```
**********************************************************
   18563 Thu Nov 14 08:20:42 2013
new/usr/src/cmd/w/w.c
2849 uptime should use locale settings for current time
**********************************************************
______unchanged_portion_omitted_

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

 125 static struct   uproc   *findhash(pid_t);
 126 static time_t   findidle(char *);
 127 static void     clnarglist(char *);
 128 static void     showtotals(struct uproc *);
 129 static void     calctotals(struct uproc *);
 130 static void     prttime(time_t, 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", LC_TIME)));
312                 } else {
313                         PRINTF((dcgettext(NULL,
314                             "User     tty           idle   what\n", LC_TIME)));
315                 }

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

323         /*
324          * loop through /proc, reading info about each process
325          * and build the parent/child tree
326          */
327         if (!(dirp = opendir(PROCDIR))) {
328                 (void) fprintf(stderr, gettext("%s: could not open %s: %s\n"),
329                     prog, PROCDIR, strerror(errno));
330                 exit(1);
331         }

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

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

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

368                         /* now we need the proc_owner privilege */
369                         (void) __priv_bracket(PRIV_ON);

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

```
373                     /* drop proc_owner privilege after open */
374                     (void) __priv_bracket(PRIV_OFF);
375
376                     if (procfd < 0)
377                             continue;
378
379                     if (read(procfd, &statinfo, sizeof (statinfo))
380                         != sizeof (statinfo)) {
381                             int err = errno;
382                             (void) close(procfd);
383                             if (err == EAGAIN)
384                                     goto retry;
385                             if (err != ENOENT)
386                                     (void) fprintf(stderr, gettext(
387                                         "%s: read() failed on %s: %s \n"),
388                                         prog, pname, strerror(err));
389                             continue;
390                     }
391                     (void) close(procfd);
392
393                     up->p_time = statinfo.pr_utime.tv_sec +
394                         statinfo.pr_stime.tv_sec;   /* seconds */
395                     up->p_ctime = statinfo.pr_cutime.tv_sec +
396                         statinfo.pr_cstime.tv_sec;
397
398                     (void) strcpy(fname, "sigact");
399
400                     /* now we need the proc_owner privilege */
401                     (void) __priv_bracket(PRIV_ON);
402
403                     procfd = open(pname, O_RDONLY);
404
405                     /* drop proc_owner privilege after open */
406                     (void) __priv_bracket(PRIV_OFF);
407
408                     if (procfd < 0)
409                             continue;
410
411                     if (read(procfd, actinfo, sizeof (actinfo))
412                         != sizeof (actinfo)) {
413                             int err = errno;
414                             (void) close(procfd);
415                             if (err == EAGAIN)
416                                     goto retry;
417                             if (err != ENOENT)
418                                     (void) fprintf(stderr, gettext(
419                                         "%s: read() failed on %s: %s \n"),
420                                         prog, pname, strerror(err));
421                             continue;
422                     }
423                     (void) close(procfd);
424
425                     up->p_igintr =
426                         actinfo[SIGINT-1].sa_handler == SIG_IGN &&
427                         actinfo[SIGQUIT-1].sa_handler == SIG_IGN;
428
429                     /*
430                      * Process args.
431                      */
432                     up->p_args[0] = 0;
433                     clnarglist(info.pr_psargs);
434                     (void) strcat(up->p_args, info.pr_psargs);
435                     if (up->p_args[0] == 0 ||
436                         up->p_args[0] == '-' && up->p_args[1] <= ' ' ||
437                         up->p_args[0] == '?') {
438                             (void) strcat(up->p_args, " (");
```



