

```

*****
9191 Wed Apr 3 09:33:10 2013
new/usr/src/cmd/grpck/grpck.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 2007 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      */

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

32 #include <sys/param.h>
33 #include <sys/types.h>
34 #include <unistd.h>
35 #include <stdlib.h>
36 #include <stdio.h>
37 #include <string.h>
38 #include <ctype.h>
39 #include <pwd.h>
40 #include <errno.h>
41 #include <locale.h>
42 #include <limits.h>

44 #define BADLINE "Too many/few fields"
45 #define TOOLONG "Line too long"
46 #define NONAME "No group name"
47 #define BADNAME "Bad character(s) in group name"
48 #define BADGID "Invalid GID"
49 #define NULLNAME "Null login name"
50 #define NOTFOUND "Logname not found in password file"
51 #define DUPNAME "Duplicate logname entry"
52 #define DUPNAME2 "Duplicate logname entry (gid first occurs in passwd entry)"
53 #define NOMEM "Out of memory"
54 #define NGROUPS "Maximum groups exceeded for logname "
55 #define BLANKLINE "Blank line detected. Please remove line"
56 #define LONGNAME "Group name too long"

58 #ifndef LOGNAME_MAX_ILLUMOS

```

```

59 #define _LOGNAME_MAX LOGNAME_MAX_ILLUMOS
60 #else /* LOGNAME_MAX_ILLUMOS */
61 #define _LOGNAME_MAX LOGNAME_MAX
62 #endif /* LOGNAME_MAX_ILLUMOS */

64 int eflag, badchar, baddigit, badlognam, colons, len;
65 static int longnam = 0;
66 int code;

68 #define MYBUFSIZE (LINE_MAX) /* max line length including newline and null */
69 #define NUM_COLONS 3

71 char *buf;
72 char *nptr;
73 char *cptr;
74 FILE *fptr;
75 gid_t gid;
76 void error(char *msg);

78 struct group {
79     struct group *nxt;
80     int cnt;
81     gid_t grp;
82 };

unchanged_portion_omitted

103 int
104 main(int argc, char *argv[])
105 {
106     struct passwd *pwp;
107     struct node *root = NULL;
108     struct node *t;
109     struct group *gp;
110     int ngroups_max;
111     int ngroups = 0;
112     int listlen;
113     int i;
114     int lineno = 0;
115     char *buf_off, *tmpbuf;
116     int delim[NUM_COLONS + 1], buf_len, bufsize;

118     (void) setlocale(LC_ALL, "");

120 #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
121 #define TEXT_DOMAIN "SYS_TEST"
122 #endif
123     (void) textdomain(TEXT_DOMAIN);

125     code = 0;
126     ngroups_max = sysconf(_SC_NGROUPS_MAX);

128     if (argc == 1)
129         argv[1] = "/etc/group";
130     else if (argc != 2) {
131         fprintf(stderr, gettext("usage: %s filename\n"), *argv);
132         exit(1);
133     }

135     if ((fptr = fopen(argv[1], "r")) == NULL) {
136         fprintf(stderr, gettext("cannot open file %s: %s\n"), argv[1],
137             strerror(errno));
138         exit(1);
139     }

141 #ifdef ORIG_SVR4
142     while ((pwp = getpwent()) != NULL) {

```

```

143         t = (struct node *)emalloc(sizeof (*t) + strlen(pwp->pw_name));
144         t->next = root;
145         root = t;
146         strcpy(t->user, pwp->pw_name);
147         t->ngroups = 1;
148         if (!ngroups_max)
149             t->groups = NULL;
150         else {
151             t->groups = (struct group *)
152                 emalloc(sizeof (struct group));
153             t->groups->grp = pwp->pw_gid;
154             t->groups->cnt = 1;
155             t->groups->nxt = NULL;
156         }
157     }
158 #endif

160 bufsize = MYBUFSIZE;
161 if ((buf = malloc(bufsize)) == NULL) {
162     (void) fprintf(stderr, gettext(NOMEM));
163     exit(1);
164 }
165 while (!feof(fp) && !ferror(fp)) {
166     buf_len = 0;
167     buf_off = buf;
168     while (fgets(buf_off, (bufsize - buf_len), fp) != NULL) {
169         buf_len += strlen(buf_off);
170         if (buf[buf_len - 1] == '\n' || feof(fp))
171             break;
172         tmpbuf = realloc(buf, (bufsize + MYBUFSIZE));
173         if (tmpbuf == NULL) {
174             (void) fprintf(stderr, gettext(NOMEM));
175             exit(1);
176         }
177         bufsize += MYBUFSIZE;
178         buf = tmpbuf;
179         buf_off = buf + buf_len;
180     }
181     if (buf_len == 0)
182         continue;

184     /* Report error to be consistent with libc */
185     if ((buf_len + 1) > LINE_MAX)
186         error(TOOLONG);

188     lineno++;
189     if (buf[0] == '\n') /* blank lines are ignored */
190     {
191         code = 1; /* exit with error code = 1 */
192         eflag = 0; /* force print of "blank" line */
193         fprintf(stderr, "\n%s %d\n", gettext(BLANKLINE),
194             lineno);
195         continue;
196     }

198     if (buf[buf_len - 1] == '\n') {
199         if ((tmpbuf = strdup(buf)) == NULL) {
200             (void) fprintf(stderr, gettext(NOMEM));
201             exit(1);
202         }
203         tmpbuf[buf_len - 1] = ',';
204     } else {
205         if ((tmpbuf = malloc(buf_len + 2)) == NULL) {
206             (void) fprintf(stderr, gettext(NOMEM));
207             exit(1);
208         }

```

```

209         (void) strcpy(tmpbuf, buf);
210         tmpbuf[buf_len++] = ',';
211         tmpbuf[buf_len] = '\0';
212     }

214     colons = 0;
215     eflag = 0;
216     badchar = 0;
217     baddigit = 0;
218     badlognam = 0;
219     gid = 0;

221     ngroups++; /* Increment number of groups found */
222     /* Check that entry is not a nameservice redirection */

224     if (buf[0] == '+' || buf[0] == '-') {
225         /*
226          * Should set flag here to allow special case checking
227          * in the rest of the code,
228          * but for now, we'll just ignore this entry.
229          */
230         free(tmpbuf);
231         continue;
232     }

234     /* Check number of fields */

236     for (i = 0; buf[i] != NULL; i++) {
237         if (buf[i] == ':') {
238             delim[colons] = i;
239             if (++colons > NUM_COLONS)
240                 break;
241         }
242     }
243     if (colons != NUM_COLONS) {
244         error(BADLINE);
245         free(tmpbuf);
246         continue;
247     }

249     /* check to see that group name is at least 1 character */
250     /* and that all characters are lowercase or digits. */

252     if (buf[0] == ':')
253         error(NONAME);
254     else {
255         for (i = 0; buf[i] != ':'; i++) {
256             if (i >= LOGNAME_MAX)
257                 if (i >= LOGNAME_MAX)
258                     longnam++;
259             if (!(islower(buf[i]) || isdigit(buf[i])))
260                 badchar++;
261         }
262         if (longnam > 0)
263             error(LONGNAME);
264         if (badchar > 0)
265             error(BADNAME);
266     }

267     /* check that GID is numeric and <= 31 bits */

269     len = (delim[2] - delim[1]) - 1;

271     if (len > 10 || len < 1)
272         error(BADGID);
273     else {

```

```

274         for (i = (delim[1]+1); i < delim[2]; i++) {
275             if (! (isdigit(buf[i])))
276                 baddigit++;
277             else if (baddigit == 0)
278                 gid = gid * 10 + (gid_t)(buf[i] - '0');
279             /* converts ascii GID to decimal */
280         }
281         if (baddigit > 0)
282             error(BADGID);
283         else if (gid > (gid_t)MAXUID)
284             error(BADGID);
285     }
286
287     /* check that logname appears in the passwd file */
288
289     nptr = &tmpbuf[delim[2]];
290     nptr++;
291
292     listlen = strlen(nptr) - 1;
293
294     while ((cptr = strchr(nptr, ',')) != NULL) {
295         *cptr = NULL;
296         if (*nptr == NULL) {
297             if (listlen)
298                 error(NULLNAME);
299             nptr++;
300             continue;
301         }
302
303         for (t = root; t != NULL; t = t->next) {
304             if (strcmp(t->user, nptr) == 0)
305                 break;
306         }
307         if (t == NULL) {
308 #ifndef ORIG_SVR4
309             /*
310              * User entry not found, so check if in
311              * password file
312              */
313             struct passwd *pwp;
314
315             if ((pwp = getpwnam(nptr)) == NULL) {
316 #endif
317                 badlognam++;
318                 error(NOTFOUND);
319                 goto getnext;
320 #ifndef ORIG_SVR4
321             }
322
323             /* Username found, so add entry to user-list */
324             t = (struct node *)
325                 emalloc(sizeof (*t) + strlen(nptr));
326             t->next = root;
327             root = t;
328             strcpy(t->user, nptr);
329             t->ngroups = 1;
330             if (!ngroups_max)
331                 t->groups = NULL;
332             else {
333                 t->groups = (struct group *)
334                     emalloc(sizeof (struct group));
335                 t->groups->grp = pwp->pw_gid;
336                 t->groups->cnt = 1;
337                 t->groups->nxt = NULL;
338             }
339         }

```

```

340 #endif
341         if (!ngroups_max)
342             goto getnext;
343
344         t->ngroups++;
345
346         /*
347          * check for duplicate logname in group
348          */
349
350         for (gp = t->groups; gp != NULL; gp = gp->nxt) {
351             if (gid == gp->grp) {
352                 if (gp->cnt++ == 1) {
353                     badlognam++;
354                     if (gp->nxt == NULL)
355                         error(DUPNAME2);
356                     else
357                         error(DUPNAME);
358                 }
359                 goto getnext;
360             }
361         }
362
363         gp = (struct group *)emalloc(sizeof (struct group));
364         gp->grp = gid;
365         gp->cnt = 1;
366         gp->nxt = t->groups;
367         t->groups = gp;
368 getnext:
369         nptr = ++cptr;
370     }
371     free(tmpbuf);
372
373     if (ngroups == 0) {
374         fprintf(stderr, gettext("Group file '%s' is empty\n"), argv[1]);
375         code = 1;
376     }
377
378     if (ngroups_max) {
379         for (t = root; t != NULL; t = t->next) {
380             if (t->ngroups > ngroups_max) {
381                 fprintf(stderr, "\n\n%s (%d)\n",
382                     NGROUPS, t->ngroups);
383                 code = 1;
384             }
385         }
386     }
387     return (code);
388 }
389 }

```

_____unchanged_portion_omitted_____

```

*****
115298 Wed Apr 3 09:33:10 2013
new/usr/src/cmd/init/init.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 ("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 */

22 /*
23 * Copyright (c) 2013 Gary Mills
24 *
25 * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
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 * init(1M) is the general process spawning program. Its primary job is to
43 * start and restart svc.startd for smf(5). For backwards-compatibility it also
44 * spawns and respawns processes according to /etc/inittab and the current
45 * run-level. It reads /etc/default/inittab for general configuration.
46 *
47 * To change run-levels the system administrator runs init from the command
48 * line with a level name. init signals svc.startd via libscf and directs the
49 * zone's init (pid 1 in the global zone) what to do by sending it a signal;
50 * these signal numbers are commonly referred to in the code as 'states'. Valid
51 * run-levels are [sS0123456]. Additionally, init can be given directives
52 * [qQabc], which indicate actions to be taken pertaining to /etc/inittab.
53 *
54 * When init processes inittab entries, it finds processes that are to be
55 * spawned at various run-levels. inittab contains the set of the levels for
56 * which each inittab entry is valid.
57 *
58 * State File and Restartability
59 * Premature exit by init(1M) is handled as a special case by the kernel:
60 * init(1M) will be immediately re-executed, retaining its original PID. (PID

```

```

61 * 1 in the global zone.) To track the processes it has previously spawned,
62 * as well as other mutable state, init(1M) regularly updates a state file
63 * such that its subsequent invocations have knowledge of its various
64 * dependent processes and duties.
65 *
66 * Process Contracts
67 * We start svc.startd(1M) in a contract and transfer inherited contracts when
68 * restarting it. Everything else is started using the legacy contract
69 * template, and the created contracts are abandoned when they become empty.
70 *
71 * utmpx Entry Handling
72 * Because init(1M) no longer governs the startup process, its knowledge of
73 * when utmpx becomes writable is indirect. However, spawned processes
74 * expect to be constructed with valid utmpx entries. As a result, attempts
75 * to write normal entries will be retried until successful.
76 *
77 * Maintenance Mode
78 * In certain failure scenarios, init(1M) will enter a maintenance mode, in
79 * which it invokes sulogin(1M) to allow the operator an opportunity to
80 * repair the system. Normally, this operation is performed as a
81 * fork(2)-exec(2)-waitpid(3C) sequence with the parent waiting for repair or
82 * diagnosis to be completed. In the cases that fork(2) requests themselves
83 * fail, init(1M) will directly execute sulogin(1M), and allow the kernel to
84 * restart init(1M) on exit from the operator session.
85 *
86 * One scenario where init(1M) enters its maintenance mode is when
87 * svc.startd(1M) begins to fail rapidly, defined as when the average time
88 * between recent failures drops below a given threshold.
89 */

91 #include <sys/contract/process.h>
92 #include <sys/ctfs.h>
93 #include <sys/stat.h>
94 #include <sys/statvfs.h>
95 #include <sys/stropts.h>
96 #include <sys/systeminfo.h>
97 #include <sys/time.h>
98 #include <sys/termios.h>
99 #include <sys/tty.h>
100 #include <sys/types.h>
101 #include <sys/utsname.h>

103 #include <bsm/adt_event.h>
104 #include <bsm/libbsm.h>
105 #include <security/pam_appl.h>

107 #include <assert.h>
108 #include <ctype.h>
109 #include <dirent.h>
110 #include <errno.h>
111 #include <fcntl.h>
112 #include <libcontract.h>
113 #include <libcontract_priv.h>
114 #include <libintl.h>
115 #include <libscf.h>
116 #include <libscf_priv.h>
117 #include <poll.h>
118 #include <procfs.h>
119 #include <signal.h>
120 #include <stdarg.h>
121 #include <stdio.h>
122 #include <stdio_ext.h>
123 #include <stdlib.h>
124 #include <string.h>
125 #include <strings.h>
126 #include <syslog.h>

```

```

127 #include <time.h>
128 #include <ulimit.h>
129 #include <unistd.h>
130 #include <utmpx.h>
131 #include <wait.h>
132 #include <zone.h>
133 #include <ucontext.h>

135 #undef sleep

137 #define fiocctl(p, sptr, cmd)   ioctl(fileno(p), sptr, cmd)
138 #define min(a, b)              ((a) < (b)) ? (a) : (b)

140 #define TRUE    1
141 #define FALSE  0
142 #define FAILURE -1

144 #define UT_USER_SZ    32    /* Size of a utmpx ut_user field */
145 #define UT_LINE_SZ    32    /* Size of a utmpx ut_line field */

147 /*
148 * SLEEPTIME    The number of seconds "init" sleeps between wakeups if
149 *              nothing else requires this "init" wakeup.
150 */
151 #define SLEEPTIME    (5 * 60)

153 /*
154 * MAXCMDL    The maximum length of a command string in inittab.
155 */
156 #define MAXCMDL    512

158 /*
159 * EXEC    The length of the prefix string added to all comamnds
160 *          found in inittab.
161 */
162 #define EXEC    (sizeof ("exec ") - 1)

164 /*
165 * TWARN    The amount of time between warning signal, SIGTERM,
166 *          and the fatal kill signal, SIGKILL.
167 */
168 #define TWARN    5

170 #define id_eq(x, y)    ((x[0] == y[0] && x[1] == y[1] && x[2] == y[2] &&
171 x[3] == y[3]) ? TRUE : FALSE)

173 /*
174 * The kernel's default umask is 022 these days; since some processes inherit
175 * their umask from init, init will set it from CMASK in /etc/default/init.
176 * init gets the default umask from the kernel, it sets it to 022 whenever
177 * it wants to create a file and reverts to CMASK afterwards.
178 */

180 static int cmask;

182 /*
183 * The following definitions, concluding with the 'lvls' array, provide a
184 * common mapping between level-name (like 'S'), signal number (state),
185 * run-level mask, and specific properties associated with a run-level.
186 * This array should be accessed using the routines lvlname_to_state(),
187 * lvlname_to_mask(), state_to_mask(), and state_to_flags().
188 */

190 /*
191 * Correspondence of signals to init actions.
192 */

```

```

193 #define LVLQ          SIGHUP
194 #define LVL0          SIGINT
195 #define LVL1          SIGQUIT
196 #define LVL2          SIGILL
197 #define LVL3          SIGTRAP
198 #define LVL4          SIGIOT
199 #define LVL5          SIGEMT
200 #define LVL6          SIGFPE
201 #define SINGLE_USER  SIGBUS
202 #define LVLa          SIGSEGV
203 #define LVLb          SIGSYS
204 #define LVLc          SIGPIPE

206 /*
207 * Bit Mask for each level.  Used to determine legal levels.
208 */
209 #define MASK0    0x0001
210 #define MASK1    0x0002
211 #define MASK2    0x0004
212 #define MASK3    0x0008
213 #define MASK4    0x0010
214 #define MASK5    0x0020
215 #define MASK6    0x0040
216 #define MASKSU   0x0080
217 #define MASKa    0x0100
218 #define MASKb    0x0200
219 #define MASKc    0x0400

221 #define MASK_NUMERIC (MASK0 | MASK1 | MASK2 | MASK3 | MASK4 | MASK5 | MASK6)
222 #define MASK_abc (MASKa | MASKb | MASKc)

224 /*
225 * Flags to indicate properties of various states.
226 */
227 #define LSEL_RUNLEVEL    0x0001 /* runlevels you can transition to */

229 typedef struct lvl {
230     int    lvl_state;
231     int    lvl_mask;
232     char   lvl_name;
233     int    lvl_flags;
234 } lvl_t;
    unchanged portion omitted

1506 /*
1507 * getcmd() parses lines from inittab.  Each time it finds a command line
1508 * it will return TRUE as well as fill the passed CMD_LINE structure and
1509 * the shell command string.  When the end of inittab is reached, FALSE
1510 * is returned inittab is automatically opened if it is not currently open
1511 * and is closed when the end of the file is reached.
1512 */
1513 static FILE *fp_inittab = NULL;

1515 static int
1516 getcmd(struct CMD_LINE *cmd, char *shcmd)
1517 {
1518     char   *ptr;
1519     int    c, lastc, state;
1520     char   *ptr1;
1521     int    answer, i, proceed;
1522     struct stat sbuff;
1523     static char *actions[] = {
1524         "off", "respawn", "ondemand", "once", "wait", "boot",
1525         "bootwait", "powerfail", "powerwait", "initdefault",
1526         "sysinit",
1527     };

```

```

1528 static short act_masks[] = {
1529     M_OFF, M_RESPAWN, M_ONDEMAND, M_ONCE, M_WAIT, M_BOOT,
1530     M_BOOTWAIT, M_PF, M_PWAIT, M_INITDEFAULT, M_SYSINIT,
1531 };
1532 /*
1533  * Only these actions will be allowed for entries which
1534  * are specified for single-user mode.
1535  */
1536 short su_acts = M_INITDEFAULT | M_PF | M_PWAIT | M_WAIT;

1538 if (fp_inittab == NULL) {
1539     /*
1540      * Before attempting to open inittab we stat it to make
1541      * sure it currently exists and is not empty. We try
1542      * several times because someone may have temporarily
1543      * unlinked or truncated the file.
1544      */
1545     for (i = 0; i < 3; i++) {
1546         if (stat(INITTAB, &sbuf) == -1) {
1547             if (i == 2) {
1548                 console(B_TRUE,
1549                     "Cannot stat %s, errno: %d\n",
1550                     INITTAB, errno);
1551                 return (FAILURE);
1552             } else {
1553                 timer(3);
1554             }
1555         } else if (sbuf.st_size < 10) {
1556             if (i == 2) {
1557                 console(B_TRUE,
1558                     "%s truncated or corrupted\n",
1559                     INITTAB);
1560                 return (FAILURE);
1561             } else {
1562                 timer(3);
1563             }
1564         } else {
1565             break;
1566         }
1567     }

1569     /*
1570      * If unable to open inittab, print error message and
1571      * return FAILURE to caller.
1572      */
1573     if ((fp_inittab = fopen(INITTAB, "r")) == NULL) {
1574         console(B_TRUE, "Cannot open %s errno: %d\n", INITTAB,
1575             errno);
1576         return (FAILURE);
1577     }
1578 }

1580 /*
1581  * Keep getting commands from inittab until you find a
1582  * good one or run out of file.
1583  */
1584 for (answer = FALSE; answer == FALSE; ) {
1585     /*
1586      * Zero out the cmd itself before trying next line.
1587      */
1588     bzero(cmd, sizeof (struct CMD_LINE));

1590     /*
1591      * Read in lines of inittab, parsing at colons, until a line is
1592      * read in which doesn't end with a backslash. Do not start if
1593      * the first character read is an EOF. Note that this means

```

```

1594     * that lines which don't end in a newline are still processed,
1595     * since the "for" will terminate normally once started,
1596     * regardless of whether line terminates with a newline or EOF.
1597     */
1598     state = FAILURE;
1599     if ((c = fgetc(fp_inittab)) == EOF) {
1600         answer = FALSE;
1601         (void) fclose(fp_inittab);
1602         fp_inittab = NULL;
1603         break;
1604     }

1606     for (proceed = TRUE, ptr = shcmd, state = ID, lastc = '\0';
1607         proceed && c != EOF;
1608         lastc = c, c = fgetc(fp_inittab)) {
1609         /* If we're not in the FAILURE state and haven't */
1610         /* yet reached the shell command field, process */
1611         /* the line, otherwise just look for a real end */
1612         /* of line. */
1613         if (state != FAILURE && state != COMMAND) {
1614             /*
1615              * Squeeze out spaces and tabs.
1616              */
1617             if (c == ' ' || c == '\t')
1618                 continue;

1620             /*
1621              * Ignore characters in a comment, except for the \n.
1622              */
1623             if (state == COMMENT) {
1624                 if (c == '\n') {
1625                     lastc = ' ';
1626                     break;
1627                 } else {
1628                     continue;
1629                 }
1630             }

1632             /*
1633              * Detect comments (lines whose first non-whitespace
1634              * character is '#') by checking that we're at the
1635              * beginning of a line, have seen a '#', and haven't
1636              * yet accumulated any characters.
1637              */
1638             if (state == ID && c == '#' && ptr == shcmd) {
1639                 state = COMMENT;
1640                 continue;
1641             }

1643             /*
1644              * If the character is a ':', then check the
1645              * previous field for correctness and advance
1646              * to the next field.
1647              */
1648             if (c == ':') {
1649                 switch (state) {

1651                     case ID :
1652                         /*
1653                          * Check to see that there are only
1654                          * 1 to 4 characters for the id.
1655                          */
1656                         if ((i = ptr - shcmd) < 1 || i > 4) {
1657                             state = FAILURE;
1658                         } else {
1659                             bcopy(shcmd, &cmd->c_id[0], i);

```

```

1660         ptr = shcmd;
1661         state = LEVELS;
1662     }
1663     break;

1665     case LEVELS :
1666     /*
1667      * Build a mask for all the levels for
1668      * which this command will be legal.
1669      */
1670     for (cmd->c_levels = 0, ptr1 = shcmd;
1671          ptr1 < ptr; ptr1++) {
1672         int mask;
1673         if (lvlname_to_mask(*ptr1,
1674                             &mask) == -1) {
1675             state = FAILURE;
1676             break;
1677         }
1678         cmd->c_levels |= mask;
1679     }
1680     if (state != FAILURE) {
1681         state = ACTION;
1682         ptr = shcmd; /* Reset the buffer */
1683     }
1684     break;

1686     case ACTION :
1687     /*
1688      * Null terminate the string in shcmd buffer and
1689      * then try to match against legal actions. If
1690      * the field is of length 0, then the default of
1691      * "RESPAWN" is used if the id is numeric,
1692      * otherwise the default is "OFF".
1693      */
1694     if (ptr == shcmd) {
1695         if (isdigit(cmd->c_id[0]) &&
1696             (cmd->c_id[1] == '\0' ||
1697              isdigit(cmd->c_id[1]) &&
1698              (cmd->c_id[2] == '\0' ||
1699               isdigit(cmd->c_id[2]) &&
1700                (cmd->c_id[3] == '\0' ||
1701                 isdigit(cmd->c_id[3]))))
1702             cmd->c_action = M_RESPAWN;
1703         else
1704             cmd->c_action = M_OFF;
1705     } else {
1706         for (cmd->c_action = 0, i = 0,
1707              *ptr = '\0';
1708              i <
1709              sizeof (actions)/sizeof (char *);
1710              for (cmd->c_action = 0, i = 0, *ptr = '\0';
1711                  i < sizeof (actions)/sizeof (char *);
1712                  i++) {
1713             if (strcmp(shcmd, actions[i]) == 0) {
1714                 if ((cmd->c_levels & MASKSU) &&
1715                     !(act_masks[i] & su_acts))
1716                     cmd->c_action = 0;
1717                 else
1718                     cmd->c_action =
1719                         act_masks[i];
1720                 break;
1721             }
1722         }
1723     }

