv_bracket(PRIV_OFF);
409
410                     if (procfd < 0)
411                             continue;
412
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);
426
427                     up->p_igintr =
428                         actinfo[SIGINT-1].sa_handler == SIG_IGN &&
429                         actinfo[SIGQUIT-1].sa_handler == SIG_IGN;
430
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             }
445
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);
457
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     }
467
468     /* revert to non-privileged user after opening */
469     (void) __priv_relinquish();
470
471     (void) closedir(dirp);
472     (void) time(&now);      /* get current time */
473
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 */
483
484             /* print login name of the user */
485             PRINTF(("%-*.*s ", LOGIN_WIDTH, NMAX, ut->ut_name));
486
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             }
500
501             /* print when the user logged in */
502             if (lflag) {
503                     time_t tim = ut->ut_xtime;
504                     prtat(&tim);
505             }
506
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 a time in days, hours, minutes, or seconds.
648  * The second argument is the 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".
649  */
650 static void
651 prttime(time_t tim, int width)
654 prttime(time_t tim, char *tail)
652 {
653         char value[12];
654         char *unit;

656         if (tim >= 36 * HR) {
657                 (void) snprintf(value, sizeof (value), "%d",
658                     (tim + (DAY / 2)) / (DAY));
659                 unit = dcgettext(NULL, "days", LC_TIME);
660         } else if (tim >= 36 * 60) {
661                 (void) snprintf(value, sizeof (value), "%d",
662                     (tim + (HR / 2)) / (HR));
```



```
663                 unit = dcgettext(NULL, "hours", LC_TIME);
664         } else if (tim >= 60) {
665                 (void) snprintf(value, sizeof (value), "%d",
666                     (tim + 30) / 60);
667                 unit = dcgettext(NULL, "mins", LC_TIME);
656         if (tim >= 60) {
657                 PRINTF((dcgettext(NULL, "%3d:%02d", LC_TIME),
658                     (int)tim/60, (int)tim%60));
668         } else if (tim > 0) {
669                 (void) snprintf(value, sizeof (value), "%d", (int)tim);
670                 unit = dcgettext(NULL, "secs", LC_TIME);
660                 PRINTF((dcgettext(NULL, "    %2d", LC_TIME), (int)tim));
671         } else {
672                 (void) strcpy(value, "0");
673                 unit = " ";
662                 PRINTF((("      "));
674         }
675         width -= 2 + strlen(value);
676         width = (width > 1) ? width : 1;
677         PRINTF(("%s %-*s ", value, width, unit));
664         PRINTF(("%s", tail));
678 }

680 /*
681  * prints a locale-specific time given a pointer to a time of day
668  * prints a 12 hour time given a pointer to a time of day
682  */
683 static void
684 prtat(time_t *time)
685 {
686         struct tm       *p;

688         p = localtime(time);
689         if (now - *time <= 18 * HR) {
690                 char timestr[50];

692                 (void) strftime(timestr, sizeof (timestr),
693                     "%X", p);
694                 PRINTF(("%-11s ", timestr));
679                     dcgettext(NULL, "%l:%M""%p", LC_TIME), p);
680                 checkampm(timestr);
681                 PRINTF((" %s", timestr));
695         } else if (now - *time <= 7 * DAY) {
696                 char weekdaytime[20];

698                 (void) strftime(weekdaytime, sizeof (weekdaytime),
699                     "%d %b", p);
700                 PRINTF(("%-11s ", weekdaytime));
686                     dcgettext(NULL, "%a%l%p", LC_TIME), p);
687                 checkampm(weekdaytime);
688                 PRINTF((" %s", weekdaytime));
701         } else {
702                 char monthtime[20];

704                 (void) strftime(monthtime, sizeof (monthtime),
705                     "%b %Y", p);
706                 PRINTF(("%-11s ", monthtime));
693                     dcgettext(NULL, "%e%b%y", LC_TIME), p);
694                 PRINTF((" %s", monthtime));
707         }
708 }
_____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 }
```

```
**********************************************************
   20882 Sat Nov 30 09:38:52 2013
new/usr/src/cmd/whodo/whodo.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 whodo 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  * Maintenance note:
  47  *
  48  * Much of this code is replicated in w.c.  If you're
  49  * fixing bugs here, then you should probably fix 'em there too.
  50  */