```
439                             (void) strcat(up->p_args, up->p_comm);
440                             (void) strcat(up->p_args, ")");
441                     }
442             }
443
444             /*
445              * link pgrp together in case parents go away
446              * Pgrp chain is a single linked list originating
447              * from the pgrp leader to its group member.
448              */
449             if (info.pr_pgid != info.pr_pid) {      /* not pgrp leader */
450                     pgrp = findhash(info.pr_pgid);
451                     up->p_pgrpl = pgrp->p_pgrpl;
452                     pgrp->p_pgrpl = up;
453             }
454             parent = findhash(info.pr_ppid);
455
456             /* if this is the new member, link it in */
457             if (parent->p_upid != INITPROCESS) {
458                     if (parent->p_child) {
459                             up->p_sibling = parent->p_child;
460                             up->p_child = 0;
461                     }
462                     parent->p_child = up;
463             }
464     }
465
466     /* revert to non-privileged user after opening */
467     (void) __priv_relinquish();
468
469     (void) closedir(dirp);
470     (void) time(&now);      /* get current time */
471
472     /*
473      * loop through utmpx file, printing process info
474      * about each logged in user
475      */
476     for (ut = utmpbegin; ut < utmpend; ut++) {
477             if (ut->ut_type != USER_PROCESS)
478                     continue;
479             if (sel_user && strncmp(ut->ut_name, sel_user, NMAX) != 0)
480                     continue;       /* we're looking for somebody else */
481
482             /* print login name of the user */
483             PRINTF(("%-*.*s ", LOGIN_WIDTH, NMAX, ut->ut_name));
484
485             /* print tty user is on */
486             if (lflag) {
487                     PRINTF(("%-*.*s", LINE_WIDTH, LMAX, ut->ut_line));
488             } else {
489                     if (ut->ut_line[0] == 'p' && ut->ut_line[1] == 't' &&
490                         ut->ut_line[2] == 's' && ut->ut_line[3] == '/') {
491                             PRINTF(("%-*.3s", LMAX, &ut->ut_line[4]));
492                     } else {
493                             PRINTF(("%-*.*s", LINE_WIDTH, LMAX,
494                                 ut->ut_line));
495                     }
496             }
497
498             /* print when the user logged in */
499             if (lflag) {
500                     time_t tim = ut->ut_xtime;
501                     prtat(&tim);
502             }
503
504             /* print idle time */
```

```
505                 idle = findidle(ut->ut_line);
506                 if (idle >= 36 * 60) {
507                         PRINTF((dcgettext(NULL, "%2ddays ", LC_TIME),
508                             (idle + 12 * 60) / (24 * 60)));
509                 } else
510                         prttime(idle, " ");
511                 showtotals(findhash(ut->ut_pid));
512         }
513         if (fclose(stdout) == EOF) {
514                 perror((gettext("%s: fclose failed"), prog));
515                 exit(1);
516         }
517         return (0);
518 }
```

*unchanged_portion_omitted_*

```
666 /*
667  * prints a 12 hour time given a pointer to a time of day
668  */
669 static void
670 prtat(time_t *time)
671 {
672         struct tm       *p;

674         p = localtime(time);
675         if (now - *time <= 18 * HR) {
676                 char timestr[50];
677                 (void) strftime(timestr, sizeof (timestr),
678                     " %R  ", p);
679                     dcgettext(NULL, "%l:%M""%p", LC_TIME), p);
680                 checkampm(timestr);
679                 PRINTF((" %s", timestr));
680         } else if (now - *time <= 7 * DAY) {
681                 char weekdaytime[20];

683                 (void) strftime(weekdaytime, sizeof (weekdaytime),
684                     "%a%H  ", p);
686                     dcgettext(NULL, "%a%l%p", LC_TIME), p);
687                 checkampm(weekdaytime);
685                 PRINTF((" %s", weekdaytime));
686         } else {
687                 char monthtime[20];

689                 (void) strftime(monthtime, sizeof (monthtime),
690                     "%e%b%y", p);
693                     dcgettext(NULL, "%e%b%y", LC_TIME), p);
691                 PRINTF((" %s", monthtime));
692         }
693 }
```

*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 }
```

```
**********************************************************
   20571 Thu Nov 14 08:20:42 2013
new/usr/src/cmd/whodo/whodo.c
2849 uptime should use locale settings for current time
**********************************************************
______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, char *);
 137 static void     prtat(time_t *);
 138 static void     checkampm(char *);

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

 153 #if SIGQUIT > SIGINT
 154 #define ACTSIZE SIGQUIT
 155 #else
 156 #define ACTSIZE SIGINT
 157 #endif

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



```
 179         int             i;
 180         int             days, hrs, mins;
 181         int             entries;

 183         /*
 184          * This program needs the proc_owner privilege
 185          */
 186         (void) __init_suid_priv(PU_CLEARLIMITSET, PRIV_PROC_OWNER,
 187             (char *)NULL);