```

```

1723     /*
1724      * If the action didn't match any legal action,
1725      * set state to FAILURE.
1726      */
1727     if (cmd->c_action == 0) {
1728         state = FAILURE;
1729     } else {
1730         state = COMMAND;
1731         (void) strcpy(shcmd, "exec ");
1732     }
1733     ptr = shcmd + EXEC;
1734     break;
1735 }
1736 }
1737 }
1738 }

1740 /* If the character is a '\n', then this is the end of a */
1741 /* line. If the '\n' wasn't preceded by a backslash, */
1742 /* it is also the end of an inittab command. If it was */
1743 /* preceded by a backslash then the next line is a */
1744 /* continuation. Note that the continuation '\n' falls */
1745 /* through and is treated like other characters and is */
1746 /* stored in the shell command line. */
1747 if (c == '\n' && lastc != '\\') {
1748     proceed = FALSE;
1749     *ptr = '\0';
1750     break;
1751 }

1753 /* For all other characters just stuff them into the */
1754 /* command as long as there aren't too many of them. */
1755 /* Make sure there is room for a terminating '\0' also. */
1756 if (ptr >= shcmd + MAXCMDL - 1)
1757     state = FAILURE;
1758 else
1759     *ptr++ = (char)c;

1761 /* If the character we just stored was a quoted */
1762 /* backslash, then change "c" to '\0', so that this */
1763 /* backslash will not cause a subsequent '\n' to appear */
1764 /* quoted. In otherwords '\ ' '\n' '\n' is the real end */
1765 /* of a command, while '\ ' '\n' is a continuation. */
1766 if (c == '\\ ' && lastc == '\\ ')
1767     c = '\0';
1768 }

1770 /*
1771  * Make sure all the fields are properly specified
1772  * for a good command line.
1773  */
1774 if (state == COMMAND) {
1775     answer = TRUE;
1776     cmd->c_command = shcmd;

1778     /*
1779      * If no default level was supplied, insert
1780      * all numerical levels.
1781      */
1782     if (cmd->c_levels == 0)
1783         cmd->c_levels = MASK_NUMERIC;

1785     /*
1786      * If no action has been supplied, declare this
1787      * entry to be OFF.
1788      */

```

```

1789         if (cmd->c_action == 0)
1790             cmd->c_action = M_OFF;

1792         /*
1793          * If no shell command has been supplied, make sure
1794          * there is a null string in the command field.
1795          */
1796         if (ptr == shcmd + EXEC)
1797             *shcmd = '\0';
1798     } else
1799         answer = FALSE;

1801     /*
1802     * If we have reached the end of inittab, then close it
1803     * and quit trying to find a good command line.
1804     */
1805     if (c == EOF) {
1806         (void) fclose(fp_inittab);
1807         fp_inittab = NULL;
1808         break;
1809     }
1810     return (answer);
1811 }
1812 }

```

unchanged portion omitted

```

2041 /*
2042 * boot_init(): Do initialization things that should be done at boot.
2043 */
2044 void
2045 boot_init()
2046 {
2047     int i;
2048     struct PROC_TABLE *process, *oprocess;
2049     struct CMD_LINE cmd;
2050     char line[MAXCMDL];
2051     char svc_aux[SVC_AUX_SIZE];
2052     char init_svc_fmri[SVC_FMRI_SIZE];
2053     char *old_path;
2054     int maxfiles;

2056     /* Use INIT_PATH for sysinit cmds */
2057     old_path = glob_envp[0];
2058     glob_envp[0] = malloc((unsigned)(strlen(INIT_PATH)+2));
2059     (void) strcpy(glob_envp[0], INIT_PATH);

2061     /*
2062     * Scan inittab(4) and process the special svc.startd entry, initdefault
2063     * and sysinit entries.
2064     */
2065     while (getcmd(&cmd, &line[0]) == TRUE) {
2066         if (startd_tmpl >= 0 && id_eq(cmd.c_id, "smf")) {
2067             process_startd_line(&cmd, line);
2068             (void) snprintf(startd_svc_aux, SVC_AUX_SIZE,
2069                 INITTAB_ENTRY_ID_STR_FORMAT, cmd.c_id);
2070         } else if (cmd.c_action == M_INITDEFAULT) {
2071             /*
2072              * initdefault is no longer meaningful, as the SMF
2073              * milestone controls what (legacy) run level we
2074              * boot to.
2075              */
2076             console(B_TRUE,
2077                 "Ignoring legacy \"initdefault\" entry.\n");
2078         } else if (cmd.c_action == M_SYSINIT) {
2079             /*
2080              * Execute the "sysinit" entry and wait for it to

```

```

2081         * complete. No bookkeeping is performed on these
2082         * entries because we avoid writing to the file system
2083         * until after there has been a chance to check it.
2084         */
2085         if (process = findpslot(&cmd)) {
2086             (void) sighold(SIGCLD);
2087             (void) snprintf(svc_aux, SVC_AUX_SIZE,
2088                 INITTAB_ENTRY_ID_STR_FORMAT, cmd.c_id);
2089             (void) snprintf(init_svc_fmri, SVC_FMRI_SIZE,
2090                 SVC_INIT_PREFIX INITTAB_ENTRY_ID_STR_FORMAT,
2091                 cmd.c_id);
2092             if (legacy_tmpl >= 0) {
2093                 (void) ct_pr_tmpl_set_svc_fmri(
2094                     legacy_tmpl, init_svc_fmri);
2095                 (void) ct_pr_tmpl_set_svc_aux(
2096                     legacy_tmpl, svc_aux);
2097             }

2099             for (oprocess = process;
2100                 (process = efork(M_OFF, oprocess,
2101                     (NAMED|NOCLEANUP))) == NO_ROOM;
2102                 /* CSTYLED */)
2103                 ;
2104             (void) sigrelse(SIGCLD);

2106             if (process == NULLPROC) {
2107                 maxfiles = ulimit(UL_GDESLIM, 0);

2109                 for (i = 0; i < maxfiles; i++)
2110                     (void) fcntl(i, F_SETFD,
2111                         FD_CLOEXEC);
2112                 (void) execle(SH, "INITSH", "-c",
2113                     cmd.c_command,
2114                     (char *)0, glob_envp);
2115                 console(B_TRUE,
2116                     "Command\n\"%s\"\n failed to execute.  errno = %d (exec of shell failed)\n",
2117                     cmd.c_command, errno);
2118                 exit(1);
2119             } else
2120                 while (waitproc(process) == FAILURE)
2121                     ;
2122             } else while (waitproc(process) == FAILURE);
2123             process->p_flags = 0;
2124             st_write();
2125         }
2126     }

2128     /* Restore the path. */
2129     free(glob_envp[0]);
2130     glob_envp[0] = old_path;

2132     /*
2133     * This will enable st_write() to complain about init_state_file.
2134     */
2135     booting = 0;

2137     /*
2138     * If the /etc/ioctl.syscon didn't exist or had invalid contents write
2139     * out a correct version.
2140     */
2141     if (write_ioctl)
2142         write_ioctl_syscon();

2144     /*
2145     * Start svc.startd(1M), which does most of the work.

```



```

2146  */
2147  if (startd_cline[0] != '\0' && startd_tmpl >= 0) {
2148      /* Start svc.startd. */
2149      if (startd_run(startd_cline, startd_tmpl, 0) == -1)
2150          cur_state = SINGLE_USER;
2151  } else {
2152      console(B_TRUE, "Absent svc.startd entry or bad "
2153              "contract template. Not starting svc.startd.\n");
2154      enter_maintenance();
2155  }
2156  }

```

unchanged portion omitted

```

2811 /*
2812  * prog_name() searches for the word or unix path name and
2813  * returns a pointer to the last element of the pathname.
2814  */
2815 static char *
2816 prog_name(char *string)
2817 {
2818     char *ptr, *ptr2;
2819     static char word[UT_USER_SZ + 1];
2820     /* XXX - utmp - fix name length */
2821     static char word[_POSIX_LOGIN_NAME_MAX];
2822
2823     /*
2824      * Search for the first word skipping leading spaces and tabs.
2825      */
2826     while (*string == ' ' || *string == '\t')
2827         string++;
2828
2829     /*
2830      * If the first non-space non-tab character is not one allowed in
2831      * a word, return a pointer to a null string, otherwise parse the
2832      * pathname.
2833      */
2834     if (*string != '.' && *string != '/' && *string != '_' &&
2835         (*string < 'a' || *string > 'z') &&
2836         (*string < 'A' || *string > 'Z') &&
2837         (*string < '0' || *string > '9'))
2838         return ("");
2839
2840     /*
2841      * Parse the pathname looking forward for '/', '\t', '\n' or
2842      * '\0'. Each time a '/' is found, move "ptr" to one past the
2843      * '/', thus when a '\t', '\n', or '\0' is found, "ptr" will
2844      * point to the last element of the pathname.
2845      */
2846     for (ptr = string; *string != ' ' && *string != '\t' &&
2847          *string != '\n' && *string != '\0'; string++) {
2848         if (*string == '/')
2849             ptr = string+1;
2850     }
2851
2852     /*
2853      * Copy out up to the size of the "ut_user" array into "word",
2854      * null terminate it and return a pointer to it.
2855      */
2856     for (ptr2 = &word[0]; ptr2 < &word[UT_USER_SZ] &&
2857          /* XXX - utmp - fix name length */
2858          for (ptr2 = &word[0]; ptr2 < &word[_POSIX_LOGIN_NAME_MAX - 1] &&
2859               ptr < string; /* CSTYLED */)
2860         *ptr2++ = *ptr++;
2861
2862     *ptr2 = '\0';
2863     return (&word[0]);

```

```

2860  }
2861  unchanged portion omitted
2862
2863  /*
2864  * /etc/inittab has more entries and we have run out of room in the proc_table
2865  * array. Double the size of proc_table to accomodate the extra entries.
2866  */
2867  static void
2868  increase_proc_table_size()
2869  {
2870      sigset_t block, unblock;
2871      void *ptr;
2872      size_t delta = num_proc * sizeof (struct PROC_TABLE);
2873
2874      /*
2875       * Block signals for realloc.
2876       */
2877      (void) sigfillset(&block);
2878      (void) sigprocmask(SIG_BLOCK, &block, &unblock);
2879
2880      /*
2881       * On failure we just return because callers of this function check
2882       * for failure.
2883       */
2884      do
2885          ptr = realloc(g_state, g_state_sz + delta);
2886      while (ptr == NULL && errno == EAGAIN);
2887
2888      if (ptr != NULL) {
2889          /* ensure that the new part is initialized to zero */
2890          bzero((caddr_t)ptr + g_state_sz, delta);
2891
2892          g_state = ptr;
2893          g_state_sz += delta;
2894          num_proc <<= 1;
2895      }
2896
2897      /* unblock our signals before returning */
2898      (void) sigprocmask(SIG_SETMASK, &unblock, NULL);
2899  }
2900  unchanged portion omitted
2901
2902  /*
2903  * Initialize our state.
2904  */
2905  * If the system just booted, then init_state_file, which is located on an
2906  * everpresent tmpfs filesystem, should not exist.
2907  *
2908  * If we were restarted, then init_state_file should exist, in
2909  * which case we'll read it in, sanity check it, and use it.
2910  *
2911  * Note: You can't call console() until proc_table is ready.
2912  */
2913  void
2914  st_init()
2915  {
2916      struct stat stb;
2917      int ret, st_fd, insane = 0;
2918      size_t to_be_read;
2919      char *ptr;

```



```

4092         return (-1);
4093     }

4095     if (err = ct_pr_tmpl_set_param(fd, CT_PR_INHERIT | CT_PR_REGENT))
4096         console(B_TRUE, "Contract set template inherit, regent "
4097             "failed: %s.\n", strerror(err));

4099     /*
4100     * These errors result in a misconfigured template, which is better
4101     * than no template at all, so warn but don't abort.
4102     */
4103     if (err = ct_tmpl_set_informative(fd, info))
4104         console(B_TRUE, ioctl_tset_emsg, "informative", strerror(err));

4106     if (err = ct_tmpl_set_critical(fd, critical))
4107         console(B_TRUE, ioctl_tset_emsg, "critical", strerror(err));

4109     if (err = ct_pr_tmpl_set_fatal(fd, fatal))
4110         console(B_TRUE, ioctl_tset_emsg, "fatal", strerror(err));

4112     if (err = ct_tmpl_set_cookie(fd, cookie))
4113         console(B_TRUE, ioctl_tset_emsg, "cookie", strerror(err));

4115     (void) fcntl(fd, F_SETFD, FD_CLOEXEC);

4117     return (fd);
4118 }

4120 /*
4121 * Create the templates and open an event file descriptor. We use dup2(2) to
4122 * get these descriptors away from the stdin/stdout/stderr group.
4123 */
4124 static void
4125 contracts_init()
4126 {
4127     int err, fd;

4129     /*
4130     * Create & configure a legacy template. We only want empty events so
4131     * we know when to abandon them.
4132     */
4133     legacy_tmpl = contract_make_template(0, CT_PR_EV_EMPTY, CT_PR_EV_HWERR,
4134         ORDINARY_COOKIE);
4135     if (legacy_tmpl >= 0) {
4136         err = ct_tmpl_activate(legacy_tmpl);
4137         if (err != 0) {
4138             (void) close(legacy_tmpl);
4139             legacy_tmpl = -1;
4140             console(B_TRUE,
4141                 "Couldn't activate legacy template (%s): "
4142                 "legacy services will be in init's contract.\n",
4143                 strerror(err));
4144         }
4145     } else
4146         console(B_TRUE,
4147             "Legacy services will be in init's contract.\n");

4149     if (dup2(legacy_tmpl, 255) == -1) {
4150         console(B_TRUE, "Could not duplicate legacy template: %s.\n",
4151             strerror(errno));
4152     } else {
4153         (void) close(legacy_tmpl);
4154         legacy_tmpl = 255;
4155     }

4157     (void) fcntl(legacy_tmpl, F_SETFD, FD_CLOEXEC);

```

```

4159     startd_tmpl = contract_make_template(0, CT_PR_EV_EMPTY,
4160         CT_PR_EV_HWERR | CT_PR_EV_SIGNAL | CT_PR_EV_CORE, STARTD_COOKIE);

4162     if (dup2(startd_tmpl, 254) == -1) {
4163         console(B_TRUE, "Could not duplicate startd template: %s.\n",
4164             strerror(errno));
4165     } else {
4166         (void) close(startd_tmpl);
4167         startd_tmpl = 254;
4168     }

4170     (void) fcntl(startd_tmpl, F_SETFD, FD_CLOEXEC);

4172     if (legacy_tmpl < 0 && startd_tmpl < 0) {
4173         /* The creation errors have already been reported. */
4174         console(B_TRUE,
4175             "Ignoring contract events. Core smf(5) services will not "
4176             "be restarted.\n");
4177         return;
4178     }

4180     /*
4181     * Open an event endpoint.
4182     */
4183     do
4184         fd = open64(CTFS_ROOT "/process/pbundle", O_RDONLY);
4185     while (fd < 0 && errno == EINTR);
4186     ;
4187     while (fd < 0 && errno == EINTR);
4188     if (fd < 0) {
4189         console(B_TRUE,
4190             "Couldn't open process pbundle: %s. Core smf(5) services "
4191             "will not be restarted.\n", strerror(errno));
4192         return;
4193     }

4194     if (dup2(fd, 253) == -1) {
4195         console(B_TRUE, "Could not duplicate process bundle: %s.\n",
4196             strerror(errno));
4197     } else {
4198         (void) close(fd);
4199         fd = 253;
4200     }

4202     (void) fcntl(fd, F_SETFD, FD_CLOEXEC);

4204     /* Reset in case we've been restarted. */
4205     (void) ct_event_reset(fd);

4207     poll_fds[0].fd = fd;
4208     poll_fds[0].events = POLLIN;
4209     poll_nfds = 1;
4210 }

4212 static int
4213 contract_getfile(ctid_t id, const char *name, int oflag)
4214 {
4215     int fd;

4217     do
4218         fd = contract_open(id, "process", name, oflag);
4219     while (fd < 0 && errno == EINTR);
4220     ;
4221     while (fd < 0 && errno == EINTR);

```

new/usr/src/cmd/init/init.c

17

```
4222     if (fd < 0)
4223         console(B_TRUE, "Couldn't open %s for contract %ld: %s.\n",
4224                 name, id, strerror(errno));
4226     return (fd);
4227 }
```

unchanged_portion_omitted

```

*****
13232 Wed Apr  3 09:33:10 2013
new/usr/src/cmd/last/last.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright (c) 2013 Gary Mills
24 *
25 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
26 * Use is subject to license terms.
27 */
28
29 /*
30 *      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T
31 *      All Rights Reserved
32 */
33
34 /*
35 * University Copyright- Copyright (c) 1982, 1986, 1988
36 * The Regents of the University of California
37 * All Rights Reserved
38 *
39 * University Acknowledgment- Portions of this document are derived from
40 * software developed by the University of California, Berkeley, and its
41 * contributors.
42 */
43
44 #pragma ident      "%Z%M% %I%      %E% SMI"
45
46 /*
47 * last
48 */
49 #include <sys/types.h>
50 #include <stdio.h>
51 #include <stdlib.h>
52 #include <unistd.h>
53 #include <strings.h>
54 #include <signal.h>
55 #include <sys/stat.h>
56 #include <pwd.h>
57 #include <fcntl.h>
58 #include <utmpx.h>
59 #include <locale.h>
60 #include <ctype.h>

```

```

60 /*
61  * Use the full lengths from utmpx for NMAX, LMAX and HMAX .
62  * NMAX, LMAX and HMAX are set to these values for now. They
63  * should be much higher because of the max allowed limit in
64  * utmpx.h
65 */
66 #define NMAX      (sizeof (((struct utmpx *)0)->ut_user))
67 #define LMAX      (sizeof (((struct utmpx *)0)->ut_line))
68 #define NMAX      8
69 #define LMAX      12
70 #define HMAX      (sizeof (((struct utmpx *)0)->ut_host))
71
72 /* Print minimum field widths. */
73 #define LOGIN_WIDTH  8
74 #define LINE_WIDTH   12
75
76 #define SECDAY      (24*60*60)
77 #define CHUNK_SIZE  256
78
79 #define lineq(a, b)      (strncmp(a, b, LMAX) == 0)
80 #define nameq(a, b)     (strncmp(a, b, NMAX) == 0)
81 #define hosteq(a, b)    (strncmp(a, b, HMAX) == 0)
82 #define linehostnameq(a, b, c, d) \
83     (lineq(a, b)&&hosteq(a+LMAX+1, c)&&nameq(a+LMAX+HMAX+2, d))
84
85 #define USAGE        "usage: last [-n number] [-f filename] [-a ] [name | tty] ...\n"
86
87 /* Beware: These are set in main() to exclude the executable name. */
88 static char      **argv;
89 static int        argc;
90 static char      **names;
91 static int        names_num;
92
93 static struct      utmpx buf[128];
94
95 /*
96  * ttnames and logouts are allocated in the blocks of
97  * CHUNK_SIZE lines whenever needed. The count of the
98  * current size is maintained in the variable "lines"
99  * The variable bootxtime is used to hold the time of
100 * the last BOOT_TIME
101 * All elements of the logouts are initialised to bootxtime
102 * everytime the buffer is reallocated.
103 */
104 static char      **ttnames;
105 static time_t    *logouts;
106 static time_t    bootxtime;
107 static int        lines;
108 static char      timef[128];
109 static char      hostf[HMAX + 1];
110
111 static char *strspl(char *, char *);
112 static void onintr(int);
113 static void reallocate_buffer();
114 static void memory_alloc(int);
115 static int want(struct utmpx *, char **, char **);
116 static void record_time(time_t *, int *, int, struct utmpx *);
117
118 int
119 main(int ac, char **av)
120 {
121     int i, j;
122     int aflag = 0;
123     int fpos; /* current position in time format buffer */

```

```

120     int chrcnt;      /* # of chars formatted by current sprintf */
121     int bl, wtmp;
122     char *ct;
123     char *ut_host;
124     char *ut_user;
125     struct utmpx *bp;
126     time_t otime;
127     struct stat stb;
128     int print = 0;
129     char *crmsg = (char *)0;
130     long outrec = 0;
131     long maxrec = 0x7fffffffL;
132     char *wtmpfile = "/var/adm/wtmpx";
133     size_t hostf_len;

135     (void) setlocale(LC_ALL, "");
136 #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
137 #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it weren't. */
138 #endif
139     (void) textdomain(TEXT_DOMAIN);

141     (void) time(&buf[0].ut_xtime);
142     ac--, av++;
143     argc = ac;
144     argv = av;
145     names = malloc(argc * sizeof (char *));
146     if (names == NULL) {
147         perror("last");
148         exit(2);
149     }
150     names_num = 0;
151     for (i = 0; i < argc; i++) {
152         if (argv[i][0] == '-') {

154             /* -[0-9]* sets max # records to print */
155             if (isdigit(argv[i][1])) {
156                 maxrec = atoi(argv[i]+1);
157                 continue;
158             }

160             for (j = 1; argv[i][j] != '\0'; ++j) {
161                 switch (argv[i][j]) {

163                     /* -f name sets filename of wtmp file */
164                     case 'f':
165                         if (argv[i][j+1] != '\0') {
166                             wtmpfile = &argv[i][j+1];
167                         } else if (i+1 < argc) {
168                             wtmpfile = argv[i+1];
169                         } else {
170                             (void) fprintf(stderr,
171                                 gettext("last: argument to "
172                                     "-f is missing\n"));
173                             (void) fprintf(stderr,
174                                 gettext(USAGE));
175                             exit(1);
176                         }
177                         goto next_word;

179                     /* -n number sets max # records to print */
180                     case 'n': {
181                         char *arg;

183                         if (argv[i][j+1] != '\0') {
184                             arg = &argv[i][j+1];
185                         } else if (i+1 < argc) {

```

```

186         arg = argv[++i];
187     } else {
188         (void) fprintf(stderr,
189             gettext("last: argument to "
190                 "-n is missing\n"));
191         (void) fprintf(stderr,
192             gettext(USAGE));
193         exit(1);
194     }

196     if (!isdigit(*arg)) {
197         (void) fprintf(stderr,
198             gettext("last: argument to "
199                 "-n is not a number\n"));
200         (void) fprintf(stderr,
201             gettext(USAGE));
202         exit(1);
203     }
204     maxrec = atoi(arg);
205     goto next_word;
206 }

208     /* -a displays hostname last on the line */
209     case 'a':
210         aflag++;
211         break;

213     default:
214         (void) fprintf(stderr, gettext(USAGE));
215         exit(1);
216     }
217 }

219 next_word:
220     continue;
221 }

223     if (strlen(argv[i]) > 2 || strcmp(argv[i], "~") == 0 ||
224         getpwnam(argv[i]) != NULL) {
225         /* Not a tty number. */
226         names[names_num] = argv[i];
227         ++names_num;
228     } else {
229         /* tty number. Prepend "tty". */
230         names[names_num] = strspn("tty", argv[i]);
231         ++names_num;
232     }
233 }

235     wtmp = open(wtmpfile, 0);
236     if (wtmp < 0) {
237         perror(wtmpfile);
238         exit(1);
239     }
240     (void) fstat(wtmp, &stb);
241     bl = (stb.st_size + sizeof (buf)-1) / sizeof (buf);
242     if (signal(SIGINT, SIG_IGN) != SIG_IGN) {
243         (void) signal(SIGINT, onintr);
244         (void) signal(SIGQUIT, onintr);
245     }
246     lines = CHUNK_SIZE;
247     tttnames = calloc(lines, sizeof (char *));
248     logouts = calloc(lines, sizeof (time_t));
249     if (tttnames == NULL || logouts == NULL) {
250         (void) fprintf(stderr, gettext("Out of memory \n "));
251         exit(2);