  52 #include <stdio.h>
  53 #include <string.h>
  54 #include <stdlib.h>
  55 #include <ctype.h>
  56 #include <fcntl.h>
  57 #include <time.h>
  58 #include <errno.h>
  59 #include <sys/types.h>
  60 #include <utmpx.h>
  61 #include <sys/utsname.h>
```



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

  72 /*
  73  * Use the full lengths from utmpx for user and line.
  74  */
  75 #define NMAX    (sizeof (((struct utmpx *)0)->ut_user))
  76 #define LMAX    (sizeof (((struct utmpx *)0)->ut_line))

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

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

  84 #ifdef ERR
  85 #undef ERR
  86 #endif
  87 #define ERR             (-1)

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

  98 static int      ndevs;                  /* number of configured devices */
  99 static int      maxdev;                 /* slots for configured devices */
 100 #define DNINCR  100
 101 static struct devl {                    /* device list   */
 102         char    dname[DEVNAMELEN];      /* device name   */
 103         dev_t   ddev;                   /* device number */
 104 } *devl;
_____unchanged_portion_omitted_

 121 /*
 122  *      define  hash table for struct uproc
 123  *      Hash function uses process id
 124  *      and the size of the hash table(HSIZE)
 125  *      to determine process index into the table.
 126  */
 127 static struct uproc     pr_htbl[HSIZE];

 129 static struct   uproc   *findhash(pid_t);
 130 static time_t   findidle(char *);
 131 static void     clnarglist(char *);
 132 static void     showproc(struct uproc *);
 133 static void     showtotals(struct uproc *);
 134 static void     calctotals(struct uproc *);
 135 static char     *getty(dev_t);
 136 static void     prttime(time_t, int);
 136 static void     prttime(time_t, char *);
 137 static void     prtat(time_t *);
 138 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                                         "%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)

736 /*
737  * prttime prints a time in days, hours, minutes, or seconds.
738  * The second argument is the 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".
739  */
740 static void
741 prttime(time_t tim, int width)
746 prttime(time_t tim, char *tail)
742 {
743         char value[12];
744         char *unit;

746         if (tim >= 36 * HR) {
747                 (void) snprintf(value, sizeof (value), "%d",
748                     (tim + (DAY / 2)) / (DAY));
749                 unit = dcgettext(NULL, "days", LC_TIME);
750         } else if (tim >= 36 * 60) {
751                 (void) snprintf(value, sizeof (value), "%d",
752                     (tim + (HR / 2)) / (HR));
753                 unit = dcgettext(NULL, "hours", LC_TIME);
754         } else if (tim >= 60) {
755                 (void) snprintf(value, sizeof (value), "%d",
756                     (tim + 30) / 60);
757                 unit = dcgettext(NULL, "mins", LC_TIME);
758         } else if (tim > 0) {
759                 (void) snprintf(value, sizeof (value), "%d", (int)tim);
760                 unit = dcgettext(NULL, "secs", LC_TIME);
761         } else {
762                 (void) strcpy(value, "0");
763                 unit = " ";
764         }
765         width -= 2 + strlen(value);
766         width = (width > 1) ? width : 1;
767         printf("%s %-*s ", value, width, unit);
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);
768 }


771 /*
772  * prints a locale-specific time given a pointer to a time of day
760  * prints a 12 hour time given a pointer to a time of day
773  */
774 static void
775 prtat(time_t *time)
```



```
776 {
777         struct tm       *p;

779         p = localtime(time);
780         if (now - *time <= 18 * HR) {
781                 char timestr[50];

783                 (void) strftime(timestr, sizeof (timestr),
784                     "%X", p);
785                 printf("%-11s ", timestr);
771                     dcgettext(NULL, " %l:%M""%p", LC_TIME), p);
772                 checkampm(timestr);
773                 (void) printf("%s", timestr);
786         } else if (now - *time <= 7 * DAY) {
787                 char weekdaytime[20];

789                 (void) strftime(weekdaytime, sizeof (weekdaytime),
790                     "%d %b", p);
791                 printf("%-11s ", weekdaytime);
778                     dcgettext(NULL, "%a%l%p", LC_TIME), p);
779                 checkampm(weekdaytime);
780                 (void) printf(" %s", weekdaytime);
792         } else {
793                 char monthtime[20];

795                 (void) strftime(monthtime, sizeof (monthtime),
796                     "%b %Y", p);
797                 printf("%-11s ", monthtime);
785                     dcgettext(NULL, "%e%b%y", LC_TIME), p);
786                 (void) printf(" %s", monthtime);
798         }
799 }
```

*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 }
```