 189         (void) setlocale(LC_ALL, "");
 190 #if !defined(TEXT_DOMAIN)
 191 #define TEXT_DOMAIN "SYS_TEST"
 192 #endif
 193         (void) textdomain(TEXT_DOMAIN);

 195         prog = argv[0];

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

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

 206                                 case 'l':
 207                                         lflag++;
 208                                         break;

 210                                 default:
 211                                         (void) printf(gettext(
 212                                             "usage: %s [ -hl ] [ user ]\n"),
 213                                             prog);
 214                                         exit(1);
 215                                 }
 216                         }
 217                 } else {
 218                         if (!isalnum(argv[1][0]) || argc > 2) {
 219                                 (void) printf(gettext(
 220                                     "usage: %s [ -hl ] [ user ]\n"), prog);
 221                                 exit(1);
 222                         } else
 223                                 sel_user = argv[1];
 224                 }
 225                 argc--; argv++;
 226         }

 228         /*
 229          * read the UTMPX_FILE (contains information about
 230          * each logged in user)
 231          */
 232         if (stat(UTMPX_FILE, &sbuf) == ERR) {
 233                 (void) fprintf(stderr, gettext("%s: stat error of %s: %s\n"),
 234                     prog, UTMPX_FILE, strerror(errno));
 235                 exit(1);
 236         }
 237         entries = sbuf.st_size / sizeof (struct futmpx);
 238         size = sizeof (struct utmpx) * entries;

 240         if ((ut = malloc(size)) == NULL) {
 241                 (void) fprintf(stderr, gettext("%s: malloc error of %s: %s\n"),
 242                     prog, UTMPX_FILE, strerror(errno));
 243                 exit(1);
 244         }
```

```
246         (void) utmpxname(UTMPX_FILE);

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

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

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

259         if (header) {   /* print a header */
260                 if (lflag) {    /* w command format header */
261                         prtat(&now);
262                         for (ut = utmpbegin; ut < utmpend; ut++) {
263                                 if (ut->ut_type == USER_PROCESS) {
264                                         nusers++;
265                                 } else if (ut->ut_type == BOOT_TIME) {
266                                         uptime = now - ut->ut_xtime;
267                                         uptime += 30;
268                                         days = uptime / (60*60*24);
269                                         uptime %= (60*60*24);
270                                         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", LC_TIME));
286                 } else {        /* standard whodo header */
287                         char date_buf[100];

289                         /*
290                          * print current time and date
291                          */
292                         (void) strftime(date_buf, sizeof (date_buf),
293                             "%+", localtime(&now));
294                             dcgettext(NULL, "%C", LC_TIME), localtime(&now));
294                         (void) printf("%s\n", date_buf);

296                         /*
297                          * print system name
298                          */
299                         (void) uname(&uts);
300                         (void) printf("%s\n", uts.nodename);
301                 }
302         }

304         /*
305          * loop through /proc, reading info about each process
306          * and build the parent/child tree
307          */
308         if (!(dirp = opendir(PROCDIR))) {
309                 (void) fprintf(stderr, gettext("%s: could not open %s: %s\n"),
```



```
310                     prog, PROCDIR, strerror(errno));
311                 exit(1);
312         }

314         while ((dp = readdir(dirp)) != NULL) {
315                 if (dp->d_name[0] == '.')
316                         continue;
317 retry:
318                 (void) snprintf(pname, sizeof (pname),
319                     "%s/%s/", PROCDIR, dp->d_name);
320                 fname = pname + strlen(pname);
321                 (void) strcpy(fname, "psinfo");
322                 if ((procfd = open(pname, O_RDONLY)) < 0)
323                         continue;
324                 if (read(procfd, &info, sizeof (info)) != sizeof (info)) {
325                         int err = errno;
326                         (void) close(procfd);
327                         if (err == EAGAIN)
328                                 goto retry;
329                         if (err != ENOENT)
330                                 (void) fprintf(stderr, gettext(
331                                     "%s: read() failed on %s: %s\n"),
332                                     prog, pname, strerror(err));
333                         continue;
334                 }
335                 (void) close(procfd);

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

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

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

353                         procfd = open(pname, O_RDONLY);

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

358                         if (procfd  < 0)
359                                 continue;