```

```

252     }
253     for (bl--; bl >= 0; bl--) {
254         (void) lseek(wtmp, (off_t)(bl * sizeof (buf)), 0);
255         bp = &buf[read(wtmp, buf, sizeof (buf)) / sizeof (buf[0]) - 1];
256         for (; bp >= buf; bp--) {
257             if (want(bp, &ut_host, &ut_user)) {
258                 for (i = 0; i <= lines; i++) {
259                     if (i == lines)
260                         reallocate_buffer();
261                     if (ttnames[i] == NULL) {
262                         memory_alloc(i);
263                         /*
264                          * LMAX+HMAX+NMAX+3 bytes have been
265                          * allocated for ttnames[i].
266                          * If bp->ut_line is longer than LMAX,
267                          * ut_host is longer than HMAX,
268                          * and ut_user is longer than NMAX,
269                          * truncate it to fit ttnames[i].
270                          */
271                         (void) strncpy(ttnames[i], bp->ut_line,
272                                         LMAX+1);
273                         (void) strncpy(ttnames[i]+LMAX+1,
274                                         ut_host, HMAX+1);
275                         (void) strncpy(ttnames[i]+LMAX+HMAX+2,
276                                         ut_user, NMAX+1);
277                         record_time(&otime, &print,
278                                     i, bp);
279                         break;
280                     } else if (linehostnameq(ttnames[i],
281                                             bp->ut_line, ut_host, ut_user)) {
282                         record_time(&otime,
283                                     &print, i, bp);
284                         break;
285                     }
286                 }
287             }
288             if (print) {
289                 if (strcmp(bp->ut_line, "ftp", 3) == 0)
290                     bp->ut_line[3] = '\0';
291                 if (strcmp(bp->ut_line, "uucp", 4) == 0)
292                     bp->ut_line[4] = '\0';
293
294                 ct = ctime(&bp->ut_xtime);
295                 (void) printf(gettext("%-*.s %-*.s "),
296                               LOGIN_WIDTH, NMAX, bp->ut_name,
297                               LINE_WIDTH, LMAX, bp->ut_line);
298                 NMAX, NMAX, bp->ut_name,
299                 LMAX, LMAX, bp->ut_line);
300                 hostf_len = strlen(bp->ut_host);
301                 (void) snprintf(hostf, sizeof (hostf),
302                                 "%-*.s", hostf_len, hostf_len,
303                                 bp->ut_host);
304                 fpos = snprintf(timef, sizeof (timef),
305                                 "%10.10s %5.5s ",
306                                 ct, 11 + ct);
307                 if (!lineq(bp->ut_line, "system boot") &&
308                     !lineq(bp->ut_line, "system down")) {
309                     if (otime == 0 &&
310                         bp->ut_type == USER_PROCESS) {
311
312                     if (fpos < sizeof (timef)) {
313                         /* timef still has room */
314                         (void) snprintf(timef + fpos, sizeof (timef) - fpos,
315                                         gettext(" still logged in"));
316                     }
317                 }
318             }
319         }
320     }
321 }

```

```

316     } else {
317         time_t delta;
318         if (otime < 0) {
319             otime = -otime;
320             /*
321              * TRANSLATION_NOTE
322              * See other notes on "down"
323              * and "- %5.5s".
324              * "-" means "until". This
325              * is displayed after the
326              * starting time as in:
327              * 16:20 - down
328              * You probably don't want to
329              * translate this. Should you
330              * decide to translate this,
331              * translate "- %5.5s" too.
332              */
333         }
334         if (fpos < sizeof (timef)) {
335             /* timef still has room */
336             chrnt = snprintf(timef + fpos, sizeof (timef) - fpos,
337                             gettext("- %s"), crmsg);
338             fpos += chrnt;
339         }
340     } else {
341
342         if (fpos < sizeof (timef)) {
343             /* timef still has room */
344             chrnt = snprintf(timef + fpos, sizeof (timef) - fpos,
345                             gettext("- %5.5s"), ctime(&otime) + 11);
346             fpos += chrnt;
347         }
348     }
349 }
350
351 delta = otime - bp->ut_xtime;
352 if (delta < SECDAY) {
353
354     if (fpos < sizeof (timef)) {
355         /* timef still has room */
356         (void) snprintf(timef + fpos, sizeof (timef) - fpos,
357                         gettext(" (%5.5s)", asctime(gmtime(&delta)) + 11));
358     }
359 } else {
360
361     if (fpos < sizeof (timef)) {
362         /* timef still has room */
363         (void) snprintf(timef + fpos, sizeof (timef) - fpos,
364                         gettext(" (%ld+%5.5s)", delta / SECDAY,
365                             asctime(gmtime(&delta)) + 11));
366     }
367 }
368
369 }
370
371 if (aflag)
372     (void) printf("%-35.35s %-*.s\n",
373                 timef, strlen(hostf), hostf);
374 else
375     (void) printf("%-16.16s %-*.35s\n",
376                 hostf, timef);
377 (void) fflush(stdout);
378 if (++outrec >= maxrec)
379     exit(0);
380 }
381

```

```
382      /*
383      * when the system is down or crashed.
384      */
385      if (bp->ut_type == BOOT_TIME) {
386          for (i = 0; i < lines; i++)
387              logouts[i] = -bp->ut_xtime;
388          bootxtime = -bp->ut_xtime;
389          /*
390          * TRANSLATION_NOTE
391          * Translation of this "down " will replace
392          * the %s in "- %s". "down" is used instead
393          * of the real time session was ended, probably
394          * because the session ended by a sudden crash.
395          */
396          crmsg = gettext("down ");
397      }
398      print = 0;      /* reset the print flag */
399  }
400  }
401  ct = ctime(&buf[0].ut_xtime);
402  (void) printf(gettext("\nwtmp begins %10.10s %5.5s \n"), ct, ct + 11);
404  /* free() called to prevent lint warning about names */
405  free(names);
407  return (0);
408 }
_____unchanged_portion_omitted_
```



```

*****
21508 Wed Apr  3 09:33:10 2013
new/usr/src/cmd/newtask/newtask.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright (c) 2013 Gary Mills
24 *
25 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
26 * Use is subject to license terms.
27 */

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

29 #include <sys/types.h>
30 #include <sys/task.h>

32 #include <alloca.h>
33 #include <libproc.h>
34 #include <libintl.h>
35 #include <libgen.h>
36 #include <limits.h>
37 #include <project.h>
38 #include <pwd.h>
39 #include <secdb.h>
40 #include <stdio.h>
41 #include <stdlib.h>
42 #include <string.h>
43 #include <sys/varargs.h>
44 #include <unistd.h>
45 #include <errno.h>
46 #include <signal.h>
47 #include <priv_utils.h>

49 #include "utils.h"

51 #define OPTIONS_STRING  "Fc:lp:v"
52 #define NENV            8
53 #define ENVSIZE        255
54 #define PATH            "PATH=/usr/bin"
55 #define SUPATH         "PATH=/usr/sbin:/usr/bin"
56 #define SHELL           "/usr/bin/sh"
57 #define SHELL2         "/sbin/sh"
58 #define TIMEZONEFILE   "/etc/default/init"

```

```

59 #define LOGINFILE      "/etc/default/login"
60 #define GLOBAL_ERR_SZ  1024
61 #define GRAB_RETRY_MAX 100

63 static const char *pname;
64 extern char **environ;
65 static char *supath = SUPATH;
66 static char *path = PATH;
67 static char global_error[GLOBAL_ERR_SZ];
68 static int verbose = 0;

70 static priv_set_t *nset;

72 /* Private definitions for libproject */
73 extern projid_t setproject_proc(const char *, const char *, int, pid_t,
74     struct ps_prochandle *, struct project *);
75 extern priv_set_t *setproject_initpriv(void);

77 static void usage(void);

79 static void preserve_error(const char *format, ...);

81 static int update_running_proc(int, char *, char *);
82 static int set_ids(struct ps_prochandle *, struct project *,
83     struct passwd *);
84 static struct passwd *match_user(uid_t, char *, int);
85 static void setproject_err(char *, char *, int, struct project *);

87 static void
88 usage(void)
89 {
90     (void) fprintf(stderr, gettext("usage: \n\t%s [-v] [-p project] "
91         "[-c pid] [-F1] [command [args ...]]\n"), pname);
92     exit(2);
93 }

unchanged_portion_omitted

649 /*
650 * Given the input arguments, return the passwd structure that matches best.
651 * Also, since we use getpwnam() and friends, subsequent calls to this
652 * function will re-use the memory previously returned.
653 */
654 static struct passwd *
655 match_user(uid_t uid, char *projname, int is_my_uid)
656 {
657     char prbuf[PROJECT_BUFSZ], username[LOGNAME_MAX_ILLUMOS+1];
658     char prbuf[PROJECT_BUFSZ], username[LOGNAME_MAX+1];
659     struct project prj;
660     char *tmp_name;
661     struct passwd *pw = NULL;

662     /*
663      * In order to allow users with the same UID but distinguishable
664      * user names to be in different projects we play a guessing
665      * game of which username is most appropriate.  If we're checking
666      * for the uid of the calling process, the login name is a
667      * good starting point.
668      */
669     if (is_my_uid) {
670         if ((tmp_name = getlogin()) == NULL ||
671             (pw = getpwnam(tmp_name)) == NULL || (pw->pw_uid != uid) ||
672             (pw->pw_name == NULL))
673             pw = NULL;
674     }

676     /*

```

```

677     * If the login name doesn't work, we try the first match for
678     * the current uid in the password file.
679     */
680     if (pw == NULL) {
681         if (((pw = getpwuid(uid)) == NULL) || pw->pw_name == NULL) {
682             preserve_error(gettext("cannot find username "
683 "for uid %d"), uid);
684             return (NULL);
685         }
686     }
687
688     /*
689     * If projname wasn't supplied, we've done our best, so just return
690     * what we've got now. Alternatively, if newtask's invoker has
691     * superuser privileges, return the pw structure we've got now, with
692     * no further checking from inproj(). Superuser should be able to
693     * join any project, and the subsequent call to setproject() will
694     * allow this.
695     */
696     if (projname == NULL || getuid() == (uid_t)0)
697         return (pw);
698
699     (void) strncpy(username, pw->pw_name, sizeof (username) - 1);
700     username[sizeof (username) - 1] = '\0';
701     (void) strcpy(projname, pw->pw_name);
702
703     if (inproj(username, projname, prbuf, PROJECT_BUFSZ) == 0) {
704         char **u;
705         tmp_name = NULL;
706
707         /*
708         * If the previous guesses didn't work, walk through all
709         * project members and test for UID-equivalence.
710         */
711
712         if (getprojbyname(projname, &prj, prbuf,
713 PROJECT_BUFSZ) == NULL) {
714             preserve_error(gettext("unknown project \"%s\""),
715 projname);
716             return (NULL);
717         }
718
719         for (u = prj.pj_users; *u; u++) {
720             if ((pw = getpwnam(*u)) == NULL)
721                 continue;
722
723             if (pw->pw_uid == uid) {
724                 tmp_name = pw->pw_name;
725                 break;
726             }
727         }
728
729         if (tmp_name == NULL) {
730             preserve_error(gettext("user \"%s\" is not a member of "
731 "project \"%s\""), username, projname);
732             return (NULL);
733         }
734     }
735     return (pw);
736 }

```

unchanged_portion_omitted

new/usr/src/cmd/oamuser/inc/users.h

1

```
*****
2641 Wed Apr 3 09:33:10 2013
new/usr/src/cmd/oamuser/inc/users.h
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
25  */

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

31 #ifndef _USERS_H
32 #define _USERS_H

35 #include <pwd.h>
36 #include <grp.h>
37 #include <project.h>

39 #define GROUP          "/etc/group"

41 /* max number of projects that can be specified when adding a user */
42 #define NPROJECTS_MAX  1024

44 /* validation returns */
45 #define NOTUNIQUE      0      /* not unique */
46 #define RESERVED      1      /* reserved */
47 #define UNIQUE         2      /* is unique */
48 #define TOOBIG        3      /* number too big */
49 #define INVALID        4
50 #define LONGNAME       5      /* string too long */

52 /*
53  * Note: constraints checking for warning (release 2.6),
54  * and these may be enforced in the future releases.
55  */
56 #define WARN_NAME_TOO_LONG      0x1
57 #define WARN_BAD_GROUP_NAME     0x2
58 #define WARN_BAD_LOGNAME_CHAR  0x4
59 #define WARN_BAD_LOGNAME_FIRST 0x8
60 #define WARN_NO_LOWERCHAR      0x10
```

new/usr/src/cmd/oamuser/inc/users.h

2

```
61 #define WARN_BAD_PROJ_NAME     0x20
62 #define WARN_LOGGED_IN         0x40

64 /* Exit codes from passmgmt */
65 #define PEX_SUCCESS            0
66 #define PEX_NO_PERM            1
67 #define PEX_SYNTAX            2
68 #define PEX_BADARG            3
69 #define PEX_BADUID            4
70 #define PEX_HOSED_FILES       5
71 #define PEX_FAILED            6
72 #define PEX_MISSING           7
73 #define PEX_BUSY              8
74 #define PEX_BADNAME           9

76 #define REL_PATH(x)           (x && *x != '/')

78 /*
79  * interfaces available from the library
80  */
81 extern int valid_login(char *, struct passwd **, int *);
82 extern int valid_gname(char *, struct group **, int *);
83 extern int valid_group(char *, struct group **, int *);
84 extern int valid_project(char *, struct project *, void *buf, size_t, int *);
85 extern int valid_projname(char *, struct project *, void *buf, size_t, int *);
86 extern void warningmsg(int, char *);
87 extern void putgrent(struct group *, FILE *);

89 /* passmgmt */
90 #define PASSMGMT              "/usr/lib/passmgmt";
91 #endif /* _USERS_H */
```

```

*****
2115 Wed Apr 3 09:33:10 2013
new/usr/src/cmd/oamuser/lib/vlogin.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright (c) 2013 Gary Mills
24 *
25 * Copyright (c) 1997, by Sun Microsystems, Inc.
26 * All rights reserved.
27 */

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

31 #pragma ident      "%Z%M% %I%      %E% SMI"      /* SVr4.0 1.3 */

32 /*LINTLIBRARY*/

34 #include      <sys/types.h>
35 #include      <stdio.h>
36 #include      <ctype.h>
37 #include      <userdefs.h>
38 #include      <users.h>
39 #include      <limits.h>

41 /*
42  * validate string given as login name.
43  */
44 int
45 valid_login(char *login, struct passwd **pptr, int *warning)
46 {
47     struct passwd *t_pptr;
48     char *ptr = login;
49     int badlchar, badc, clower, len;
50     char c;

52     len = 0; clower = 0; badc = 0; badlchar = 0;
53     *warning = 0;
54     if (!login || !*login)
55         return (INVALID);

57     c = *ptr;

```

```

58     if (!isalpha(c))
59         badlchar++;
60     for (; c != NULL; ptr++, c = *ptr) {
61         len++;
62         if (!isprint(c) || (c == ':' ) || (c == '\n'))
63             return (INVALID);
64         if (!isalnum(c) && c != '_' && c != '-' && c != '.')
65             badc++;
66         if (islower(c))
67             clower++;
68     }

70     if (len > LOGNAME_MAX_ILLUMOS)
71         return (LONGNAME);

71     /*
72      * XXX length checking causes some operational/compatibility problem.
73      * This has to be revisited in the future as ARC/standards issue.
74      */
75     if (len > LOGNAME_MAX)
76         *warning = *warning | WARN_NAME_TOO_LONG;
77     if (clower == 0)
78         *warning = *warning | WARN_NO_LOWERCHAR;
79     if (badc != 0)
80         *warning = *warning | WARN_BAD_LOGNAME_CHAR;
81     if (badlchar != 0)
82         *warning = *warning | WARN_BAD_LOGNAME_FIRST;

83     if ((t_pptr = getpwnam(login)) != NULL) {
84         if (pptr) *pptr = t_pptr;
85         return (NOTUNIQUE);
86     }

    return (UNIQUE);
}

    unchanged_portion_omitted

```

```

*****
4796 Wed Apr 3 09:33:11 2013
new/usr/src/cmd/oamuser/user/messages.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
22 /*      All Rights Reserved */

25 /*
26  * Copyright (c) 2013 Gary Mills
27  *
28  * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
29  * Use is subject to license terms.
30  */

30 #pragma ident      "%Z%M% %I%      %E% SMI"      /* SVr4.0 1.6 */

32 char *errmsgs[] = {
33     "WARNING: uid %ld is reserved.\n",
34     "WARNING: more than NGROUPS_MAX(%d) groups specified.\n",
35     "ERROR: invalid syntax.\n"
36     "usage:  useradd [-u uid [-o] | -g group | -G group[,group]...] |"
37     "-d dir | -b base_dir |\n"
38     "\t\t-s shell | -c comment | -m [-k skel_dir] | -f inactive |\n"
39     "\t\t-e expire | -A authorization [, authorization ...] |\n"
40     "\t\t-P profile [, profile ...] | -R role [, role ...] |\n"
41     "\t\t-K key=value | -p project [, project ...] login\n"
42     "\tuseradd -D [-g group | -b base_dir | -f inactive | -e expire\n"
43     "\t\t-A authorization [, authorization ...] |\n"
44     "\t\t-P profile [, profile ...] | -R role [, role ...] |\n"
45     "\t\t-K key=value ... -p project] | [-s shell] | [-k skel_dir]\n",
46     "ERROR: Invalid syntax.\nusage:  userdel [-r] login\n",
47     "ERROR: Invalid syntax.\n"
48     "usage:  usermod -u uid [-o] | -g group | -G group[,group]... |\n"
49     "\t\t-d dir [-m] | -s shell | -c comment |\n"
50     "\t\t-l new_logname | -f inactive | -e expire |\n"
51     "\t\t-A authorization [, authorization ...] | -K key=value ... |\n"
52     "\t\t-P profile [, profile ...] | -R role [, role ...] login\n",
53     "ERROR: Unexpected failure. Defaults unchanged.\n",
54     "ERROR: Unable to remove files from home directory.\n",
55     "ERROR: Unable to remove home directory.\n",
56     "ERROR: Cannot update system files - login cannot be %s.\n",
57     "ERROR: uid %ld is already in use. Choose another.\n",
58     "ERROR: %s is already in use. Choose another.\n",

```

```

59     "ERROR: %s does not exist.\n",
60     "ERROR: %s is not a valid %s. Choose another.\n",
61     "ERROR: %s is in use. Cannot %s it.\n",
62     "WARNING: %s has no permissions to use %s.\n",
63     "ERROR: There is not sufficient space to move %s home directory to %s"
64     "\n",
65     "ERROR: %s %ld is too big. Choose another.\n",
66     "ERROR: group %s does not exist. Choose another.\n",
67     "ERROR: Unable to %s: %s.\n",
68     "ERROR: %s is not a full path name. Choose another.\n",
69     "ERROR: %s is the primary group name. Choose another.\n",
70     "ERROR: Inconsistent password files. See pwconv(1M).\n",
71     "ERROR: %s is not a local user.\n",
72     "ERROR: Permission denied.\n",
73     "WARNING: Group entry exceeds 2048 char: /etc/group entry truncated.\n",
74     "ERROR: invalid syntax.\n"
75     "usage:  roleadd [-u uid [-o] | -g group | -G group[,group]...] |"
76     "-d dir |\n"
77     "\t\t-s shell | -c comment | -m [-k skel_dir] | -f inactive |\n"
78     "\t\t-e expire | -A authorization [, authorization ...] |\n"
79     "\t\t-P profile [, profile ...] | -K key=value ] login\n"
80     "\troleadd -D [-g group | -b base_dir | -f inactive | -e expire\n"
81     "\t\t-A authorization [, authorization ...] |\n"
82     "\t\t-P profile [, profile ...] |\n",
83     "ERROR: Invalid syntax.\nusage:  roledel [-r] login\n",
84     "ERROR: Invalid syntax.\n"
85     "usage:  rolemod -u uid [-o] | -g group | -G group[,group]... |\n"
86     "\t\t-d dir [-m] | -s shell | -c comment |\n"
87     "\t\t-l new_logname | -f inactive | -e expire |\n"
88     "\t\t-A authorization [, authorization ...] | -K key=value |\n"
89     "\t\t-P profile [, profile ...] login\n",
90     "ERROR: project %s does not exist. Choose another.\n",
91     "WARNING: more than NPROJECTS_MAX(%d) projects specified.\n",
92     "WARNING: Project entry exceeds %d char: /etc/project entry truncated."
93     "\n",
94     "ERROR: Invalid key.\n",
95     "ERROR: Missing value specification.\n",
96     "ERROR: Multiple definitions of key ``%s``.\n",
97     "ERROR: Roles must be modified with ``rolemod``.\n",
98     "ERROR: Users must be modified with ``usermod``.\n",
99     "WARNING: gid %ld is reserved.\n",
100    "ERROR: Failed to read /etc/group file due to invalid entry or"
101    " read error.\n",
102    "ERROR: %s is too long. Choose another.\n",
103 };
    unchanged_portion_omitted

```

new/usr/src/cmd/oamuser/user/messages.h

1

```
*****
4075 Wed Apr 3 09:33:11 2013
new/usr/src/cmd/oamuser/user/messages.h
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
22 /*      All Rights Reserved */

25 /*
26  * Copyright (c) 2013 Gary Mills
27  *
28  * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
29  * Use is subject to license terms.
30  */

32 #ifndef _MESSAGES_H
33 #define _MESSAGES_H

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

35 extern void errmsg(int, ...);

37 /* WARNING: uid %d is reserved. */
38 #define M_RESERVED      0

40 /* WARNING: more than NGROUPS_MAX(%d) groups specified. */
41 #define M_MAXGROUPS      1

43 /* ERROR: invalid syntax.\nusage: useradd ... */
44 #define M_AUSAGE      2

46 /* ERROR: Invalid syntax.\nusage: userdel [-r] login\n" */
47 #define M_DUSAGE      3

49 /* ERROR: Invalid syntax.\nusage: usermod ... */
50 #define M_MUSAGE      4

53 /* ERROR: Unexpected failure. Defaults unchanged. */
54 #define M_FAILED      5

56 /* ERROR: Unable to remove files from home directory. */
57 #define M_RMFILES      6
```

new/usr/src/cmd/oamuser/user/messages.h

2

```
59 /* ERROR: Unable to remove home directory. */
60 #define M_RMHOME      7

62 /* ERROR: Cannot update system files - login cannot be %s. */
63 #define M_UPDATE      8

65 /* ERROR: uid %d is already in use. Choose another. */
66 #define M_UID_USED      9

68 /* ERROR: %s is already in use. Choose another. */
69 #define M_USED      10

71 /* ERROR: %s does not exist. */
72 #define M_EXIST      11

74 /* ERROR: %s is not a valid %s. Choose another. */
75 #define M_INVALID      12

77 /* ERROR: %s is in use. Cannot %s it. */
78 #define M_BUSY      13

80 /* WARNING: %s has no permissions to use %s. */
81 #define M_NO_PERM      14

83 /* ERROR: There is not sufficient space to move %s home directory to %s */
84 #define M_NOSPACE      15

86 /* ERROR: %s %d is too big. Choose another. */
87 #define M_TOOBIG      16

89 /* ERROR: group %s does not exist. Choose another. */
90 #define M_GRP_NOTUSED      17

92 /* ERROR: Unable to %s: %s */
93 #define M_OOPS      18

95 /* ERROR: %s is not a full path name. Choose another. */
96 #define M_RELPATH      19

98 /* ERROR: %s is the primary group name. Choose another. */
99 #define M_SAME_GRP      20

101 /* ERROR: Inconsistent password files. See pwconv(1M). */
102 #define M_HOSSED_FILES      21

104 /* ERROR: %s is not a local user. */
105 #define M_NONLOCAL      22

107 /* ERROR: Permission denied. */
108 #define M_PERM_DENIED      23

110 /* WARNING: Group entry exceeds 2048 char: /etc/group entry truncated. */
111 #define M_GROUP_ENTRY_OVF      24

113 /* ERROR: invalid syntax.\nusage: roleadd ... */
114 #define M_ARUSAGE      25

116 /* ERROR: Invalid syntax.\nusage: roledel [-r] login\n" */
117 #define M_DRUSAGE      26

119 /* ERROR: Invalid syntax.\nusage: rolemod -u ... */
120 #define M_MRUSAGE      27

122 /* ERROR: project %s does not exist. Choose another. */
123 #define M_PROJ_NOTUSED      28
```

new/usr/src/cmd/oamuser/user/messages.h

3

```
125 /* WARNING: more than NPROJECTS_MAX(%d) projects specified. */
126 #define M_MAXPROJECTS 29

128 /* WARNING: Project entry exceeds 512 char: /etc/project entry truncated. */
129 #define M_PROJ_ENTRY_OVF 30

131 /* ERROR: Invalid key. */
132 #define M_INVALID_KEY 31

134 /* ERROR: Missing value specification. */
135 #define M_INVALID_VALUE 32

137 /* ERROR: Multiple definitions of key ``%s''. */
138 #define M_REDEFINED_KEY 33

140 /* ERROR: Roles must be modified with rolemod */
141 #define M_ISROLE 34

143 /* ERROR: Users must be modified with usermod */
144 #define M_ISUSER 35

146 /* WARNING: gid %d is reserved. */
147 #define M_RESERVED_GID 36

149 /* ERROR: Failed to read /etc/group file due to invalid entry or read error. */
150 #define M_READ_ERROR 37

152 /* ERROR: %s is too long. Choose another. */
153 #define M_TOO_LONG 38

155 #endif /* _MESSAGES_H */
```

new/usr/src/cmd/oamuser/user/useradd.c

1

```
*****
17433 Wed Apr  3 09:33:11 2013
new/usr/src/cmd/oamuser/user/useradd.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 2008 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      */

32 #include <sys/types.h>
33 #include <sys/stat.h>
34 #include <sys/param.h>
35 #include <stdio.h>
36 #include <stdlib.h>
37 #include <ctype.h>
38 #include <limits.h>
39 #include <string.h>
40 #include <userdefs.h>
41 #include <errno.h>
42 #include <project.h>
43 #include <unistd.h>
44 #include <user_attr.h>
45 #include "users.h"
46 #include "messages.h"
47 #include "userdisp.h"
48 #include "funcs.h"

50 /*
51  * useradd [-u uid [-o] | -g group | -G group [[, group]...] | -d dir [-m]
52  *          | -s shell | -c comment | -k skel_dir | -b base_dir ]
53  *          [ -A authorization [, authorization ...]]
54  *          [ -P profile [, profile ...]]
55  *          [ -K key=value ]
56  *          [ -R role [, role ...]] [-p project [, project ...]] login
57  * useradd -D [ -g group | [ -b base_dir | -f inactive | -e expire |
58  *              -s shell | -k skel_dir ]
59  *          [ -A authorization [, authorization ...]]
60  *          [ -P profile [, profile ...]] [ -K key=value ]
```

new/usr/src/cmd/oamuser/user/useradd.c

2

```
61  *          [ -R role [, role ...]] [-p project [, project ...]] login
62  *
63  * This command adds new user logins to the system. Arguments are:
64  *
65  * uid - an integer
66  * group - an existing group's integer ID or char string name
67  * dir - home directory
68  * shell - a program to be used as a shell
69  * comment - any text string
70  * skel_dir - a skeleton directory
71  * base_dir - a directory
72  * login - a string of printable chars except colon(:)
73  * authorization - One or more comma separated authorizations defined
74  *                  in auth_attr(4).
75  * profile - One or more comma separated execution profiles defined
76  *           in prof_attr(4)
77  * role - One or more comma-separated role names defined in user_attr(4)
78  * project - One or more comma-separated project names or numbers
79  *
80  */

82 extern struct userdefs *getusrdef();
83 extern void dispusrdef();

85 static void cleanup();

87 extern uid_t findnextuid(void);
88 extern int check_perm(), valid_expire();
89 extern int putusrdef(), valid_uid();
90 extern int call_passmgmt(), edit_group(), create_home();
91 extern int edit_project();
92 extern int **valid_lgroup();
93 extern projid_t **valid_lproject();
94 extern void update_def(struct userdefs *);
95 extern void import_def(struct userdefs *);

97 static uid_t uid; /* new uid */
98 static char *logname; /* login name to add */
99 static struct userdefs *usrdefs; /* defaults for useradd */

101 char *cmdname;

103 static char homedir[ PATH_MAX + 1 ]; /* home directory */
104 static char gidstring[32]; /* group id string representation */
105 static gid_t gid; /* gid of new login */
106 static char uidstring[32]; /* user id string representation */
107 static char *uidstr = NULL; /* uid from command line */
108 static char *base_dir = NULL; /* base_dir from command line */
109 static char *group = NULL; /* group from command line */
110 static char *grps = NULL; /* multi groups from command line */
111 static char *dir = NULL; /* home dir from command line */
112 static char *shell = NULL; /* shell from command line */
113 static char *comment = NULL; /* comment from command line */
114 static char *skel_dir = NULL; /* skel dir from command line */
115 static long inact; /* inactive days */
116 static char *inactstr = NULL; /* inactive from command line */
117 static char inactstring[10]; /* inactivity string representation */
118 static char *expirestr = NULL; /* expiration date from command line */
119 static char *projects = NULL; /* project id's from command line */

121 static char *usertype = NULL; /* type of user, either role or normal */

123 typedef enum {
124     BASEDIR = 0,
125     SKELDIR,
126     SHELL
```



```

127 } path_opt_t;

130 static void valid_input(path_opt_t, const char *);

132 int
133 main(argc, argv)
134 int argc;
135 char *argv[];
136 {
137     int ch, ret, mflag = 0, oflag = 0, Dflag = 0, **gidlist;
138     projid_t **projlist;
139     char *ptr; /* loc in a str, may be set by strtol */
140     struct group *g_ptr;
141     struct project p_ptr;
142     char mybuf[PROJECT_BUFSZ];
143     struct stat statbuf; /* status buffer for stat */
144     int warning;
145     int busy = 0;
146     char **nargv; /* arguments for execvp of passmgmt */
147     int argindex; /* argument index into nargv */

149     cmdname = argv[0];

151     if (geteuid() != 0) {
152         errmsg(M_PERM_DENIED);
153         exit(EX_NO_PERM);
154     }

156     opterr = 0; /* no print errors from getopt */
157     usertype = getusertype(argv[0]);

159     change_key(USERATTR_TYPE_KW, usertype);

161     while ((ch = getopt(argc, argv,
162         "b:c:Dd:e:f:G:g:k:mop:s:u:A:P:R:K:")) != EOF)
163         switch (ch) {
164             case 'b':
165                 base_dir = optarg;
166                 break;

168             case 'c':
169                 comment = optarg;
170                 break;

172             case 'D':
173                 Dflag++;
174                 break;

176             case 'd':
177                 dir = optarg;
178                 break;

180             case 'e':
181                 expirestr = optarg;
182                 break;

184             case 'f':
185                 inactstr = optarg;
186                 break;

188             case 'G':
189                 grps = optarg;
190                 break;

192             case 'g':

```

```

193                 group = optarg;
194                 break;

196             case 'k':
197                 skel_dir = optarg;
198                 break;

200             case 'm':
201                 mflag++;
202                 break;

204             case 'o':
205                 oflag++;
206                 break;

208             case 'p':
209                 projects = optarg;
210                 break;

212             case 's':
213                 shell = optarg;
214                 break;

216             case 'u':
217                 uidstr = optarg;
218                 break;

220             case 'A':
221                 change_key(USERATTR_AUTHS_KW, optarg);
222                 break;

224             case 'P':
225                 change_key(USERATTR_PROFILES_KW, optarg);
226                 break;

228             case 'R':
229                 if (is_role(usertype)) {
230                     errmsg(M_ARUSAGE);
231                     exit(EX_SYNTAX);
232                 }
233                 change_key(USERATTR_ROLES_KW, optarg);
234                 break;

236             case 'K':
237                 change_key(NULL, optarg);
238                 break;

240             default:
241                 case '?':
242                     if (is_role(usertype))
243                         errmsg(M_ARUSAGE);
244                     else
245                         errmsg(M_AUSAGE);
246                     exit(EX_SYNTAX);
247                 }

249     /* get defaults for adding new users */
250     usrdefs = getusrdef(usertype);

252     if (Dflag) {
253         /* DISPLAY mode */

255         /* check syntax */
256         if (optind != argc) {
257             if (is_role(usertype))
258                 errmsg(M_ARUSAGE);

```

```

259         else
260             errmsg(M_AUSAGE);
261         exit(EX_SYNTAX);
262     }
263
264     if (uidstr != NULL || oflag || grps != NULL ||
265         dir != NULL || mflag || comment != NULL) {
266         if (is_role(usertype))
267             errmsg(M_ARUSAGE);
268         else
269             errmsg(M_AUSAGE);
270         exit(EX_SYNTAX);
271     }
272
273     /* Group must be an existing group */
274     if (group != NULL) {
275         switch (valid_group(group, &g_ptr, &warning)) {
276             case INVALID:
277                 errmsg(M_INVALID, group, "group id");
278                 exit(EX_BADARG);
279                 /*NOTREACHED*/
280             case TOOBIG:
281                 errmsg(M_TOOBIG, "gid", group);
282                 exit(EX_BADARG);
283                 /*NOTREACHED*/
284             case RESERVED:
285             case UNIQUE:
286                 errmsg(M_GRP_NOTUSED, group);
287                 exit(EX_NAME_NOT_EXIST);
288             }
289         if (warning)
290             warningmsg(warning, group);
291
292         usrdefs->defgroup = g_ptr->gr_gid;
293         usrdefs->defgname = g_ptr->gr_name;
294     }
295
296     /* project must be an existing project */
297     if (projects != NULL) {
298         switch (valid_project(projects, &p_ptr, mybuf,
299             sizeof(mybuf), &warning)) {
300             case INVALID:
301                 errmsg(M_INVALID, projects, "project id");
302                 exit(EX_BADARG);
303                 /*NOTREACHED*/
304             case TOOBIG:
305                 errmsg(M_TOOBIG, "projid", projects);
306                 exit(EX_BADARG);
307                 /*NOTREACHED*/
308             case UNIQUE:
309                 errmsg(M_PROJ_NOTUSED, projects);
310                 exit(EX_NAME_NOT_EXIST);
311             }
312         if (warning)
313             warningmsg(warning, projects);
314
315         usrdefs->defproj = p_ptr.pj_projid;
316         usrdefs->defprojname = p_ptr.pj_name;
317     }
318
319     /* base_dir must be an existing directory */
320     if (base_dir != NULL) {
321         valid_input(BASEDIR, base_dir);
322         usrdefs->defparent = base_dir;
323     }
324

```

```

326         /* inactivity period is an integer */
327         if (inactstr != NULL) {
328             /* convert inactstr to integer */
329             inact = strtol(inactstr, &ptr, 10);
330             if (*ptr || inact < 0) {
331                 errmsg(M_INVALID, inactstr,
332                     "inactivity period");
333                 exit(EX_BADARG);
334             }
335
336             usrdefs->definact = inact;
337         }
338
339         /* expiration string is a date, newer than today */
340         if (expirestr != NULL) {
341             if (*expirestr) {
342                 if (valid_expire(expirestr, (time_t *)0)
343                     == INVALID) {
344                     errmsg(M_INVALID, expirestr,
345                         "expiration date");
346                     exit(EX_BADARG);
347                 }
348                 usrdefs->defexpire = expirestr;
349             } else
350                 /* Unset the expiration date */
351                 usrdefs->defexpire = "";
352         }
353
354         if (shell != NULL) {
355             valid_input(SHELL, shell);
356             usrdefs->defshell = shell;
357         }
358         if (skel_dir != NULL) {
359             valid_input(SKELDIR, skel_dir);
360             usrdefs->defskel = skel_dir;
361         }
362         update_def(usrdefs);
363
364         /* change defaults for useradd */
365         if (putusrdef(usrdefs, usertype) < 0) {
366             errmsg(M_UPDATE, "created");
367             exit(EX_UPDATE);
368         }
369
370         /* Now, display */
371         dispusrdef(stdout, (D_ALL & ~D_RID), usertype);
372         exit(EX_SUCCESS);
373     }
374
375     /* ADD mode */
376
377     /* check syntax */
378     if (optind != argc - 1 || (skel_dir != NULL && !mflag)) {
379         if (is_role(usertype))
380             errmsg(M_ARUSAGE);
381         else
382             errmsg(M_AUSAGE);
383         exit(EX_SYNTAX);
384     }
385
386     logname = argv[optind];
387     switch (valid_login(logname, (struct passwd **)NULL, &warning)) {
388     case INVALID:
389         errmsg(M_INVALID, logname, "login name");
390

```

```

391         exit(EX_BADARG);
392         /*NOTREACHED*/

394     case NOTUNIQUE:
395         errmsg(M_USED, logname);
396         exit(EX_NAME_EXISTS);
397         /*NOTREACHED*/

399     case LONGNAME:
400         errmsg(M_TOO_LONG, logname);
401         exit(EX_BADARG);
402         /*NOTREACHED*/
403     }

405     if (warning)
406         warningmsg(warning, logname);
407     if (uidstr != NULL) {
408         /* convert uidstr to integer */
409         errno = 0;
410         uid = (uid_t)strtol(uidstr, &ptr, (int)10);
411         if (*ptr || errno == ERANGE) {
412             errmsg(M_INVALID, uidstr, "user id");
413             exit(EX_BADARG);
414         }

416         switch (valid_uid(uid, NULL)) {
417             case NOTUNIQUE:
418                 if (!oflag) {
419                     /* override not specified */
420                     errmsg(M_UID_USED, uid);
421                     exit(EX_ID_EXISTS);
422                 }
423                 break;
424             case RESERVED:
425                 errmsg(M_RESERVED, uid);
426                 break;
427             case TOOBIG:
428                 errmsg(M_TOOBIG, "uid", uid);
429                 exit(EX_BADARG);
430                 break;
431         }

433     } else {

435         if ((uid = findnextuid()) < 0) {
436             errmsg(M_INVALID, "default id", "user id");
437             exit(EX_ID_EXISTS);
438         }
439     }

441     if (group != NULL) {
442         switch (valid_group(group, &g_ptr, &warning)) {
443             case INVALID:
444                 errmsg(M_INVALID, group, "group id");
445                 exit(EX_BADARG);
446                 /*NOTREACHED*/
447             case TOOBIG:
448                 errmsg(M_TOOBIG, "gid", group);
449                 exit(EX_BADARG);
450                 /*NOTREACHED*/
451             case RESERVED:
452             case UNIQUE:
453                 errmsg(M_GRP_NOTUSED, group);
454                 exit(EX_NAME_NOT_EXIST);
455                 /*NOTREACHED*/
456         }

```

```

458         if (warning)
459             warningmsg(warning, group);
460         gid = g_ptr->gr_gid;

462     } else gid = usrdefs->defgroup;

464     if (grps != NULL) {
465         if (!*grps)
466             /* ignore -G "" */
467             grps = (char *)0;
468         else if (!(gidlist = valid_lgroup(grps, gid)))
469             exit(EX_BADARG);
470     }

472     if (projects != NULL) {
473         if (!*projects)
474             projects = (char *)0;
475         else if (!(projlist = valid_lproject(projects)))
476             exit(EX_BADARG);
477     }

479     /* if base_dir is provided, check its validity; otherwise default */
480     if (base_dir != NULL)
481         valid_input(BASEDIR, base_dir);
482     else
483         base_dir = usrdefs->defparent;

485     if (dir == NULL) {
486         /* set homedir to home directory made from base_dir */
487         (void) sprintf(homedir, "%s/%s", base_dir, logname);

489     } else if (REL_PATH(dir)) {
490         errmsg(M_RELPATH, dir);
491         exit(EX_BADARG);

493     } else
494         (void) strcpy(homedir, dir);

496     if (mflag) {
497         /* Does home dir. already exist? */
498         if (stat(homedir, &statbuf) == 0) {
499             /* directory exists - don't try to create */
500             mflag = 0;

502             if (check_perm(statbuf, uid, gid, S_IXOTH) != 0)
503                 errmsg(M_NO_PERM, logname, homedir);
504         }
505     }
506     /*
507     * if shell, skel_dir are provided, check their validity.
508     * Otherwise default.
509     */
510     if (shell != NULL)
511         valid_input(SHELL, shell);
512     else
513         shell = usrdefs->defshell;

515     if (skel_dir != NULL)
516         valid_input(SKELDIR, skel_dir);
517     else
518         skel_dir = usrdefs->defskel;

520     if (inactstr != NULL) {
521         /* convert inactstr to integer */
522         inact = strtol(inactstr, &ptr, 10);

```

```

523         if (*ptr || inact < 0) {
524             errmsg(M_INVALID, inactstr, "inactivity period");
525             exit(EX_BADARG);
526         }
527     } else inact = usrdefs->definact;

529     /* expiration string is a date, newer than today */
530     if (expirestr != NULL) {
531         if (*expirestr) {
532             if (valid_expire(expirestr, (time_t *)0) == INVALID) {
533                 errmsg(M_INVALID, expirestr, "expiration date");
534                 exit(EX_BADARG);
535             }
536             usrdefs->defexpire = expirestr;
537         } else
538             /* Unset the expiration date */
539             expirestr = (char *)0;

541     } else expirestr = usrdefs->defexpire;

543     import_def(usrdefs);

545     /* must now call passmgmt */

547     /* set up arguments to passmgmt in nargv array */
548     nargv = malloc((30 + nkeys * 2) * sizeof(char *));
549     argindex = 0;
550     nargv[argindex++] = PASSMGMT;
551     nargv[argindex++] = "-a";        /* add */

553     if (comment != NULL) {
554         /* comment */
555         nargv[argindex++] = "-c";
556         nargv[argindex++] = comment;
557     }

559     /* flags for home directory */
560     nargv[argindex++] = "-h";
561     nargv[argindex++] = homedir;

563     /* set gid flag */
564     nargv[argindex++] = "-g";
565     (void) sprintf(gidstring, "%u", gid);
566     nargv[argindex++] = gidstring;

568     /* shell */
569     nargv[argindex++] = "-s";
570     nargv[argindex++] = shell;

572     /* set inactive */
573     nargv[argindex++] = "-f";
574     (void) sprintf(inactstring, "%ld", inact);
575     nargv[argindex++] = inactstring;

577     /* set expiration date */
578     if (expirestr != NULL) {
579         nargv[argindex++] = "-e";
580         nargv[argindex++] = expirestr;
581     }

583     /* set uid flag */
584     nargv[argindex++] = "-u";
585     (void) sprintf(uidstring, "%u", uid);
586     nargv[argindex++] = uidstring;

588     if (oflag) nargv[argindex++] = "-o";

```

```

590     if (nkeys > 1)
591         addkey_args(nargv, &argindex);

593     /* finally - login name */
594     nargv[argindex++] = logname;

596     /* set the last to null */
597     nargv[argindex++] = NULL;

599     /* now call passmgmt */
600     ret = PEX_FAILED;
601     /*
602     * If call_passmgmt fails for any reason other than PEX_BADUID, exit
603     * is invoked with an appropriate error message. If PEX_BADUID is
604     * returned, then if the user specified the ID, exit is invoked
605     * with an appropriate error message. Otherwise we try to pick a
606     * different ID and try again. If we run out of IDs, i.e. no more
607     * users can be created, then -1 is returned and we terminate via exit.
608     * If PEX_BUSY is returned we increment a count, since we will stop
609     * trying if PEX_BUSY reaches 3. For PEX_SUCCESS we immediately
610     * terminate the loop.
611     */
612     while (busy < 3 && ret != PEX_SUCCESS) {
613         switch (ret = call_passmgmt(nargv)) {
614             case PEX_SUCCESS:
615                 break;
616             case PEX_BUSY:
617                 busy++;
618                 break;
619             case PEX_HOSED_FILES:
620                 errmsg(M_HOSED_FILES);
621                 exit(EX_INCONSISTENT);
622                 break;

624             case PEX_SYNTAX:
625             case PEX_BADARG:
626                 /* should NEVER occur that passmgmt usage is wrong */
627                 if (is_role(usertype))
628                     errmsg(M_ARUSAGE);
629                 else
630                     errmsg(M_AUSAGE);
631                 exit(EX_SYNTAX);
632                 break;

634             case PEX_BADUID:
635                 /*
636                 * The uid has been taken. If it was specified by a
637                 * user, then we must fail. Otherwise, keep trying
638                 * to get a good uid until we run out of IDs.
639                 */
640                 if (uidstr != NULL) {
641                     errmsg(M_UID_USED, uid);
642                     exit(EX_ID_EXISTS);
643                 } else {
644                     if ((uid = findnextuid()) < 0) {
645                         errmsg(M_INVALID, "default id",
646                             "user id");
647                         exit(EX_ID_EXISTS);
648                     }
649                     (void) sprintf(uidstring, "%u", uid);
650                 }
651                 break;

653             case PEX_BADNAME:
654                 /* invalid loname */

```

```
655         errmsg(M_USED, logname);
656         exit(EX_NAME_EXISTS);
657         break;

659         default:
660             errmsg(M_UPDATE, "created");
661             exit(ret);
662             break;
663     }
664 }
665 if (busy == 3) {
666     errmsg(M_UPDATE, "created");
667     exit(ret);
668 }

670 /* add group entry */
671 if ((grps != NULL) && edit_group(logname, (char *)0, gidlist, 0)) {
672     errmsg(M_UPDATE, "created");
673     cleanup(logname);
674     exit(EX_UPDATE);
675 }

677 /* update project database */
678 if ((projects != NULL) &&
679     edit_project(logname, (char *)NULL, projlist, 0)) {
680     errmsg(M_UPDATE, "created");
681     cleanup(logname);
682     exit(EX_UPDATE);
683 }

685 /* create home directory */
686 if (mflag &&
687     (create_home(homedir, skel_dir, uid, gid) != EX_SUCCESS)) {
688     (void) edit_group(logname, (char *)0, (int **)0, 1);
689     cleanup(logname);
690     exit(EX_HOMEDIR);
691 }

693     return (ret);
694 }
unchanged portion omitted
```

```

*****
15671 Wed Apr  3 09:33:11 2013
new/usr/src/cmd/oamuser/user/usermod.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 2008 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      */

33 #include <sys/types.h>
34 #include <sys/stat.h>
35 #include <sys/param.h>
36 #include <stdio.h>
37 #include <stdlib.h>
38 #include <ctype.h>
39 #include <limits.h>
40 #include <string.h>
41 #include <userdefs.h>
42 #include <user_attr.h>
43 #include <nss_dbdefs.h>
44 #include <errno.h>
45 #include <project.h>
46 #include "users.h"
47 #include "messages.h"
48 #include "funcs.h"

50 /*
51 * usermod [-u uid [-o] | -g group | -G group [[,group]...] | -d dir [-m]
52 *          | -s shell | -c comment | -l new_logname]
53 *          [-f inactive | -e expire ]
54 *          [-A authorization [, authorization ...]]
55 *          [-P profile [, profile ...]]
56 *          [-R role [, role ...]]
57 *          [-K key=value ]
58 *          [-p project [, project]] login
59 *
60 *      This command adds new user logins to the system.  Arguments are:

```

```

61 *
62 *      uid - an integer less than MAXUID
63 *      group - an existing group's integer ID or char string name
64 *      dir - a directory
65 *      shell - a program to be used as a shell
66 *      comment - any text string
67 *      skel_dir - a directory
68 *      base_dir - a directory
69 *      rid - an integer less than 2**16 (USHORT)
70 *      login - a string of printable chars except colon (:)
71 *      inactive - number of days a login maybe inactive before it is locked
72 *      expire - date when a login is no longer valid
73 *      authorization - One or more comma separated authorizations defined
74 *                    in auth_attr(4).
75 *      profile - One or more comma separated execution profiles defined
76 *              in prof_attr(4)
77 *      role - One or more comma-separated role names defined in user_attr(4)
78 *      key=value - One or more -K options each specifying a valid user_attr(4)
79 *                attribute.
80 *
81 */

83 extern int **valid_lgroup(), isbusy();
84 extern int valid_uid(), check_perm(), create_home(), move_dir();
85 extern int valid_expire(), edit_group(), call_passmgmt();
86 extern projid_t **valid_lproject();

88 static uid_t uid;          /* new uid */
89 static gid_t gid;         /* gid of new login */
90 static char *new_logname = NULL; /* new login name with -l option */
91 static char *uidstr = NULL; /* uid from command line */
92 static char *group = NULL; /* group from command line */
93 static char *grps = NULL; /* multi groups from command line */
94 static char *dir = NULL; /* home dir from command line */
95 static char *shell = NULL; /* shell from command line */
96 static char *comment = NULL; /* comment from command line */
97 static char *logname = NULL; /* login name to add */
98 static char *inactstr = NULL; /* inactive from command line */
99 static char *expire = NULL; /* expiration date from command line */
100 static char *projects = NULL; /* project ids from command line */
101 static char *usertype;

103 char *cmdname;
104 static char gidstring[32], uidstring[32];
105 char inactstring[10];

107 char *
108 strcpmalloc(str)
109 char *str;
110 {
111     if (str == NULL)
112         return (NULL);
113
114     return (strdup(str));
115 }

unchanged_portion_omitted

141 int
142 main(argc, argv)
143 int argc;
144 char **argv;
145 {
146     int ch, ret = EX_SUCCESS, call_pass = 0, oflag = 0;
147     int tries, mflag = 0, inact, **gidlist, flag = 0;
148     boolean_t fail_if_busy = B_FALSE;
149     char *ptr;