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

375                         up->p_time = statinfo.pr_utime.tv_sec +
```

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

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

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

385                 procfd = open(pname, O_RDONLY);

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

390                 if (procfd  < 0)
391                         continue;
392                 if (read(procfd, actinfo, sizeof (actinfo))
393                     != sizeof (actinfo)) {
394                         int err = errno;
395                         (void) close(procfd);
396                         if (err == EAGAIN)
397                                 goto retry;
398                         if (err != ENOENT)
399                                 (void) fprintf(stderr, gettext(
400                                     "%s: read() failed on %s: %s \n"),
401                                     prog, pname, strerror(err));
402                         continue;
403                 }
404                 (void) close(procfd);

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

410                 up->p_args[0] = 0;

412                 /*
413                  * Process args if there's a chance we'll print it.
414                  */
415                 if (lflag) { /* w command needs args */
416                         clnarglist(info.pr_psargs);
417                         (void) strcpy(up->p_args, info.pr_psargs);
418                         if (up->p_args[0] == 0 ||
419                             up->p_args[0] == '-' &&
420                             up->p_args[1] <= ' ' ||
421                             up->p_args[0] == '?') {
422                                 (void) strcat(up->p_args, " (");
423                                 (void) strcat(up->p_args, up->p_comm);
424                                 (void) strcat(up->p_args, ")");
425                         }
426                 }

428         }

430         /*
431          * link pgrp together in case parents go away
432          * Pgrp chain is a single linked list originating
433          * from the pgrp leader to its group member.
434          */
435         if (info.pr_pgid != info.pr_pid) {      /* not pgrp leader */
436                 pgrp = findhash(info.pr_pgid);
437                 up->p_pgrplink = pgrp->p_pgrplink;
438                 pgrp->p_pgrplink = up;
439         }
440         parent = findhash(info.pr_ppid);
```



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

451         }

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

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

459         /*
460          * loop through utmpx file, printing process info
461          * about each logged in user
462          */
463         for (ut = utmpbegin; ut < utmpend; ut++) {
464                 time_t tim;

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

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

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

483                         /* print idle time */
484                         idle = findidle(ut->ut_line);
485                         if (idle >= 36 * 60)
486                                 (void) printf(dcgettext(NULL, "%2ddays ",
487                                     LC_TIME), (idle + 12 * 60) / (24 * 60));
488                         else
489                                 prttime(idle, " ");
490                         showtotals(findhash((pid_t)ut->ut_pid));
491                 } else {        /* standard whodo format */
492                         tim = ut->ut_xtime;
493                         tm = localtime(&tim);
494                         (void) printf("\n%-*.*s %-*.*s %2.1d:%2.2d\n",
495                             LINE_WIDTH, (int)LMAX, ut->ut_line,
496                             LOGIN_WIDTH, (int)NMAX, ut->ut_name, tm->tm_hour,
497                             tm->tm_min);
498                         showproc(findhash((pid_t)ut->ut_pid));
499                 }
500         }

502         return (0);
503 }
```

_____***unchanged_portion_omitted_***

```
758 /*
```

```
 * prints a 12 hour time given a pointer to a time of day
 */
static void
prtat(time_t *time)
{
        struct tm *p;

        p = localtime(time);
        if (now - *time <= 18 * HR) {
                char timestr[50];
                (void) strftime(timestr, sizeof (timestr),
                    " %R  ", p);
                    dcgettext(NULL, " %l:%M""%p", LC_TIME), p);
                checkampm(timestr);
                (void) printf("%s", timestr);
        } else if (now - *time <= 7 * DAY) {
                char weekdaytime[20];

                (void) strftime(weekdaytime, sizeof (weekdaytime),
                    "%a%H  ", p);
                    dcgettext(NULL, "%a%l%p", LC_TIME), p);
                checkampm(weekdaytime);
                (void) printf(" %s", weekdaytime);
        } else {
                char monthtime[20];

                (void) strftime(monthtime, sizeof (monthtime),
                    "%e%b%y", p);
                    dcgettext(NULL, "%e%b%y", LC_TIME), p);
                (void) printf(" %s", monthtime);
        }
}
```

unchanged_portion_omitted_

```
/* replaces all occurences of AM/PM with am/pm */
static void
checkampm(char *str)
{
        char *ampm;
        while ((ampm = strstr(str, "AM")) != NULL ||
            (ampm = strstr(str, "PM")) != NULL) {
                *ampm = tolower(*ampm);
                *(ampm+1) = tolower(*(ampm+1));
        }
}
```