```

```

150     struct passwd *pstruct;          /* password struct for login */
151     struct passwd *pw;
152     struct group *g_ptr;             /* validated group from -g */
153     struct stat statbuf;             /* status buffer for stat */
154 #ifndef att
155     FILE *pwf;                       /* fille ptr for opened passwd file */
156 #endif
157     int warning;
158     projid_t **projlist;
159     char **nargv;                    /* arguments for execvp of passgmt */
160     int argindex;                    /* argument index into nargv */
161     userattr_t *ua;
162     char *val;
163     int isrole;                      /* current account is role */

165     cmdname = argv[0];

167     if (geteuid() != 0) {
168         errmsg(M_PERM_DENIED);
169         exit(EX_NO_PERM);
170     }

172     opterr = 0;                      /* no print errors from getopt */
173     /* get user type based on the program name */
174     usertype = getusertype(argv[0]);

176     while ((ch = getopt(argc, argv,
177         "c:d:e:f:G:g:l:mop:s:u:A:P:R:K:")) != EOF)
178         switch (ch) {
179             case 'c':
180                 comment = optarg;
181                 flag++;
182                 break;
183             case 'd':
184                 dir = optarg;
185                 fail_if_busy = B_TRUE;
186                 flag++;
187                 break;
188             case 'e':
189                 expire = optarg;
190                 flag++;
191                 break;
192             case 'f':
193                 inactstr = optarg;
194                 flag++;
195                 break;
196             case 'G':
197                 grps = optarg;
198                 flag++;
199                 break;
200             case 'g':
201                 group = optarg;
202                 fail_if_busy = B_TRUE;
203                 flag++;
204                 break;
205             case 'l':
206                 new_logname = optarg;
207                 fail_if_busy = B_TRUE;
208                 flag++;
209                 break;
210             case 'm':
211                 mflag++;
212                 flag++;
213                 fail_if_busy = B_TRUE;
214                 break;
215             case 'o':

```

```

216                 oflag++;
217                 flag++;
218                 fail_if_busy = B_TRUE;
219                 break;
220             case 'p':
221                 projects = optarg;
222                 flag++;
223                 break;
224             case 's':
225                 shell = optarg;
226                 flag++;
227                 break;
228             case 'u':
229                 uidstr = optarg;
230                 flag++;
231                 fail_if_busy = B_TRUE;
232                 break;
233             case 'A':
234                 change_key(USERATTR_AUTHS_KW, optarg);
235                 flag++;
236                 break;
237             case 'P':
238                 change_key(USERATTR_PROFILES_KW, optarg);
239                 flag++;
240                 break;
241             case 'R':
242                 change_key(USERATTR_ROLES_KW, optarg);
243                 flag++;
244                 break;
245             case 'K':
246                 change_key(NULL, optarg);
247                 flag++;
248                 break;
249             default:
250                 case '?':
251                     if (is_role(usertype))
252                         errmsg(M_MRUSAGE);
253                     else
254                         errmsg(M_MUSAGE);
255                     exit(EX_SYNTAX);
256                 }

258     if (optind != argc - 1 || flag == 0) {
259         if (is_role(usertype))
260             errmsg(M_MRUSAGE);
261         else
262             errmsg(M_MUSAGE);
263         exit(EX_SYNTAX);
264     }

266     if ((!uidstr && oflag) || (mflag && !dir)) {
267         if (is_role(usertype))
268             errmsg(M_MRUSAGE);
269         else
270             errmsg(M_MUSAGE);
271         exit(EX_SYNTAX);
272     }

274     logname = argv[optind];

276     /* Determine whether the account is a role or not */
277     if ((ua = getusernam(logname)) == NULL ||
278         (val = kva_match(ua->attr, USERATTR_TYPE_KW)) == NULL ||
279         strcmp(val, USERATTR_TYPE_NONADMIN_KW) != 0)
280         isrole = 0;
281     else

```

```

282         isrole = 1;

284     /* Verify that rolemod is used for roles and usermod for users */
285     if (isrole != is_role(usertype)) {
286         if (isrole)
287             errormsg(M_ISROLE);
288         else
289             errormsg(M_ISUSER);
290         exit(EX_SYNTAX);
291     }

293     /* Set the usertype key; defaults to the commandline */
294     usertype = getsetdefval(USERATTR_TYPE_KW, usertype);

296     if (is_role(usertype)) {
297         /* Roles can't have roles */
298         if (getsetdefval(USERATTR_ROLES_KW, NULL) != NULL) {
299             errormsg(M_MRUSAGE);
300             exit(EX_SYNTAX);
301         }
302         /* If it was an ordinary user, delete its roles */
303         if (!isrole)
304             change_key(USERATTR_ROLES_KW, "");
305     }

307 #ifdef att
308     pw = getpwnam(logname);
309 #else
310     /*
311     * Do this with fgetpwent to make sure we are only looking on local
312     * system (since passgmt only works on local system).
313     */
314     if ((pwf = fopen("/etc/passwd", "r")) == NULL) {
315         errormsg(M_OOPS, "open", "/etc/passwd");
316         exit(EX_FAILURE);
317     }
318     while ((pw = fgetpwent(pwf)) != NULL)
319         if (strcmp(pw->pw_name, logname) == 0)
320             break;

322     fclose(pwf);
323 #endif

325     if (pw == NULL) {
326         char          pwdb[NSS_BUFLLEN_PASSWD];
327         struct passwd  pwd;

329         if (getpwnam_r(logname, &pwd, pwdb, sizeof (pwdb)) == NULL) {
330             /* This user does not exist. */
331             errormsg(M_EXIST, logname);
332             exit(EX_NAME_NOT_EXIST);
333         } else {
334             /* This user exists in non-local name service. */
335             errormsg(M_NONLOCAL, logname);
336             exit(EX_NOT_LOCAL);
337         }
338     }

340     pstruct = passwd_cpmalloc(pw);

342     /*
343     * We can't modify a logged in user if any of the following
344     * are being changed:
345     * uid (-u & -o), group (-g), home dir (-m), loginname (-l).
346     * If none of those are specified it is okay to go ahead
347     * some types of changes only take effect on next login, some

```

```

348     * like authorisations and profiles take effect instantly.
349     * One might think that -K type=role should require that the
350     * user not be logged in, however this would make it very
351     * difficult to make the root account a role using this command.
352     */
353     if (isbusy(logname)) {
354         if (fail_if_busy) {
355             errormsg(M_BUSY, logname, "change");
356             exit(EX_BUSY);
357         }
358         warningmsg(WARN_LOGGED_IN, logname);
359     }

361     if (new_logname && strcmp(new_logname, logname)) {
362         switch (valid_login(new_logname, (struct passwd **)NULL,
363             &warning)) {
364             case INVALID:
365                 errormsg(M_INVALID, new_logname, "login name");
366                 exit(EX_BADARG);
367                 /*NOTREACHED*/

369             case NOTUNIQUE:
370                 errormsg(M_USED, new_logname);
371                 exit(EX_NAME_EXISTS);
372                 /*NOTREACHED*/

374             case LONGNAME:
375                 errormsg(M_TOO_LONG, new_logname);
376                 exit(EX_BADARG);
377                 /*NOTREACHED*/

379             default:
380                 call_pass = 1;
381                 break;
382         }
383         if (warning)
384             warningmsg(warning, logname);
385     }

387     if (uidstr) {
388         /* convert uidstr to integer */
389         errno = 0;
390         uid = (uid_t)strtol(uidstr, &ptr, (int)10);
391         if (*ptr || errno == ERANGE) {
392             errormsg(M_INVALID, uidstr, "user id");
393             exit(EX_BADARG);
394         }

396         if (uid != pstruct->pw_uid) {
397             switch (valid_uid(uid, NULL)) {
398                 case NOTUNIQUE:
399                     if (!oflag) {
400                         /* override not specified */
401                         errormsg(M_UID_USED, uid);
402                         exit(EX_ID_EXISTS);
403                     }
404                     break;
405                 case RESERVED:
406                     errormsg(M_RESERVED, uid);
407                     break;
408                 case TOOBIG:
409                     errormsg(M_TOOBIG, "uid", uid);
410                     exit(EX_BADARG);
411                     break;
412             }

```



```

414         call_pass = 1;
416     } else {
417         /* uid's the same, so don't change anything */
418         uidstr = NULL;
419         oflag = 0;
420     }
422 } else uid = pstruct->pw_uid;
424 if (group) {
425     switch (valid_group(group, &g_ptr, &warning)) {
426     case INVALID:
427         errmsg(M_INVALID, group, "group id");
428         exit(EX_BADARG);
429         /*NOTREACHED*/
430     case TOOBIG:
431         errmsg(M_TOOBIG, "gid", group);
432         exit(EX_BADARG);
433         /*NOTREACHED*/
434     case UNIQUE:
435         errmsg(M_GRP_NOTUSED, group);
436         exit(EX_NAME_NOT_EXIST);
437         /*NOTREACHED*/
438     case RESERVED:
439         gid = (gid_t)strtol(group, &ptr, (int)10);
440         errmsg(M_RESERVED_GID, gid);
441         break;
442     }
443     if (warning)
444         warningmsg(warning, group);
446     if (g_ptr != NULL)
447         gid = g_ptr->gr_gid;
448     else
449         gid = pstruct->pw_gid;
451     /* call passmgmt if gid is different, else ignore group */
452     if (gid != pstruct->pw_gid)
453         call_pass = 1;
454     else group = NULL;
456 } else gid = pstruct->pw_gid;
458 if (grps && *grps) {
459     if (!(gidlist = valid_lgroup(grps, gid)))
460         exit(EX_BADARG);
461 } else
462     gidlist = (int **)0;
464 if (projects && *projects) {
465     if (!(projlist = valid_lproject(projects)))
466         exit(EX_BADARG);
467 } else
468     projlist = (projid_t **)0;
470 if (dir) {
471     if (REL_PATH(dir)) {
472         errmsg(M_RELPATH, dir);
473         exit(EX_BADARG);
474     }
475     if (strcmp(pstruct->pw_dir, dir) == 0) {
476         /* home directory is the same so ignore dflag & mflag */
477         dir = NULL;
478         mflag = 0;
479     } else call_pass = 1;

```

```

480     }
482     if (mflag) {
483         if (stat(dir, &statbuf) == 0) {
484             /* Home directory exists */
485             if (check_perm(statbuf, pstruct->pw_uid,
486                 pstruct->pw_gid, S_IWOTH|S_IXOTH) != 0) {
487                 errmsg(M_NO_PERM, logname, dir);
488                 exit(EX_NO_PERM);
489             }
491         } else ret = create_home(dir, NULL, uid, gid);
493         if (ret == EX_SUCCESS)
494             ret = move_dir(pstruct->pw_dir, dir, logname);
496         if (ret != EX_SUCCESS)
497             exit(ret);
498     }
500     if (shell) {
501         if (REL_PATH(shell)) {
502             errmsg(M_RELPATH, shell);
503             exit(EX_BADARG);
504         }
505         if (strcmp(pstruct->pw_shell, shell) == 0) {
506             /* ignore s option if shell is not different */
507             shell = NULL;
508         } else {
509             if (stat(shell, &statbuf) < 0 ||
510                 (statbuf.st_mode & S_IFMT) != S_IFREG ||
511                 (statbuf.st_mode & 0555) != 0555) {
513                 errmsg(M_INVALID, shell, "shell");
514                 exit(EX_BADARG);
515             }
517             call_pass = 1;
518         }
519     }
521     if (comment)
522         /* ignore comment if comment is not changed */
523         if (strcmp(pstruct->pw_comment, comment))
524             call_pass = 1;
525     else
526         comment = NULL;
528     /* inactive string is a positive integer */
529     if (inactstr) {
530         /* convert inactstr to integer */
531         inact = (int)strtol(inactstr, &ptr, 10);
532         if (*ptr || inact < 0) {
533             errmsg(M_INVALID, inactstr, "inactivity period");
534             exit(EX_BADARG);
535         }
536         call_pass = 1;
537     }
539     /* expiration string is a date, newer than today */
540     if (expire) {
541         if (*expire &&
542             valid_expire(expire, (time_t *)0) == INVALID) {
543             errmsg(M_INVALID, expire, "expiration date");
544             exit(EX_BADARG);
545         }

```

```

546         call_pass = 1;
547     }

549     if (nkeys > 0)
550         call_pass = 1;

552     /* that's it for validations - now do the work */

554     if (grps) {
555         /* redefine login's supplementary group memberships */
556         ret = edit_group(logname, new_logname, gidlist, 1);
557         if (ret != EX_SUCCESS) {
558             errmsg(M_UPDATE, "modified");
559             exit(ret);
560         }
561     }
562     if (projects) {
563         ret = edit_project(logname, (char *)NULL, projlist, 0);
564         if (ret != EX_SUCCESS) {
565             errmsg(M_UPDATE, "modified");
566             exit(ret);
567         }
568     }

571     if (!call_pass) exit(ret);

573     /* only get to here if need to call passmgmt */
574     /* set up arguments to passmgmt in nargv array */
575     nargv = malloc((30 + nkeys * 2) * sizeof(char *));

577     argindex = 0;
578     nargv[argindex++] = PASSMGMT;
579     nargv[argindex++] = "-m";      /* modify */

581     if (comment) { /* comment */
582         nargv[argindex++] = "-c";
583         nargv[argindex++] = comment;
584     }

586     if (dir) {
587         /* flags for home directory */
588         nargv[argindex++] = "-h";
589         nargv[argindex++] = dir;
590     }

592     if (group) {
593         /* set gid flag */
594         nargv[argindex++] = "-g";
595         (void) sprintf(gidstring, "%u", gid);
596         nargv[argindex++] = gidstring;
597     }

599     if (shell) { /* shell */
600         nargv[argindex++] = "-s";
601         nargv[argindex++] = shell;
602     }

604     if (inactstr) {
605         nargv[argindex++] = "-f";
606         nargv[argindex++] = inactstr;
607     }

609     if (expire) {
610         nargv[argindex++] = "-e";
611         nargv[argindex++] = expire;

```

```

612     }

614     if (uidstr) { /* set uid flag */
615         nargv[argindex++] = "-u";
616         (void) sprintf(uidstring, "%u", uid);
617         nargv[argindex++] = uidstring;
618     }

620     if (oflag) nargv[argindex++] = "-o";

622     if (new_logname) { /* redefine login name */
623         nargv[argindex++] = "-l";
624         nargv[argindex++] = new_logname;
625     }

627     if (nkeys > 0)
628         addkey_args(nargv, &argindex);

630     /* finally - login name */
631     nargv[argindex++] = logname;

633     /* set the last to null */
634     nargv[argindex++] = NULL;

636     /* now call passmgmt */
637     ret = PEX_FAILED;
638     for (tries = 3; ret != PEX_SUCCESS && tries--;) {
639         switch (ret = call_passmgmt(nargv)) {
640             case PEX_SUCCESS:
641                 break;
642             case PEX_HOSED_FILES:
643                 errmsg(M_HOSED_FILES);
644                 exit(EX_INCONSISTENT);
645                 break;
646             case PEX_SYNTAX:
647                 break;
648             case PEX_BADARG:
649                 /* should NEVER occur that passmgmt usage is wrong */
650                 if (is_role(usertype))
651                     errmsg(M_MRUSAGE);
652                 else
653                     errmsg(M_MUSAGE);
654                 exit(EX_SYNTAX);
655                 break;
656             case PEX_BADUID:
657                 /* uid in use - shouldn't happen print message anyway */
658                 errmsg(M_UID_USED, uid);
659                 exit(EX_ID_EXISTS);
660                 break;
661             case PEX_BADNAME:
662                 /* invalid loname */
663                 errmsg(M_USED, logname);
664                 exit(EX_NAME_EXISTS);
665                 break;
666             default:
667                 errmsg(M_UPDATE, "modified");
668                 exit(ret);
669                 break;
670         }
671     }
672     if (tries == 0) {

```

new/usr/src/cmd/oamuser/user/usermod.c

11

```
678         errmsg(M_UPDATE, "modified");
679     }
681     exit(ret);
682     /*NOTREACHED*/
683 }
unchanged_portion_omitted
```



```

125 #define ZONE_LINE \
126 "%6d %8d %5s %5s %3.3s%% %9s %3.3s%% %28s"

128 #define TOTAL_LINE \
129 "Total: %d processes, %d lwps, load averages: %3.2f, %3.2f, %3.2f"

131 /* global variables */

133 static char      *t_ulon;          /* termcap: start underline */
134 static char      *t_uloff;         /* termcap: end underline */
135 static char      *t_up;            /* termcap: cursor 1 line up */
136 static char      *t_eol;           /* termcap: clear end of line */
137 static char      *t_smcup;         /* termcap: cursor mvcap on */
138 static char      *t_rmcup;         /* termcap: cursor mvcap off */
139 static char      *t_home;          /* termcap: move cursor home */
140 static char      *movecur = NULL;  /* termcap: move up string */
141 static char      *empty_string = "\0"; /* termcap: empty string */
142 static uint_t    print_movecur = FALSE; /* print movecur or not */
143 static int       is_curses_on = FALSE; /* current curses state */

145 static table_t   pid_tbl = {0, 0, NULL}; /* selected processes */
146 static table_t   cpu_tbl = {0, 0, NULL}; /* selected processors */
147 static table_t   set_tbl = {0, 0, NULL}; /* selected processor sets */
148 static table_t   prj_tbl = {0, 0, NULL}; /* selected projects */
149 static table_t   tsk_tbl = {0, 0, NULL}; /* selected tasks */
150 static table_t   lgr_tbl = {0, 0, NULL}; /* selected lgroups */
151 static zonetbl_t zone_tbl = {0, 0, NULL}; /* selected zones */
152 static uidtbl_t  euid_tbl = {0, 0, NULL}; /* selected effective users */
153 static uidtbl_t  ruid_tbl = {0, 0, NULL}; /* selected real users */

155 static uint_t    total_procs;       /* total number of procs */
156 static uint_t    total_lwps;        /* total number of lwps */
157 static float     total_cpu;         /* total cpu usage */
158 static float     total_mem;         /* total memory usage */

160 static list_t    lwps;              /* list of lwps/processes */
161 static list_t    users;             /* list of users */
162 static list_t    tasks;            /* list of tasks */
163 static list_t    projects;         /* list of projects */
164 static list_t    zones;            /* list of zones */
165 static list_t    lgroups;          /* list of lgroups */

167 static volatile uint_t sigwinch = 0;
168 static volatile uint_t sigtstp = 0;
169 static volatile uint_t sigterm = 0;

171 static long      pagesize;

173 /* default settings */

175 static optdesc_t opts = {
176     5,                /* interval between updates, seconds */
177     15,               /* number of lines in top part */
178     5,                /* number of lines in bottom part */
179     -1,               /* number of iterations; infinitely */
180     OPT_PSINFO | OPT_FULLSCREEN | OPT_USEHOME | OPT_TERMCAP,
181     -1                /* sort in decreasing order */
182 };

```

unchanged portion omitted

```

350 /*
351 * A routine to display the contents of the list on the screen
352 */
353 static void
354 list_print(list_t *list)
355 {

```

```

356     lwp_info_t *lwp;
357     id_info_t *id;
358     char usr[4], sys[4], trp[4], tfl[4];
359     char dfl[4], lck[4], slp[4], lat[4];
360     char vcx[4], icx[4], scl[4], sig[4];
361     char psize[6], prssize[6], pmem[6], pcpu[6], ptime[12];
362     char pstate[7], pnice[4], ppri[4];
363     char pname[LOGNAME_MAX_ILLUMOS+1];
364     char pname[LOGNAME_MAX+1];
365     char projname[PROJNAME_MAX+1];
366     char zonename[ZONENAME_MAX+1];
367     float cpu, mem;
368     double loadavg[3] = {0, 0, 0};
369     int i, lwpid;

370     if (foreach_element(&set_tbl, &loadavg, psetloadavg) == 0) {
371         /*
372          * If processor sets aren't specified, we display system-wide
373          * load averages.
374          */
375         (void) getloadavg(loadavg, 3);
376     }

378     if (((opts.o_outpmode & OPT_UPDATE) || (opts.o_outpmode & OPT_DDATE)) &&
379         ((list->l_type == LT_LWPS) || !(opts.o_outpmode & OPT_SPLIT)))
380         print_timestamp();
381     if (opts.o_outpmode & OPT_TTY)
382         (void) putchar('\r');
383     (void) putp(t_ulon);

385     switch (list->l_type) {
386     case LT_PROJECTS:
387         if (opts.o_outpmode & OPT_LWPS)
388             (void) printf(PROJECT_HEADER_LWP);
389         else
390             (void) printf(PROJECT_HEADER_PROC);
391         break;
392     case LT_TASKS:
393         if (opts.o_outpmode & OPT_LWPS)
394             (void) printf(TASK_HEADER_LWP);
395         else
396             (void) printf(TASK_HEADER_PROC);
397         break;
398     case LT_ZONES:
399         if (opts.o_outpmode & OPT_LWPS)
400             (void) printf(ZONE_HEADER_LWP);
401         else
402             (void) printf(ZONE_HEADER_PROC);
403         break;
404     case LT_USERS:
405         if (opts.o_outpmode & OPT_LWPS)
406             (void) printf(USER_HEADER_LWP);
407         else
408             (void) printf(USER_HEADER_PROC);
409         break;
410     case LT_LWPS:
411         if (opts.o_outpmode & OPT_LWPS) {
412             if (opts.o_outpmode & OPT_PSINFO) {
413                 if (opts.o_outpmode & OPT_LGRP)
414                     (void) printf(PSINFO_HEADER_LWP_LGRP);
415                 else
416                     (void) printf(PSINFO_HEADER_LWP);
417             }
418             if (opts.o_outpmode & OPT_MSACCT)
419                 (void) printf(USAGE_HEADER_LWP);
420         } else {

```



```

526         ptime, pcpu,
527         (int)lwp->li_info.pr_lwp.pr_lgrp,
528         lwp->li_info.pr_fname, lwpid);
529     } else {
530         (void) printf(PSINFO_LINE,
531         (int)lwp->li_info.pr_pid, pname,
532         psize, prssize,
533         pstate, ppri, pnice,
534         psize, prssize, pstate, ppri, pnice,
535         ptime, pcpu,
536         lwp->li_info.pr_fname, lwpid);
537     }
538     (void) putp(t_eol);
539     (void) putchar('\n');
540 }
541 if (opts.o_outpmode & OPT_MSACCT) {
542     Format_pct(usr, lwp->li_usr, 4);
543     Format_pct(sys, lwp->li_sys, 4);
544     Format_pct(slp, lwp->li_slp, 4);
545     Format_num(vcx, lwp->li_vcx, 4);
546     Format_num(icx, lwp->li_icx, 4);
547     Format_num(scl, lwp->li_scl, 4);
548     Format_num(sig, lwp->li_sig, 4);
549     Format_pct(trp, lwp->li_trp, 4);
550     Format_pct(tfl, lwp->li_tfl, 4);
551     Format_pct(df1, lwp->li_df1, 4);
552     Format_pct(lck, lwp->li_lck, 4);
553     Format_pct(lat, lwp->li_lat, 4);
554     if (opts.o_outpmode & OPT_TTY)
555         (void) putchar('\r');
556     stripfname(lwp->li_info.pr_fname);
557     (void) printf(USAGE_LINE,
558     (int)lwp->li_info.pr_pid, pname,
559     usr, sys, trp, tfl, df1, lck,
560     slp, lat, vcx, icx, scl, sig,
561     lwp->li_info.pr_fname, lwpid);
562     (void) putp(t_eol);
563     (void) putchar('\n');
564 }
565     break;
566 }
567 }
568
569 if (opts.o_outpmode & OPT_TTY)
570     (void) putchar('\r');
571 if (opts.o_outpmode & OPT_TERMCAP) {
572     switch (list->l_type) {
573     case LT_PROJECTS:
574     case LT_USERS:
575     case LT_TASKS:
576     case LT_ZONES:
577         while (i++ < opts.o_nbottom) {
578             (void) putp(t_eol);
579             (void) putchar('\n');
580         }
581         break;
582     case LT_LWPS:
583         while (i++ < opts.o_ntop) {
584             (void) putp(t_eol);
585             (void) putchar('\n');
586         }
587     }
588 }
589
590 if (opts.o_outpmode & OPT_TTY)
591     (void) putchar('\r');

```

```

607     if ((opts.o_outpmode & OPT_SPLIT) && list->l_type == LT_LWPS)
608         return;
609
610     (void) printf(TOTAL_LINE, total_procs, total_lwps,
611     loadavg[LOADAVG_1MIN], loadavg[LOADAVG_5MIN],
612     loadavg[LOADAVG_15MIN]);
613     (void) putp(t_eol);
614     (void) putchar('\n');
615     if (opts.o_outpmode & OPT_TTY)
616         (void) putchar('\r');
617     (void) putp(t_eol);
618     (void) fflush(stdout);
619 }
620 _____unchanged_portion_omitted_____

```

```

*****
6709 Wed Apr 3 09:33:11 2013
new/usr/src/cmd/prstat/prtable.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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  *
27  * Portions Copyright 2009 Chad Mynhier
28  */

30 #include <proefs.h>
31 #include <unistd.h>
32 #include <stdlib.h>
33 #include <pwd.h>
34 #include <ctype.h>
35 #include <string.h>
36 #include <libintl.h>
37 #include <errno.h>
38 #include <zone.h>
39 #include <libzonecfg.h>

41 #include "prstat.h"
42 #include "prutil.h"
43 #include "prtable.h"

45 static plwp_t *plwp_tbl[PLWP_TBL_SZ];

47 void
48 lwpid_init()
49 {
50     (void) memset(&plwp_tbl, 0, sizeof (plwp_t *) * PLWP_TBL_SZ);
51 }
    unchanged_portion_omitted_

63 void
64 pwd_getname(uid_t uid, char *name, size_t length, int noresolve,
65             int termcap, size_t width)
66 {
67     struct passwd *pwd;
68     size_t n;

```

```

70     if (noresolve || (pwd = getpwuid(uid)) == NULL) {
71         (void) snprintf(name, length, "%u", uid);
72     } else {
73         n = strlen(pwd->pw_name);
74         if (termcap && n > width)
75             (void) snprintf(name, length, "%.s%c",
76                             width - 1, pwd->pw_name, '*');
77         else
78             (void) snprintf(name, length, "%s", pwd->pw_name);
79     }
80 }
    unchanged_portion_omitted_

```


new/usr/src/cmd/prstat/prtable.h

1

```
*****
2466 Wed Apr 3 09:33:11 2013
new/usr/src/cmd/prstat/prtable.h
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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  *
27  * Portions Copyright 2009 Chad Mynhier
28  */

30 #ifndef _PRTABLE_H
31 #define _PRTABLE_H

33 #ifdef __cplusplus
34 extern "C" {
35 #endif

37 #include <limits.h>
38 #include <zone.h>
39 #include "prstat.h"

41 #define PLWP_TBL_SZ 4096 /* hash table of plwp_t structures */
42 #define LWP_ACTIVE 1

44 typedef struct {
45     size_t t_size;
46     size_t t_nent;
47     long *t_list;
48 } table_t;
    unchanged_portion_omitted

75 extern void pwd_getname(uid_t, char *, size_t, int, int, size_t);
73 extern void pwd_getname(uid_t, char *, int, int);
76 extern void add_uid(uidtbl_t *, char *);
77 extern int has_uid(uidtbl_t *, uid_t);
78 extern void add_element(table_t *, long);
79 extern int has_element(table_t *, long);
80 extern void add_zone(zonetbl_t *, char *);
81 extern int has_zone(zonetbl_t *, zoneid_t);
82 extern void convert_zone(zonetbl_t *);
83 extern int foreach_element(table_t *, void *, void (*)(long, void *));
```

new/usr/src/cmd/prstat/prtable.h

2

```
84 extern void lwpid_init();
85 extern void lwpid_add(lwp_info_t *, pid_t, id_t);
86 extern lwp_info_t *lwpid_get(pid_t, id_t);
87 extern int lwpid_pidcheck(pid_t);
88 extern void lwpid_del(pid_t, id_t);
89 extern void lwpid_set_active(pid_t, id_t);
90 extern int lwpid_is_active(pid_t, id_t);

92 #ifdef __cplusplus
93 }
    unchanged_portion_omitted
```

```

*****
7551 Wed Apr 3 09:33:11 2013
new/usr/src/cmd/prstat/prutil.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 *
27 * Portions Copyright 2009 Chad Mynhier
28 */

30 #include <sys/types.h>
31 #include <sys/param.h>
32 #include <sys/resource.h>
33 #include <sys/priocntl.h>
34 #include <sys/rtpriocntl.h>
35 #include <sys/tspriocntl.h>
36 #include <zone.h>

38 #include <libintl.h>
39 #include <limits.h>
40 #include <wchar.h>
41 #include <unistd.h>
42 #include <string.h>
43 #include <stdlib.h>
44 #include <stdarg.h>
45 #include <stdio.h>
46 #include <stdio_ext.h>
47 #include <errno.h>
48 #include <ctype.h>
49 #include <poll.h>
50 #include <project.h>

52 #include "prfile.h"
53 #include "prstat.h"
54 #include "prutil.h"

56 static char PRG_FMT[] = "%s: ";
57 static char ERR_FMT[] = ": %s\n";
58 static char *progname;
59 static char projbuf[PROJECT_BUFSZ];

```

```

61 #define RLIMIT_NOFILE_MAX 32767

63 /*PRINTFLIKE1*/
64 void
65 Warn(char *format, ...)
66 {
67     int err = errno;
68     va_list alist;

70     if (progname != NULL)
71         (void) fprintf(stderr, PRG_FMT, progname);
72     va_start(alist, format);
73     (void) vfprintf(stderr, format, alist);
74     va_end(alist);
75     if (strchr(format, '\n') == NULL)
76         (void) fprintf(stderr, gettext(ERR_FMT), strerror(err));
77 }
unchanged_portion_omitted

282 void
283 getprojname(projid_t projid, char *str, size_t len, int noresolve,
284             int termcap, size_t width)
285 getprojname(projid_t projid, char *str, int len, int noresolve)
286 {
287     struct project proj;
288     size_t n;

289     if (noresolve || getprojbyid(projid, &proj, projbuf, PROJECT_BUFSZ) ==
290         NULL) {
291         (void) snprintf(str, len, "%-6d", (int)projid);
292     } else {
293         n = strlen(proj.pj_name);
294         if (termcap && n > width)
295             (void) snprintf(str, len, "%.*s%c", width - 1,
296                             proj.pj_name, '*');
297         else
298             (void) snprintf(str, len, "%-28s", proj.pj_name);
299     }
300 }

302 void
303 getzonename(zoneid_t zoneid, char *str, size_t len, int termcap, size_t width)
304 getzonename(zoneid_t zoneid, char *str, int len)
305 {
306     char zone_name[ZONENAME_MAX];
307     size_t n;

308     if (getzonenamebyid(zoneid, zone_name, sizeof (zone_name)) < 0) {
309         if (getzonenamebyid(zoneid, zone_name, sizeof (zone_name)) < 0)
310             (void) snprintf(str, len, "%-6d", (int)zoneid);
311     } else {
312         n = strlen(zone_name);
313         if (termcap && n > width)
314             (void) snprintf(str, len, "%.*s%c", width - 1,
315                             zone_name, '*');
316         else
317             (void) snprintf(str, len, "%-28s", zone_name);
318     }
}
unchanged_portion_omitted

```

new/usr/src/cmd/prstat/prutil.h

1

```
*****
1878 Wed Apr 3 09:33:11 2013
new/usr/src/cmd/prstat/prutil.h
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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  *
27  * Portions Copyright 2009 Chad Mynhier
28  */

30 #ifndef _PRUTIL_H
31 #define _PRUTIL_H

33 #include <sys/processor.h>
34 #include <sys/types.h>

36 #ifdef __cplusplus
37 extern "C" {
38 #endif

40 extern void Die(char *, ...);
41 extern void Warn(char *, ...);
42 extern void Progname(char *);
43 extern void Usage();
44 extern int Atoi(char *);
45 extern void Format_size(char *, size_t, int);
46 extern void Format_pct(char *, float, int);
47 extern void Format_num(char *, int, int);
48 extern void Format_time(char *, ulong_t, int);
49 extern void Format_state(char *, char, processorid_t, int);
50 extern void *Realloc(void *, size_t);
51 extern void *Malloc(size_t);
52 extern void *Zalloc(size_t);
53 extern int Setrlimit();
54 extern void Prioctl(char *);
55 extern void getprojname(projid_t, char *, size_t, int, int, size_t);
56 extern void getzonename(projid_t, char *, size_t, int, size_t);
53 extern void getprojname(projid_t, char *, int, int);
54 extern void getzonename(projid_t, char *, int);
57 extern void stripfname(char *);
```

new/usr/src/cmd/prstat/prutil.h

2

```
59 #ifdef __cplusplus
60 }
_____unchanged_portion_omitted_
```

new/usr/src/cmd/ps/ps.c

1

```
*****
57960 Wed Apr  3 09:33:11 2013
new/usr/src/cmd/ps/ps.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 */

22 /*
23 * Copyright (c) 2013 Gary Mills
24 *
25 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26 * Use is subject to license terms.
27 */

29 /*
30 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
31 */

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

36 /*
37 * ps -- print things about processes.
38 */
39 #include <stdio.h>
40 #include <ctype.h>
41 #include <string.h>
42 #include <errno.h>
43 #include <fcntl.h>
44 #include <pwd.h>
45 #include <grp.h>
46 #include <sys/types.h>
47 #include <sys/stat.h>
48 #include <sys/mkdev.h>
49 #include <unistd.h>
50 #include <stdlib.h>
51 #include <limits.h>
52 #include <dirent.h>
53 #include <sys/signal.h>
54 #include <sys/fault.h>
55 #include <sys/syscall.h>
56 #include <sys/time.h>
57 #include <procfs.h>
58 #include <locale.h>
59 #include <wctype.h>
60 #include <wchar.h>
```

new/usr/src/cmd/ps/ps.c

2

```
61 #include <libw.h>
62 #include <stdarg.h>
63 #include <sys/proc.h>
64 #include <sys/pset.h>
65 #include <project.h>
66 #include <zone.h>

68 #define min(a, b)      ((a) > (b) ? (b) : (a))
69 #define max(a, b)      ((a) < (b) ? (b) : (a))

71 #define NTTYS    20      /* initial size of table for -t option */
72 #define SIZ      30      /* initial size of tables for -p, -s, -g, -h and -z */

74 /*
75 * Size of buffer holding args for t, p, s, g, u, U, G, z options.
76 * Set to ZONENAME_MAX, the minimum value needed to allow any
77 * zone to be specified.
78 */
79 #define ARGSIZ ZONENAME_MAX

81 #define MAXUGNAME (LOGNAME_MAX_ILLUMOS+2) /* max chars in a user/group */
82 /* name or printed u/g id */
79 #define MAXUGNAME 10 /* max chars in a user/group name or printed u/g id */

84 /* Structure for storing user or group info */
85 struct ugdata {
86     id_t    id;          /* numeric user-id or group-id */
87     char    name[MAXUGNAME+1]; /* user/group name, null terminated */
88 };
      unchanged_portion_omitted
```

```

*****
5363 Wed Apr 3 09:33:12 2013
new/usr/src/cmd/pwck/pwck.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 2007 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      */

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

32 #include <sys/types.h>
33 #include <sys/param.h>
34 #include <sys/signal.h>
35 #include <sys/sysmacros.h>
36 #include <sys/stat.h>
37 #include <stdio.h>
38 #include <stdlib.h>
39 #include <string.h>
40 #include <ctype.h>
41 #include <locale.h>
42 #include <errno.h>
43 #include <unistd.h>
44 #include <limits.h>

46 #define ERROR1      "Too many/few fields"
47 #define ERROR2      "Bad character(s) in logname"
48 #define ERROR2a     "First char in logname not alphabetic"
49 #define ERROR2b     "Logname field NULL"
50 #define ERROR2c     "Logname contains no lower-case letters"
51 #define ERROR3      "Logname too long/short"
52 #define ERROR4      "Invalid UID"
53 #define ERROR5      "Invalid GID"
54 #define ERROR6      "Login directory not found"
55 #define ERROR6a     "Login directory null"
56 #define ERROR7      "Optional shell file not found"

58 static int eflag, code = 0;

```

```

59 static int badc;
60 static int lc;
61 static char buf[512];
62 static void error(char *);

64 int
65 main(int argc, char **argv)
66 {
67     int delim[512];
68     char logbuf[512];
69     FILE *fptr;
70     struct stat obuf;
71     uid_t uid;
72     gid_t gid;
73     int i, j, colons;
74     char *pw_file;
75     struct stat stat_buf;
76     char *str, *lastc;

78     (void) setlocale(LC_ALL, "");

80 #if !defined(TEXT_DOMAIN)      /* Should be defined by cc -D */
81 #define TEXT_DOMAIN "SYS_TEST"
82 #endif
83     (void) textdomain(TEXT_DOMAIN);

85     if (argc == 1)
86         pw_file = "/etc/passwd";
87     else
88         pw_file = argv[1];

90     if ((fptr = fopen(pw_file, "r")) == NULL) {
91         (void) fprintf(stderr, gettext("cannot open %s\n"), pw_file);
92         exit(1);
93     }

95     if (fstat(fileno(fp), &stat_buf) < 0) {
96         (void) fprintf(stderr, gettext("fstat failed for %s\n"),
97             pw_file);
98         (void) fclose(fp);
99         exit(1);
100    }

102    if (stat_buf.st_size == 0) {
103        (void) fprintf(stderr, gettext("file %s is empty\n"), pw_file);
104        (void) fclose(fp);
105        exit(1);
106    }

108    while (fgets(buf, sizeof(buf), fptr) != NULL) {

110        colons = 0;
111        badc = 0;
112        lc = 0;
113        eflag = 0;

115        /* Check that entry is not a nameservice redirection */

117        if (buf[0] == '+' || buf[0] == '-') {
118            /*
119             * Should set flag here to allow special case checking
120             * in the rest of the code,
121             * but for now, we'll just ignore this entry.
122             */
123            continue;
124        }

```

```

126         /* Check number of fields */
128     for (i = 0; buf[i] != NULL; i++)
129         if (buf[i] == ':') {
130             delim[colons] = i;
131             ++colons;
132         }
134     if (colons != 6) {
135         error(ERROR1);
136         continue;
137     }
138     delim[6] = i - 1;
139     delim[7] = NULL;
141     /*
142     * Check the first char is alpha; the rest alphanumeric;
143     * and that the name does not consist solely of uppercase
144     * alpha chars
145     */
146     if (buf[0] == ':')
147         error(ERROR2b);
148     else if (!isalpha(buf[0]))
149         error(ERROR2a);
151     for (i = 0; buf[i] != ':'; i++) {
152         if (!isalnum(buf[i]) &&
153             buf[i] != '-' &&
154             buf[i] != '_' &&
155             buf[i] != '.')
156             badc++;
157         else if (islower(buf[i]))
158             lc++;
159     }
160     if (lc == 0)
161         error(ERROR2c);
162     if (badc > 0)
163         error(ERROR2);
165     /* Check for valid number of characters in logname */
166     if (i <= 0 || i > LOGNAME_MAX_ILLUMOS)
167     if (i <= 0 || i > 8)
168         error(ERROR3);
170     /* Check that UID is numeric and <= MAXUID */
172     errno = 0;
173     str = &buf[delim[1] + 1];
174     uid = strtol(str, &lastc, 10);
175     if (lastc != str + (delim[2] - delim[1]) - 1 ||
176         uid > MAXUID || errno == ERANGE)
177         error(ERROR4);
179     /* Check that GID is numeric and <= MAXUID */
181     errno = 0;
182     str = &buf[delim[2] + 1];
183     gid = strtol(str, &lastc, 10);
184     if (lastc != str + (delim[3] - delim[2]) - 1 ||
185         gid > MAXUID || errno == ERANGE)
186         error(ERROR5);
188     /* Check initial working directory */

```

```

190         for (j = 0, i = (delim[4] + 1); i < delim[5]; j++, i++)
191             logbuf[j] = buf[i];
192         logbuf[j] = '\0';
194         if (logbuf[0] == NULL)
195             error(ERROR6a);
196         else if ((stat(logbuf, &obuf)) == -1)
197             error(ERROR6);
199     /* Check program to use as shell */
201     if ((buf[(delim[5] + 1)]) != '\n') {
203         for (j = 0, i = (delim[5] + 1); i < delim[6]; j++, i++)
204             logbuf[j] = buf[i];
205         logbuf[j] = '\0';
207         if (strcmp(logbuf, "**") == 0) /* subsystem login */
208             continue;
210         if ((stat(logbuf, &obuf)) == -1)
211             error(ERROR7);
213         for (j = 0; j < 512; j++)
214             logbuf[j] = NULL;
215     }
216     }
217     (void) fclose(fp);
218     return (code);
219 }

```

unchanged portion omitted

```

*****
18980 Wed Apr 3 09:33:12 2013
new/usr/src/cmd/w/w.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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  * utmpx defines wider fields for user and line. For compatibility of output,
76  * we are limiting these to the old maximums in utmp. Define UTMPX_NAMELEN
77  * to use the full lengths.
78  */
79 #ifndef UTMPX_NAMELEN
80 #define UTMPX_NAMELEN 12
81 #endif

82 #define LOGIN_WIDTH 8
83 #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 PROC_DIR "/proc"
93 #define INITPROC (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     struct uproc *p_sibling; /* sibling pointer */
113     int p_pgrp; /* pgrp link */
114     struct uproc *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, char *);
131 static void        prtat(time_t *time);
132 static void        checkampm(char *str);

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

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

156 int
157 main(int argc, char *argv[])
158 {
159     struct utmpx      *ut;
160     struct utmpx      *utmpbegin;
161     struct utmpx      *utmpend;
162     struct utmpx      *utp;
163     struct uproc      *up, *parent, *pgrp;
164     struct psinfo      info;
165     struct sigaction  actinfo[ACTSIZE];
166     struct pstatus      statinfo;
167     size_t             size;
168     struct stat        sbuf;
169     DIR                *dirp;
170     struct dirent      *dp;
171     char               pname[64];
172     char               *fname;
173     int                procfd;
174     char               *cp;
175     int                i;
176     int                days, hrs, mins;
177     int                entries;
178     double             loadavg[3];

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

```

```

184         (char *)NULL);

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

192         login = (argv[0][0] == '-');
193         cp = strrchr(argv[0], '/');
194         firstchar = login ? argv[0][1] : (cp == 0) ? argv[0][0] : cp[1];
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;
209                         case 's':
210                             lflag = 0;
211                             break;

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

218                         default:
219                             (void) fprintf(stderr, gettext(
220                                 "%s: bad flag %s\n"),
221                                 prog, argv[1]);
222                             exit(1);
223                     }
224                 }
225             } else {
226                 if (!isalnum(argv[1][0]) || argc > 2) {
227                     (void) fprintf(stderr, gettext(
228                         "usage: %s [ -hlsuw ] [ user ]\n"), prog);
229                     exit(1);
230                 } else
231                     sel_user = argv[1];
232             }
233             argc--; argv++;
234         }

236         /*
237          * read the UTMP_FILE (contains information about each logged in user)
238          */
239         if (stat(UTMPX_FILE, &sbuf) == ERR) {
240             (void) fprintf(stderr, gettext("%s: stat error of %s: %s\n"),
241                 prog, UTMPX_FILE, strerror(errno));
242             exit(1);
243         }
244         entries = sbuf.st_size / sizeof (struct futmpx);
245         size = sizeof (struct utmpx) * entries;
246         if ((ut = malloc(size)) == NULL) {
247             (void) fprintf(stderr, gettext("%s: malloc error of %s: %s\n"),
248                 prog, UTMPX_FILE, strerror(errno));
249             exit(1);

```



```

250     }
252     (void) utmpxname(UTMPX_FILE);
254     utmpbegin = ut;
255     utmpend = (struct utmpx *)((char *)utmpbegin + size);
257     setutxent();
258     while ((ut < utmpend) && ((utp = getutxent()) != NULL))
259         (void) memcpy(ut++, utp, sizeof (*ut));
260     endutxent();
262     (void) time(&now);      /* get current time */
264     if (header) { /* print a header */
265         prtat(&now);
266         for (ut = utmpbegin; ut < utmpend; ut++) {
267             if (ut->ut_type == USER_PROCESS) {
268                 if (!nonuser(*ut))
269                     nusers++;
270             } else if (ut->ut_type == BOOT_TIME) {
271                 uptime = now - ut->ut_xtime;
272                 uptime += 30;
273                 days = uptime / (60*60*24);
274                 uptime %= (60*60*24);
275                 hrs = uptime / (60*60);
276                 uptime %= (60*60);
277                 mins = uptime / 60;
279                 PRINTF((gettext(" up")));
280                 if (days > 0)
281                     PRINTF((gettext(
282                         " %d day(s)", days));
283                 if (hrs > 0 && mins > 0) {
284                     PRINTF((" %2d:%02d", hrs, mins));
285                 } else {
286                     if (hrs > 0)
287                         PRINTF((gettext(
288                             " %d hr(s)", hrs));
289                     if (mins > 0)
290                         PRINTF((gettext(
291                             " %d min(s)", mins));
292                 }
293             }
294         }
296         ut = utmpbegin; /* rewind utmp data */
297         PRINTF(((nusers == 1) ?
298             gettext(" %d user") : gettext(" %d users")), nusers));
299         /*
300          * Print 1, 5, and 15 minute load averages.
301          */
302         (void) getloadavg(loadavg, 3);
303         PRINTF((gettext(", load average: %.2f, %.2f, %.2f\n"),
304             loadavg[LOADAVG_1MIN], loadavg[LOADAVG_5MIN],
305             loadavg[LOADAVG_15MIN]));
307         if (firstchar == 'u') /* uptime command */
308             exit(0);
310         if (lflag) {
311             PRINTF((dcgettext(NULL, "User      tty
312             "login@ idle JCPU PCPU what\n", LC_TIME)));
313         } else {
314             PRINTF((dcgettext(NULL,
315             "User      tty          idle  what\n", LC_TIME)));

```

```

316     }
318     if (fflush(stdout) == EOF) {
319         perror((gettext("%s: fflush failed\n"), prog));
320         exit(1);
321     }
322 }
324 /*
325  * loop through /proc, reading info about each process
326  * and build the parent/child tree
327  */
328 if (!(dirp = opendir(PROCDIR))) {
329     (void) fprintf(stderr, gettext("%s: could not open %s: %s\n"),
330         prog, PROCDIR, strerror(errno));
331     exit(1);
332 }
334 while ((dp = readdir(dirp)) != NULL) {
335     if (dp->d_name[0] == '.')
336         continue;
337 retry:
338     (void) sprintf(pname, "%s/%s/", PROCDIR, dp->d_name);
339     fname = pname + strlen(pname);
340     (void) strcpy(fname, "psinfo");
341     if ((procfd = open(pname, O_RDONLY)) < 0)
342         continue;
343     if (read(procfd, &info, sizeof (info)) != sizeof (info)) {
344         int err = errno;
345         (void) close(procfd);
346         if (err == EAGAIN)
347             goto retry;
348         if (err != ENOENT)
349             (void) fprintf(stderr, gettext(
350                 "%s: read() failed on %s: %s\n"),
351                 prog, pname, strerror(err));
352         continue;
353     }
354     (void) close(procfd);
356     up = findhash(info.pr_pid);
357     up->p_ttyd = info.pr_ttydev;
358     up->p_state = (info.pr_nlwp == 0? ZOMBIE : RUNNING);
359     up->p_time = 0;
360     up->p_ctime = 0;
361     up->p_igintr = 0;
362     (void) strncpy(up->p_comm, info.pr_fname,
363         sizeof (info.pr_fname));
364     up->p_args[0] = 0;
366     if (up->p_state != NONE && up->p_state != ZOMBIE) {
367         (void) strcpy(fname, "status");
369         /* now we need the proc_owner privilege */
370         (void) __priv_bracket(PRIV_ON);
372         procfd = open(pname, O_RDONLY);
374         /* drop proc_owner privilege after open */
375         (void) __priv_bracket(PRIV_OFF);
377         if (procfd < 0)
378             continue;
380         if (read(procfd, &statinfo, sizeof (statinfo))
381             != sizeof (statinfo)) {

```

```

382         int err = errno;
383         (void) close(procfd);
384         if (err == EAGAIN)
385             goto retry;
386         if (err != ENOENT)
387             (void) fprintf(stderr, gettext(
388                 "%s: read() failed on %s: %s\n"),
389                 prog, pname, strerror(err));
390         continue;
391     }
392     (void) close(procfd);
393
394     up->p_time = statinfo.pr_utime.tv_sec +
395         statinfo.pr_stime.tv_sec; /* seconds */
396     up->p_ctime = statinfo.pr_cutime.tv_sec +
397         statinfo.pr_cstime.tv_sec;
398
399     (void) strcpy(fname, "sigact");
400
401     /* now we need the proc_owner privilege */
402     (void) __priv_bracket(PRIV_ON);
403
404     procfd = open(pname, O_RDONLY);
405
406     /* drop proc_owner privilege after open */
407     (void) __priv_bracket(PRIV_OFF);
408
409     if (procfd < 0)
410         continue;
411
412     if (read(procfd, actinfo, sizeof (actinfo))
413         != sizeof (actinfo)) {
414         int err = errno;
415         (void) close(procfd);
416         if (err == EAGAIN)
417             goto retry;
418         if (err != ENOENT)
419             (void) fprintf(stderr, gettext(
420                 "%s: read() failed on %s: %s\n"),
421                 prog, pname, strerror(err));
422         continue;
423     }
424     (void) close(procfd);
425
426     up->p_igintr =
427         actinfo[SIGINT-1].sa_handler == SIG_IGN &&
428         actinfo[SIGQUIT-1].sa_handler == SIG_IGN;
429
430     /*
431      * Process args.
432      */
433     up->p_args[0] = 0;
434     clnarglist(info.pr_psargs);
435     (void) strcat(up->p_args, info.pr_psargs);
436     if (up->p_args[0] == 0 ||
437         up->p_args[0] == '-' && up->p_args[1] <= ' ' ||
438         up->p_args[0] == '?') {
439         (void) strcat(up->p_args, "(");
440         (void) strcat(up->p_args, up->p_comm);
441         (void) strcat(up->p_args, ")");
442     }
443 }
444
445 /*
446  * link pgrp together in case parents go away
447  * Pgrp chain is a single linked list originating

```

```

448     * from the pgrp leader to its group member.
449     */
450     if (info.pr_pgid != info.pr_pid) { /* not pgrp leader */
451         pgrp = findhash(info.pr_pgid);
452         up->p_pgrppl = pgrp->p_pgrppl;
453         pgrp->p_pgrppl = up;
454     }
455     parent = findhash(info.pr_ppid);
456
457     /* if this is the new member, link it in */
458     if (parent->p_upid != INITPROCESS) {
459         if (parent->p_child) {
460             up->p_sibling = parent->p_child;
461             up->p_child = 0;
462         }
463         parent->p_child = up;
464     }
465 }
466
467 /* revert to non-privileged user after opening */
468 (void) __priv_relinquish();
469
470 (void) closedir(dirp);
471 (void) time(&now); /* get current time */
472
473 /*
474  * loop through utmpx file, printing process info
475  * about each logged in user
476  */
477 for (ut = utmpbegin; ut < utmpend; ut++) {
478     if (ut->ut_type != USER_PROCESS)
479         continue;
480     if (sel_user && strcmp(ut->ut_name, sel_user, NMAX) != 0)
481         continue; /* we're looking for somebody else */
482
483     /* print login name of the user */
484     PRINTF(("%-*.s", LOGIN_WIDTH, NMAX, ut->ut_name));
485     PRINTF(("%-*.s", NMAX, NMAX, ut->ut_name));
486
487     /* print tty user is on */
488     if (lflag) {
489         PRINTF(("%-*.s", LINE_WIDTH, LMAX, ut->ut_line));
490         PRINTF(("%-*.s", LMAX, LMAX, ut->ut_line));
491     } else {
492         if (ut->ut_line[0] == 'p' && ut->ut_line[1] == 't' &&
493             ut->ut_line[2] == 's' && ut->ut_line[3] == '/') {
494             PRINTF(("%-*.3s", LMAX, &ut->ut_line[4]));
495         } else {
496             PRINTF(("%-*.s", LINE_WIDTH, LMAX,
497                 ut->ut_line));
498             PRINTF(("%-*.s", LMAX, LMAX, ut->ut_line));
499         }
500     }
501
502     /* print when the user logged in */
503     if (lflag) {
504         time_t tim = ut->ut_xtime;
505         prtat(&tim);
506     }
507
508     /* print idle time */
509     idle = findidle(ut->ut_line);
510     if (idle >= 36 * 60) {
511         PRINTF((dcgettext(NULL, "%2ddays", LC_TIME),
512             (idle + 12 * 60) / (24 * 60)));
513     } else

```

```
511         prttime(idle, " ");
512         showtotals(findhash(ut->ut_pid));
513     }
514     if (fclose(stdout) == EOF) {
515         perror((gettext("%s: fclose failed"), prog));
516         exit(1);
517     }
518     return (0);
519 }
```

unchanged portion omitted

new/usr/src/cmd/wall/wall.c

1

```
*****
11107 Wed Apr  3 09:33:12 2013
new/usr/src/cmd/wall/wall.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
23 /*      All Rights Reserved      */

27 /*
28 * Copyright 1988-2003 Sun Microsystems, Inc.  All rights reserved.
29 * Use is subject to license terms.
30 */

32 /*
33 * Copyright 2012 Joyent, Inc. All rights reserved.
34 *
35 * Copyright (c) 2013 Gary Mills
36 */

38 #include <signal.h>
39 #include <stdio.h>
40 #include <stdlib.h>
41 #include <grp.h>
42 #include <sys/types.h>
43 #include <unistd.h>
44 #include <string.h>
45 #include <ctype.h>
46 #include <sys/stat.h>
47 #include <utmpx.h>
48 #include <sys/utsname.h>
49 #include <dirent.h>
50 #include <pwd.h>
51 #include <fcntl.h>
52 #include <time.h>
53 #include <errno.h>
54 #include <locale.h>
55 #include <syslog.h>
56 #include <sys/wait.h>
57 #include <limits.h>
58 #include <libzonecfg.h>
59 #include <zone.h>
60 #include <sys/contract/process.h>
```

new/usr/src/cmd/wall/wall.c

2

```
61 #include <libcontract.h>
62 #include <sys/ctfs.h>

64 /*
65 * Use the full lengths from utmpx for user and line.
66 * utmpx defines wider fields for user and line.  For compatibility of output,
67 * we are limiting these to the old maximums in utmp.  Define UTMPX_NAMELEN
68 * to use the full lengths.
69 */
70 #define NMAX      (sizeof (((struct utmpx *)0)->ut_user))
71 #define LMAX      (sizeof (((struct utmpx *)0)->ut_line))
72 #ifndef UTMPX_NAMELEN
73 #define XXX - utmp -fix name length */
74 #define NMAX      (_POSIX_LOGIN_NAME_MAX - 1)
75 #define LMAX      12
76 #else /* UTMPX_NAMELEN */
77 #define NMAX      (sizeof (((struct utmpx *)0)->ut_user))
78 #define LMAX      (sizeof (((struct utmpx *)0)->ut_line))
79 #endif /* UTMPX_NAMELEN */

80 static char      mesg[3000];
81 static char      *infile;
82 static int       gflag;
83 static struct    group *pgrp;
84 static char      *grpname;
85 static char      line[MAXNAMLEN+1] = "???";
86 static char      systm[MAXNAMLEN+1];
87 static time_t    tloc;
88 static struct    utsname utsn;
89 static char      who[NMAX+1] = "???";
90 static char      who[9] = "???";
91 static char      time_buf[50];
92 #define DATE_FMT  "%a %b %e %H:%M:%S"

93 static void sendmes(struct utmpx *, zoneid_t);
94 static void sendmes_tozone(zoneid_t, int);
95 static int chkgrp(char *);
96 static char *copy_str_till(char *, char *, char, int);

97 static int init_template(void);
98 int contract_abandon_id(ctid_t);

99 int
100 main(int argc, char *argv[])
101 {
102     FILE *f;
103     char *ptr, *start;
104     struct passwd *pwd;
105     char *term_name;
106     int c;
107     int aflag = 0;
108     int errflg = 0;
109     int zflg = 0;
110     int Zflg = 0;

111     char *zonename = NULL;
112     zoneid_t *zoneidlist = NULL;
113     uint_t nzids_saved, nzids = 0;

114     (void) setlocale(LC_ALL, "");

115     while ((c = getopt(argc, argv, "g:az:Z")) != EOF)
116         switch (c) {
117             case 'a':
118                 aflag++;
119                 break;
120         }
```

```

115     case 'g':
116         if (gflag) {
117             (void) fprintf(stderr,
118                 "Only one group allowed\n");
119             return (1);
120         }
121         if ((pgrp = getgrnam(grpname = optarg)) == NULL) {
122             (void) fprintf(stderr, "Unknown group %s\n",
123                 grpname);
124             return (1);
125         }
126         gflag++;
127         break;
128     case 'z':
129         zflag++;
130         zonename = optarg;
131         if (getzoneidbyname(zonename) == -1) {
132             (void) fprintf(stderr, "Specified zone %s
133                 "is invalid", zonename);
134             return (1);
135         }
136         break;
137     case 'Z':
138         Zflag++;
139         break;
140     case '?':
141         errflg++;
142         break;
143     }
144
145     if (errflg) {
146         (void) fprintf(stderr,
147             "Usage: wall [-a] [-g group] [-z zone] [-Z] [files...]\n");
148         return (1);
149     }
150
151     if (zflag && Zflag) {
152         (void) fprintf(stderr, "Cannot use -z with -Z\n");
153         return (1);
154     }
155
156     if (optind < argc)
157         infile = argv[optind];
158
159     if (uname(&utsn) == -1) {
160         (void) fprintf(stderr, "wall: uname() failed, %s\n",
161             strerror(errno));
162         return (2);
163     }
164     (void) strcpy(system, utsn.nodename);
165
166     /*
167     * Get the name of the terminal wall is running from.
168     */
169
170     if ((term_name = ttyname(fileno(stderr))) != NULL) {
171         /*
172         * skip the leading "/dev/" in term_name
173         */
174         (void) strncpy(line, &term_name[5], sizeof (line) - 1);
175     }
176
177     if (who[0] == '?') {
178         if (pwd = getpwuid(getuid()))
179             (void) strncpy(&who[0], pwd->pw_name, sizeof (who));
180     }

```

```

182     f = stdin;
183     if (infile) {
184         f = fopen(infile, "r");
185         if (f == NULL) {
186             (void) fprintf(stderr, "Cannot open %s\n", infile);
187             return (1);
188         }
189     }
190
191     start = &mesg[0];
192     ptr = start;
193     while ((ptr - start) < 3000) {
194         size_t n;
195
196         if (fgets(ptr, &mesg[sizeof (mesg)] - ptr, f) == NULL)
197             break;
198         if ((n = strlen(ptr)) == 0)
199             break;
200         ptr += n;
201     }
202     (void) fclose(f);
203
204     /*
205     * If the request is from the rwall daemon then use the caller's
206     * name and host. We determine this if all of the following is true:
207     * 1) First 5 characters are "From "
208     * 2) Next non-white characters are of the form "name@host:"
209     */
210     if (strncmp(line, "???" ) == 0) {
211         char rwho[MAXNAMLEN+1];
212         char rsystem[MAXNAMLEN+1];
213         char *cp;
214
215         if (strncmp(mesg, "From ", 5) == 0) {
216             cp = &mesg[5];
217             cp = copy_str_till(rwho, cp, '@', MAXNAMLEN + 1);
218             if (rwho[0] != '\0') {
219                 cp = copy_str_till(rsystem, ++cp, ':',
220                     MAXNAMLEN + 1);
221                 if (rsystem[0] != '\0') {
222                     (void) strcpy(system, rsystem);
223                     (void) strncpy(rwho, who,
224                         sizeof (who));
225                     (void) strncpy(rwho, who, 9);
226                     (void) strcpy(line, "rpc.rwalld");
227                 }
228             }
229         }
230         (void) time(&tloc);
231         (void) strftime(time_buf, sizeof (time_buf),
232             DATE_FMT, localtime(&tloc));
233
234         if (zflag != 0) {
235             if ((zoneidlist =
236                 malloc(sizeof (zoneid_t))) == NULL ||
237                 (*zoneidlist = getzoneidbyname(zonename)) == -1)
238                 return (errno);
239             nzids = 1;
240         } else if (Zflag != 0) {
241             if (zone_list(NULL, &nzids) != 0)
242                 return (errno);
243         }
244         nzids *= 2;
245         if ((zoneidlist = malloc(nzids * sizeof (zoneid_t))) == NULL)

```

```

246         exit(errno);
247         nzids_saved = nzids;
248         if (zone_list(zoneidlist, &nzids) != 0) {
249             (void) free(zoneidlist);
250             return (errno);
251         }
252         if (nzids > nzids_saved) {
253             free(zoneidlist);
254             goto again;
255         }
256     }
257     if (zflg || Zflg) {
258         for (; nzids > 0; --nzids)
259             sendmes_tozone(zoneidlist[nzids-1], aflag);
260         free(zoneidlist);
261     } else
262         sendmes_tozone(getzoneid(), aflag);
264     return (0);
265 }

```

unchanged_portion_omitted

```

329 /*
330 * Note to future maintainers: with the change of wall to use the
331 * getutxent() API, the forked children (created by this function)
332 * must call _exit as opposed to exit. This is necessary to avoid
333 * unwanted fflushing of getutxent's stdio stream (caused by atexit
334 * processing).
335 */
336 static void
337 sendmes(struct utmpx *p, zoneid_t zid)
338 {
339     int i;
340     char *s;
341     static char device[LMAX + 6];
342     char *bp;
343     int ibp;
344     FILE *f;
345     int fd, tmpl_fd;
346     boolean_t zoneenter = B_FALSE;
348     if (zid != getzoneid()) {
349         zoneenter = B_TRUE;
350         tmpl_fd = init_template();
351         if (tmpl_fd == -1) {
352             (void) fprintf(stderr, "Could not initialize "
353                 "process contract");
354             return;
355         }
356     }
358     while ((i = (int)fork()) == -1) {
359         (void) alarm(60);
360         (void) wait((int *)0);
361         (void) alarm(0);
362     }
364     if (i)
365         return;
367     if (zoneenter && zone_enter(zid) == -1) {
368         char zonename[ZONENAME_MAX];
369         (void) getzonenamebyid(zid, zonename, ZONENAME_MAX);
370         (void) fprintf(stderr, "Could not enter zone "
371             "%s\n", zonename);
372     }

```

```

373     if (zoneenter)
374         (void) ct_tmpl_clear(tmpl_fd);
376     if (gflag)
377         if (!chkgrp(p->ut_user))
378             _exit(0);
380     (void) signal(SIGHUP, SIG_IGN);
381     (void) alarm(60);
382     s = &device[0];
383     (void) snprintf(s, sizeof (device), "/dev/%.*s", LMAX, p->ut_line);
385     /* check if the device is really a tty */
386     if ((fd = open(s, O_WRONLY|O_NOCTTY|O_NONBLOCK)) == -1) {
387         (void) fprintf(stderr, "Cannot send to %.*s on %s\n",
388             NMAX, p->ut_user, s);
389         perror("open");
390         (void) fflush(stderr);
391         _exit(1);
392     } else {
393         if (!isatty(fd)) {
394             (void) fprintf(stderr,
395                 "Cannot send to device %.*s %s\n",
396                 LMAX, p->ut_line,
397                 "because it's not a tty");
398             openlog("wall", 0, LOG_AUTH);
399             syslog(LOG_CRIT, "%.*s in utmpx is not a tty\n",
400                 LMAX, p->ut_line);
401             closelog();
402             (void) fflush(stderr);
403             _exit(1);
404         }
405     }
406 #ifdef DEBUG
407     (void) close(fd);
408     f = fopen("wall.debug", "a");
409 #else
410     f = fdopen(fd, "w");
411 #endif
412     if (f == NULL) {
413         (void) fprintf(stderr, "Cannot send to %.*s on %s\n",
414             NMAX, &p->ut_user[0], s);
415         perror("open");
416         (void) fflush(stderr);
417         _exit(1);
418     }
419     (void) fprintf(f,
420         "\07\07\07Broadcast Message from %s (%s) on %s %19.19s",
421         who, line, system, time_buf);
422     if (gflag)
423         (void) fprintf(f, " to group %s", grpname);
424     (void) fprintf(f, "... \n");
425 #ifdef DEBUG
426     (void) fprintf(f, "DEBUG: To %.*s on %s\n", NMAX, p->ut_user, s);
427     (void) fprintf(f, "DEBUG: To %.*s on %s\n", p->ut_user, s);
428 #endif
429     i = strlen(msg);
430     for (bp = msg; --i >= 0; bp++) {
431         ibp = (unsigned int)((unsigned char) *bp);
432         if (*bp == '\n')
433             (void) putc('\r', f);
434         if (isprint(ibp) || *bp == '\r' || *bp == '\013' ||
435             *bp == ' ' || *bp == '\t' || *bp == '\n' || *bp == '\007') {
436             (void) putc(*bp, f);
437         } else {
438             if (!isascii(*bp)) {

```

```
438         (void) fputs("M-", f);
439         *bp = toascii(*bp);
440     }
441     if (iscntrl(*bp)) {
442         (void) putc('^', f);
443         (void) putc(*bp + 0100, f);
444     }
445     else
446         (void) putc(*bp, f);
447 }
449 if (*bp == '\n')
450     (void) fflush(f);
452 if (ferror(f) || feof(f)) {
453     (void) printf("\n\007Write failed\n");
454     (void) fflush(stdout);
455     _exit(1);
456 }
457 }
458 (void) fclose(f);
459 (void) close(fd);
460 _exit(0);
461 }
```

```
464 static int
465 chkgrp(char *name)
466 {
467     int i;
468     char user[NMAX + 1];
469     char *p;
470     (void) strncpy(user, name, NMAX);
471     user[NMAX] = '\0';
472     for (i = 0; pgrp->gr_mem[i] && pgrp->gr_mem[i][0]; i++) {
473         if (strcmp(user, pgrp->gr_mem[i]) == 0)
474             for (p = name; *p && *p != ' '; p++)
475                 *p = 0;
476         if (strncmp(name, pgrp->gr_mem[i], 8) == 0)
477             return (1);
478     }
479     return (0);
480 }
```

unchanged portion omitted

new/usr/src/cmd/who/who.c

1

20484 Wed Apr 3 09:33:12 2013

new/usr/src/cmd/who/who.c

2989 Eliminate use of LOGNAME_MAX in ON

1166 useradd have warning with name more 8 chars

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

```
25 /*
26 * Copyright (c) 2013 Gary Mills
27 *
28 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
29 * Use is subject to license terms.
30 */
```

```
30 #pragma ident      "%Z%M% %I%      %E% SMI"
```

```
32 /*
33 *      This program analyzes information found in /var/adm/utmpx
34 *
35 *      Additionally information is gathered from /etc/inittab
36 *      if requested.
37 *
38 *
39 *      Syntax:
40 *
41 *      who am i          Displays info on yourself
42 *
43 *      who -a           Displays information about All
44 *                      entries in /var/adm/utmpx
45 *
46 *      who -b           Displays info on last boot
47 *
48 *      who -d           Displays info on DEAD PROCESSES
49 *
50 *      who -H           Displays HEADERS for output
51 *
52 *      who -l           Displays info on LOGIN entries
53 *
54 *      who -m           Same as who am i
55 *
56 *      who -p           Displays info on PROCESSES spawned by init
57 *
58 *      who -q           Displays short information on
```

new/usr/src/cmd/who/who.c

2

```
59 *
60 *
61 *      who -r          Displays info of current run-level
62 *
63 *      who -s          Displays requested info in SHORT form
64 *
65 *      who -t          Displays info on TIME changes
66 *
67 *      who -T          Displays writeability of each user
68 *                      (+ writeable, - non-writeable, ? hung)
69 *
70 *      who -u          Displays LONG info on users
71 *                      who have LOGGED ON
72 */
```

```
74 #define      DATE_FMT      "%b %e %H:%M"
```

```
76 /*
77 *      %b      Abbreviated month name
78 *      %e      Day of month
79 *      %H      hour (24-hour clock)
80 *      %M      minute
81 */
82 #include      <errno.h>
83 #include      <fcntl.h>
84 #include      <stdio.h>
85 #include      <string.h>
86 #include      <sys/types.h>
87 #include      <unistd.h>
88 #include      <stdlib.h>
89 #include      <sys/stat.h>
90 #include      <time.h>
91 #include      <utmpx.h>
92 #include      <locale.h>
93 #include      <pwd.h>
94 #include      <limits.h>
```

```
96 static void process(void);
97 static void ck_file(char *);
98 static void dump(void);
```

```
100 static struct      utmpx *utmpp;      /* pointer for getutxent() */
```

```
102 /*
103 *      Use the full lengths from utmpx for user and line.
104 *      utmpx defines wider fields for user and line. For compatibility of output,
105 *      we are limiting these to the old maximums in utmp. Define UTMPX_NAMELEN
106 *      to use the full lengths.
107 */
```

```
107 #ifndef      UTMPX_NAMELEN
108 /*      XXX - utmp - fix name length */
109 #define      NMAX      (_POSIX_LOGIN_NAME_MAX - 1)
110 #define      LMAX      12
111 #else /*      UTMPX_NAMELEN */
112 #define      NMAX      (sizeof (utmpp->ut_user))
113 #define      LMAX      (sizeof (utmpp->ut_line))
114 #endif
```

```
108 /*      Print minimum field widths. */
109 #define      LOGIN_WIDTH      8
110 #define      LINE_WIDTH      12
```

```
112 static char      comment[BUFSIZ]; /* holds inittab comment */
113 static char      errmsg[BUFSIZ]; /* used in snprintf for errors */
114 static int      fd; /* file descriptor for inittab */
115 static int      Hopt = 0; /* 1 = who -H */
```



```

116 static char *inittab; /* ptr to inittab contents */
117 static char *iinit; /* index into inittab */
118 static int justme = 0; /* 1 = who am i */
119 static struct tm *lptr; /* holds user login time */
120 static char *myname; /* pointer to invoker's name */
121 static char *mytty; /* holds device user is on */
122 static char nameval[sizeof (utmp->ut_user) + 1]; /* invoker's name */
123 static int number = 8; /* number of users per -q line */
124 static int optcnt = 0; /* keeps count of options */
125 static char outbuf[BUFSIZ]; /* buffer for output */
126 static char *program; /* holds name of this program */
127 #ifdef XPG4
128 static int aopt = 0; /* 1 = who -a */
129 static int dopt = 0; /* 1 = who -d */
130 #endif /* XPG4 */
131 static int qopt = 0; /* 1 = who -q */
132 static int sopt = 0; /* 1 = who -s */
133 static struct stat stbuf; /* area for stat buffer */
134 static struct stat *stbufp; /* ptr to structure */
135 static int terse = 1; /* 1 = print terse msgs */
136 static int Topt = 0; /* 1 = who -T */
137 static time_t timnow; /* holds current time */
138 static int totlusr = 0; /* cntr for users on system */
139 static int uopt = 0; /* 1 = who -u */
140 static char user[sizeof (utmp->ut_user) + 1]; /* holds user name */
141 static int validtype[UTMAXTYPE+1]; /* holds valid types */
142 static int wrap; /* flag to indicate wrap */
143 static char time_buf[128]; /* holds date and time string */
144 static char *end; /* used in strtol for end pointer */

146 int
147 main(int argc, char **argv)
148 {
149     int goerr = 0; /* non-zero indicates cmd error */
150     int i;
151     int optsw; /* switch for while of getopt() */

153     (void) setlocale(LC_ALL, "");

155 #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
156 #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it weren't */
157 #endif
158     (void) textdomain(TEXT_DOMAIN);

160     validtype[USER_PROCESS] = 1;
161     validtype[EMPTY] = 0;
162     stbufp = &stbuf;

164     /*
165      * Strip off path name of this command
166      */
167     for (i = strlen(argv[0]); i >= 0 && argv[0][i] != '/'; --i)
168         ;
169     for (i = strlen(argv[0]); i >= 0 && argv[0][i] != '/'; --i);
170     if (i >= 0)
171         argv[0] += i+1;
172     program = argv[0];

173     /*
174      * Buffer stdout for speed
175      */
176     setbuf(stdout, outbuf);

178     /*
179      * Retrieve options specified on command line
180      * XCU4 - add -m option

```

```

181     /*
182     while ((optsw = getopt(argc, argv, "abdHlmm:prstTu")) != EOF) {
183         optcnt++;
184         switch (optsw) {

186             case 'a':
187                 optcnt += 7;
188                 validtype[BOOT_TIME] = 1;
189                 validtype[DEAD_PROCESS] = 1;
190                 validtype[LOGIN_PROCESS] = 1;
191                 validtype[INIT_PROCESS] = 1;
192                 validtype[RUN_LVL] = 1;
193                 validtype[OLD_TIME] = 1;
194                 validtype[NEW_TIME] = 1;
195                 validtype[USER_PROCESS] = 1;
196 #ifdef XPG4
197                 aopt = 1;
198 #endif /* XPG4 */
199                 uopt = 1;
200                 Topt = 1;
201                 if (!sopt) terse = 0;
202                 break;

204             case 'b':
205                 validtype[BOOT_TIME] = 1;
206                 if (!uopt) validtype[USER_PROCESS] = 0;
207                 break;

209             case 'd':
210                 validtype[DEAD_PROCESS] = 1;
211                 if (!uopt) validtype[USER_PROCESS] = 0;
212 #ifdef XPG4
213                 dopt = 1;
214 #endif /* XPG4 */
215                 break;

217             case 'H':
218                 optcnt--; /* Don't count Header */
219                 Hopt = 1;
220                 break;

222             case 'l':
223                 validtype[LOGIN_PROCESS] = 1;
224                 if (!uopt) validtype[USER_PROCESS] = 0;
225                 terse = 0;
226                 break;

227             case 'm': /* New XCU4 option */
228                 justme = 1;
229                 break;

231             case 'n':
232                 errno = 0;
233                 number = strtol(optarg, &end, 10);
234                 if (errno != 0 || *end != '\0') {
235                     (void) fprintf(stderr, gettext(
236                         "%s: Invalid numeric argument\n"),
237                         program);
238                     exit(1);
239                 }
240                 if (number < 1) {
241                     (void) fprintf(stderr, gettext(
242                         "%s: Number of users per line must
243                         \"be at least 1\n\"), program);
244                     exit(1);
245                 }
246                 break;

```

```

248     case 'p':
249         validtype[INIT_PROCESS] = 1;
250         if (!uopt) validtype[USER_PROCESS] = 0;
251         break;

253     case 'q':
254         qopt = 1;
255         break;

257     case 'r':
258         validtype[RUN_LVL] = 1;
259         terse = 0;
260         if (!uopt) validtype[USER_PROCESS] = 0;
261         break;

263     case 's':
264         sopt = 1;
265         terse = 1;
266         break;

268     case 't':
269         validtype[OLD_TIME] = 1;
270         validtype[NEW_TIME] = 1;
271         if (!uopt) validtype[USER_PROCESS] = 0;
272         break;

274     case 'T':
275         Topt = 1;
276 #ifdef XPG4
277         terse = 1;      /* XPG4 requires -T */
278 #else /* XPG4 */
279         terse = 0;
280 #endif /* XPG4 */
281         break;

283     case 'u':
284         uopt = 1;
285         validtype[USER_PROCESS] = 1;
286         if (!sopt) terse = 0;
287         break;

289     case '?':
290         goerr++;
291         break;
292     default:
293         break;
294     }
295 }
296 #ifdef XPG4
297 /*
298  * XCU4 changes - check for illegal sopt, Topt & aopt combination
299  */
300 if (sopt == 1) {
301     terse = 1;
302     if (Topt == 1 || aopt == 1)
303         goerr++;
304 }
305 #endif /* XPG4 */

307 if (goerr > 0) {
308 #ifdef XPG4
309     /*
310      * XCU4 - slightly different usage with -s -a & -T
311      */
312     (void) fprintf(stderr, gettext("\nUsage:\t%s"), program);

```

```

313         (void) fprintf(stderr,
314             gettext("-s [-bdHlmpqrstu] [utmpx_like_file]\n"));

316     (void) fprintf(stderr, gettext(
317         "\t%s [-abdHlmpqrstu] [utmpx_like_file]\n"), program);
318 #else /* XPG4 */
319     (void) fprintf(stderr, gettext(
320         "\nUsage:\t%s [-abdHlmpqrstu] [utmpx_like_file]\n"),
321         program);
322 #endif /* XPG4 */
323     (void) fprintf(stderr,
324         gettext("\t%s -q [-n x] [utmpx_like_file]\n"), program);
325     (void) fprintf(stderr, gettext("\t%s [am i]\n"), program);
326     /*
327      * XCU4 changes - be explicit with "am i" options
328      */
329     (void) fprintf(stderr, gettext("\t%s [am I]\n"), program);
330     (void) fprintf(stderr, gettext(
331         "a\tall (bdlprrtu options)\n"));
332     (void) fprintf(stderr, gettext("b\tboot time\n"));
333     (void) fprintf(stderr, gettext("d\tdead processes\n"));
334     (void) fprintf(stderr, gettext("H\tprint header\n"));
335     (void) fprintf(stderr, gettext("l\tlogin processes\n"));
336     (void) fprintf(stderr, gettext(
337         "n #\tspecify number of users per line for -q\n"));
338     (void) fprintf(stderr,
339         gettext("p\tprocesses other than getty or users\n"));
340     (void) fprintf(stderr, gettext("q\tquick %s\n"), program);
341     (void) fprintf(stderr, gettext("r\ttrun level\n"));
342     (void) fprintf(stderr, gettext(
343         "s\tshort form of %s (no time since last output or pid)\n"),
344         program);
345     (void) fprintf(stderr, gettext("t\ttime changes\n"));
346     (void) fprintf(stderr, gettext(
347         "T\tstatus of tty (+ writable, - not writable, "
348         "? hung)\n"));
349     (void) fprintf(stderr, gettext("u\tuseful information\n"));
350     (void) fprintf(stderr,
351         gettext("m\tinformation only about current terminal\n"));
352     (void) fprintf(stderr, gettext(
353         "am i\tinformation about current terminal "
354         "(same as -m)\n"));
355     (void) fprintf(stderr, gettext(
356         "am I\tinformation about current terminal "
357         "(same as -m)\n"));
358     exit(1);
359 }

361 /*
362  * XCU4: If -q option ignore all other options
363  */
364 if (qopt == 1) {
365     Hopt = 0;
366     sopt = 0;
367     Topt = 0;
368     uopt = 0;
369     justme = 0;
370     validtype[ACCOUNTING] = 0;
371     validtype[BOOT_TIME] = 0;
372     validtype[DEAD_PROCESS] = 0;
373     validtype[LOGIN_PROCESS] = 0;
374     validtype[INIT_PROCESS] = 0;
375     validtype[RUN_LVL] = 0;
376     validtype[OLD_TIME] = 0;
377     validtype[NEW_TIME] = 0;
378     validtype[USER_PROCESS] = 1;

```

```

379     }
381     if (argc == optind + 1) {
382         optcnt++;
383         ck_file(argv[optind]);
384         (void) utmpxname(argv[optind]);
385     }
387     /*
388     *   Test for 'who am i' or 'who am I'
389     *   XCU4 - check if justme was already set by -m option
390     */
391     if (justme == 1 || (argc == 3 && strcmp(argv[1], "am") == 0 &&
392         ((argv[2][0] == 'i' || argv[2][0] == 'I') &&
393         argv[2][1] == '\0'))) {
394         justme = 1;
395         myname = nameval;
396         (void) cuserid(myname);
397         if ((mytty = ttyname(fileno(stdin))) == NULL &&
398             (mytty = ttyname(fileno(stdout))) == NULL &&
399             (mytty = ttyname(fileno(stderr))) == NULL) {
400             (void) fprintf(stderr, gettext(
401                 "Must be attached to terminal for 'am I' option\n"));
402             (void) fflush(stderr);
403             exit(1);
404         } else
405             mytty += 5; /* bump past "/dev/" */
406     }
408     if (!terse) {
409         if (Hopt)
410             (void) printf(gettext(
411                 "NAME      LINE      TIME      IDLE      PID      COMMENTS\n"));
413         timnow = time(0);
415         if ((fildes = open("/etc/inittab",
416             O_NONBLOCK|O_RDONLY)) == -1) {
417             (void) snprintf(errmsg, sizeof (errmsg),
418                 gettext("%s: Cannot open /etc/inittab"), program);
419             perror(errmsg);
420             exit(errno);
421         }
423         if (fstat(fildes, stbufp) == -1) {
424             (void) snprintf(errmsg, sizeof (errmsg),
425                 gettext("%s: Cannot stat /etc/inittab"), program);
426             perror(errmsg);
427             exit(errno);
428         }
430         if ((inittab = malloc(stbufp->st_size + 1)) == NULL) {
431             (void) snprintf(errmsg, sizeof (errmsg),
432                 gettext("%s: Cannot allocate %ld bytes"),
433                 program, stbufp->st_size);
434             perror(errmsg);
435             exit(errno);
436         }
438         if (read(fildes, inittab, stbufp->st_size)
439             != stbufp->st_size) {
440             (void) snprintf(errmsg, sizeof (errmsg),
441                 gettext("%s: Error reading /etc/inittab"),
442                 program);
443             perror(errmsg);
444             exit(errno);

```

```

445     }
447         inittab[stbufp->st_size] = '\0';
448         iinit = inittab;
449     } else {
450         if (Hopt) {
451             #ifdef XPG4
452                 if (dopt) {
453                     (void) printf(gettext(
454                         "NAME      LINE      TIME      COMMENTS\n"));
455                 } else {
456                     (void) printf(
457                         gettext("NAME      LINE      TIME\n"));
458                 }
459             #else /* XPG4 */
460                 (void) printf(
461                     gettext("NAME      LINE      TIME\n"));
462             #endif /* XPG4 */
463         }
464     }
465     process();
467     /*
468     *   'who -q' requires EOL upon exit,
469     *   followed by total line
470     */
471     if (qopt)
472         (void) printf(gettext("\n# users=%d\n"), totlusrs);
473     return (0);
474 }
476 static void
477 dump()
478 {
479     char    device[sizeof (utmp->ut_line) + 1];
480     time_t  hr;
481     time_t  idle;
482     time_t  min;
483     char    path[sizeof (utmp->ut_line) + 6];
484     int     pexit;
485     int     pterm;
486     int     rc;
487     char    w; /* writeability indicator */
489     /*
490     * Get and check user name
491     */
492     if (utmp->ut_user[0] == '\0')
493         (void) strcpy(user, " .");
494     else {
495         (void) strncpy(user, utmp->ut_user, sizeof (user));
496         user[sizeof (user) - 1] = '\0';
497     }
498     totlusrs++;
500     /*
501     * Do print in 'who -q' format
502     */
503     if (qopt) {
504         /*
505         * XCU4 - Use non user macro for correct user count
506         */
507         if (((totlusrs - 1) % number) == 0 && totlusrs > 1)
508             (void) printf("\n");
509         (void) printf("%-*. *s ", LOGIN_WIDTH, NMAX, user);
510         (void) printf("%-*s ", NMAX, user);

```

```

510         return;
511     }

514     pexit = (int)' ';
515     pterm = (int)' ';

517     /*
518     *       Get exit info if applicable
519     */
520     if (utmpp->ut_type == RUN_LVL || utmpp->ut_type == DEAD_PROCESS) {
521         pterm = utmpp->ut_exit.e_termination;
522         pexit = utmpp->ut_exit.e_exit;
523     }

525     /*
526     *       Message ut_xtime field
527     */
528     lptr = localtime(&utmpp->ut_xtime);
529     (void) strftime(time_buf, sizeof (time_buf),
530         dcgettext(NULL, DATE_FMT, LC_TIME), lptr);

532     /*
533     *       Get and message device
534     */
535     if (utmpp->ut_line[0] == '\0')
536         (void) strcpy(device, " .");
537     else {
538         (void) strncpy(device, utmpp->ut_line,
539             sizeof (utmpp->ut_line));
540         device[sizeof (utmpp->ut_line)] = '\0';
541     }

543     /*
544     *       Get writeability if requested
545     *       XCU4 - only print + or - for user processes
546     */
547     if (Topt && (utmpp->ut_type == USER_PROCESS)) {
548         w = '-';
549         (void) strcpy(path, "/dev/");
550         (void) strncpy(path + 5, utmpp->ut_line,
551             sizeof (utmpp->ut_line));
552         path[5 + sizeof (utmpp->ut_line)] = '\0';

554         if ((rc = stat(path, stbufp)) == -1) w = '?';
555         else if ((stbufp->st_mode & S_IWOTH) ||
556             (stbufp->st_mode & S_IWGRP)) /* Check group & other */
557             w = '+';

559     } else
560         w = ' ';

562     /*
563     *       Print the TERSE portion of the output
564     */
565     (void) printf("%-*s %c %-12s %s", LOGIN_WIDTH, NMAX, user,
566         w, device, time_buf);
568     (void) printf("%-*s %c %-12s %s", NMAX, user, w, device, time_buf);

568     if (!terse) {
569         /*
570         *       Stat device for idle time
571         *       (Don't complain if you can't)
572         */
573         rc = -1;
574         if (utmpp->ut_type == USER_PROCESS) {

```

```

575         (void) strcpy(path, "/dev/");
576         (void) strncpy(path + 5, utmpp->ut_line,
577             sizeof (utmpp->ut_line));
578         path[5 + sizeof (utmpp->ut_line)] = '\0';
579         rc = stat(path, stbufp);
580     }
581     if (rc != -1) {
582         idle = timnow - stbufp->st_mtime;
583         hr = idle/3600;
584         min = (unsigned)(idle/60)%60;
585         if (hr == 0 && min == 0)
586             (void) printf(gettext(" . "));
587         else {
588             if (hr < 24)
589                 (void) printf("%2d:%2.2d", (int)hr,
590                     (int)min);
591             else
592                 (void) printf(gettext(" old "));
593         }
594     }

596     /*
597     *       Add PID for verbose output
598     */
599     if (utmpp->ut_type != BOOT_TIME &&
600         utmpp->ut_type != RUN_LVL &&
601         utmpp->ut_type != ACCOUNTING)
602         (void) printf(" %5ld", utmpp->ut_pid);

604     /*
605     *       Handle /etc/inittab comment
606     */
607     if (utmpp->ut_type == DEAD_PROCESS) {
608         (void) printf(gettext(" id=%4.4s "),
609             utmpp->ut_id);
610         (void) printf(gettext("term=%-3d "), pterm);
611         (void) printf(gettext("exit=%d "), pexit);
612     } else if (utmpp->ut_type != INIT_PROCESS) {
613         /*
614         *       Search for each entry in inittab
615         *       string. Keep our place from
616         *       search to search to try and
617         *       minimize the work. Wrap once if needed
618         *       for each entry.
619         */
620         wrap = 0;
621         /*
622         *       Look for a line beginning with
623         *       utmpp->ut_id
624         */
625         while ((rc = strncmp(utmpp->ut_id, iinit,
626             strcspn(iinit, ":")) != 0) {
627             for (; *iinit != '\n'; iinit++)
628                 ;
629             for (; *iinit != '\n'; iinit++);
630             iinit++;

631             /*
632             *       Wrap once if necessary to
633             *       find entry in inittab
634             */
635             if (*iinit == '\0') {
636                 if (!wrap) {
637                     iinit = inittab;
638                     wrap = 1;
639                 }

```

```

640     }
641     }
642
643     if (*iinit != '\0') {
644         /*
645          *       We found our entry
646          */
647         for (iinit++; *iinit != '#' &&
648                *iinit != '\n'; iinit++)
649             ;
650         if (*iinit == '#') {
651             for (iinit++; *iinit == ' ' ||
652                    *iinit == '\t'; iinit++)
653                 ;
654             for (rc = 0; *iinit != '\n'; iinit++)
655                 comment[rc++] = *iinit;
656             comment[rc] = '\0';
657         } else
658             (void) strcpy(comment, " ");
659
660         (void) printf(" %s", comment);
661     } else
662         iinit = inittab; /* Reset pointer */
663 }
664 if (utmpp->ut_type == INIT_PROCESS)
665     (void) printf(gettext(" id=%4.4s"), utmpp->ut_id);
666 }
667 #ifdef XPG4
668 else
669     if (dopt && utmpp->ut_type == DEAD_PROCESS) {
670         (void) printf(gettext("\tterm=%-3d "), pterm);
671         (void) printf(gettext("exit=%d "), pexit);
672     }
673 #endif /* XPG4 */
674
675 /*
676 *       Handle RUN_LVL process - If no alt. file - Only one!
677 */
678 if (utmpp->ut_type == RUN_LVL) {
679     (void) printf(" %c %5ld %c", pterm, utmpp->ut_pid,
680                 pexit);
681     if (optcnt == 1 && !validtype[USER_PROCESS]) {
682         (void) printf("\n");
683         exit(0);
684     }
685 }
686
687 /*
688 *       Handle BOOT_TIME process - If no alt. file - Only one!
689 */
690 if (utmpp->ut_type == BOOT_TIME) {
691     if (optcnt == 1 && !validtype[USER_PROCESS]) {
692         (void) printf("\n");
693         exit(0);
694     }
695 }
696
697 /*
698 *       Get remote host from utmpx structure
699 */
700 if (utmpp && utmpp->ut_host[0])
701     (void) printf("\t(%.*s)", sizeof (utmpp->ut_host),
702                 utmpp->ut_host);

```

```

705     /*
706     *       Now, put on the trailing EOL
707     */
708     (void) printf("\n");
709 }
710
711 static void
712 process()
713 {
714     struct passwd *pwp;
715     int i = 0;
716     char *ttname;
717
718     /*
719     *       Loop over each entry in /var/adm/utmpx
720     */
721
722     setutxent();
723     while ((utmpp = getutxent()) != NULL) {
724 #ifdef DEBUG
725         (void) printf(
726             "ut_user '%s'\nut_id '%s'\nut_line '%s'\nut_type '%d'\n\n",
727             utmpp->ut_user, utmpp->ut_id, utmpp->ut_line, utmpp->ut_type);
728 #endif
729         if (utmpp->ut_type <= UTMAXTYPE) {
730             /*
731              *       Handle "am i"
732              */
733             if (justme) {
734                 if (strncmp(myname, utmpp->ut_user,
735                             sizeof (utmpp->ut_user)) == 0 &&
736                     strncmp(mytty, utmpp->ut_line,
737                             sizeof (utmpp->ut_line)) == 0 &&
738                     utmpp->ut_type == USER_PROCESS) {
739                     /*
740                      * we have have found ourselves
741                      * in the utmp file and the entry
742                      * is a user process, this is not
743                      * meaningful otherwise
744                      */
745
746                     dump();
747                     exit(0);
748                 }
749                 continue;
750             }
751
752             /*
753              *       Print the line if we want it
754              */
755             if (validtype[utmpp->ut_type]) {
756 #ifdef XPG4
757                 if (utmpp->ut_type == LOGIN_PROCESS) {
758                     if ((utmpp->ut_line[0] == '\0') ||
759                         (strcmp(utmpp->ut_user,
760                                "LOGIN") != 0))
759                         (strcmp(utmpp->ut_user, "LOGIN") != 0))
760                             continue;
761                 }
762 #endif /* XPG4 */
763                 dump();
764             } else {
765                 (void) fprintf(stderr,

```

```
769         gettext("%s: Error --- entry has ut_type "  
770         "of %d\n"), program, utmpp->ut_type);  
771         (void) fprintf(stderr,  
772         gettext(" when maximum is %d\n"), UTMATYPE);  
773     }  
774 }  
  
776 /*  
777  * If justme is set at this point than the utmp entry  
778  * was not found.  
779  */  
780 if (justme) {  
781     static struct utmpx utmpt;  
  
783     pwp = getpwuid(geteuid());  
784     if (pwp != NULL)  
785         while (i < (int)sizeof (utmpt.ut_user) &&  
786                *pwp->pw_name != 0)  
787             utmpt.ut_user[i++] = *pwp->pw_name++;  
  
789     ttname = ttyname(1);  
  
791     i = 0;  
792     if (ttname != NULL)  
793         while (i < (int)sizeof (utmpt.ut_line) &&  
794                *ttname != 0)  
795             utmpt.ut_line[i++] = *ttname++;  
  
797     utmpt.ut_id[0] = 0;  
798     utmpt.ut_pid = getpid();  
799     utmpt.ut_type = USER_PROCESS;  
800     (void) time(&utmpt.ut_xtime);  
801     utmpp = &utmpt;  
802     dump();  
803     exit(0);  
804 }  
805 }  
  
_____unchanged_portion_omitted_____
```

new/usr/src/cmd/whodo/whodo.c

1

```
*****
21012 Wed Apr 3 09:33:12 2013
new/usr/src/cmd/whodo/whodo.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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>
```

new/usr/src/cmd/whodo/whodo.c

2

```
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  * utmpx defines wider fields for user and line. For compatibility of output,
75  * we are limiting these to the old maximums in utmp. Define UTMPX_NAMELEN
76  * to use the full lengths.
77  */
78 #define NMAX      (sizeof (((struct utmpx *)0)->ut_user))
79 #define LMAX      (sizeof (((struct utmpx *)0)->ut_line))
80 #ifndef UTMPX_NAMELEN
81 /* XXX - utmp - fix name length */
82 #define NMAX      (_POSIX_LOGIN_NAME_MAX - 1)
83 #define LMAX      12
84 #else /* UTMPX_NAMELEN */
85 static struct utmpx dummy;
86 #define NMAX      (sizeof (dummy.ut_user))
87 #define LMAX      (sizeof (dummy.ut_line))
88 #endif /* UTMPX_NAMELEN */

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

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

93 #ifdef ERR
94 #undef ERR
95 #endif
96 #define ERR            (-1)

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

105 static int             ndevs;      /* number of configured devices */
106 static int             maxdev;     /* slots for configured devices */
107 #define DNINCR         100
108 static struct devl {
109     char               dname[DEVNAMELEN]; /* device name */
110     dev_t              ddev;          /* device number */
111 } *devl;

112 #ifndef unchanged_portion_omitted
113 /*
114  * define hash table for struct uproc
115  * Hash function uses process id
116  * and the size of the hash table(HSIZE)
117  * to determine process index into the table.
118  */
119 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 *);

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

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

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

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

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

```

```

196     prog = argv[0];

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

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

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

211                     default:
212                         (void) printf(gettext(
213                             "usage: %s [ -hl ] [ user ]\n"),
214                             prog);
215                         exit(1);
216                 }
217             }
218         } else {
219             if (!isalnum(argv[1][0]) || argc > 2) {
220                 (void) printf(gettext(
221                     "usage: %s [ -hl ] [ user ]\n"), prog);
222                 exit(1);
223             } else
224                 sel_user = argv[1];
225         }
226         argc--; argv++;
227     }

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

241     if ((ut = malloc(size)) == NULL) {
242         (void) fprintf(stderr, gettext("%s: malloc error of %s: %s\n"),
243                       prog, UTMPX_FILE, strerror(errno));
244         exit(1);
245     }

247     (void) utmpxname(UTMPX_FILE);

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

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

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

260     if (header) { /* print a header */

```



```

261     if (lflag) { /* w command format header */
262         prtat(&now);
263         for (ut = utmpbegin; ut < utmpend; ut++) {
264             if (ut->ut_type == USER_PROCESS) {
265                 nusers++;
266             } else if (ut->ut_type == BOOT_TIME) {
267                 uptime = now - ut->ut_xtime;
268                 uptime += 30;
269                 days = uptime / (60*60*24);
270                 uptime %= (60*60*24);
271                 hrs = uptime / (60*60);
272                 uptime %= (60*60);
273                 mins = uptime / 60;
274
275                 (void) printf(dcgettext(NULL,
276                     " up %d day(s), %d hr(s), "
277                     "%d min(s)", LC_TIME),
278                     days, hrs, mins);
279             }
280         }
281
282         ut = utmpbegin; /* rewind utmp data */
283         (void) printf(dcgettext(NULL,
284             " %d user(s)\n", LC_TIME), nusers);
285         (void) printf(dcgettext(NULL, "User      tty
286             "login@ idle JCPU PCPU what\n", LC_TIME));
287     } else { /* standard whodo header */
288         char date_buf[100];
289
290         /*
291          * print current time and date
292          */
293         (void) strftime(date_buf, sizeof (date_buf),
294             dcgettext(NULL, "%C", LC_TIME), localtime(&now));
295         (void) printf("%s\n", date_buf);
296
297         /*
298          * print system name
299          */
300         (void) uname(&uts);
301         (void) printf("%s\n", uts.nodename);
302     }
303 }
304
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 }
314
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);
337
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;
347
348     if (up->p_state != NONE && up->p_state != ZOMBIE) {
349         (void) strcpy(fname, "status");
350
351         /* now we need the proc_owner privilege */
352         (void) __priv_bracket(PRIV_ON);
353
354         procfd = open(pname, O_RDONLY);
355
356         /* drop proc_owner privilege after open */
357         (void) __priv_bracket(PRIV_OFF);
358
359         if (procfd < 0)
360             continue;
361
362         if (read(procfd, &stainfo, sizeof (stainfo))
363             != sizeof (stainfo)) {
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);
375
376         up->p_time = stainfo.pr_utime.tv_sec +
377             stainfo.pr_stime.tv_sec;
378         up->p_ctime = stainfo.pr_cutime.tv_sec +
379             stainfo.pr_cstime.tv_sec;
380
381         (void) strcpy(fname, "sigact");
382
383         /* now we need the proc_owner privilege */
384         (void) __priv_bracket(PRIV_ON);
385
386         procfd = open(pname, O_RDONLY);
387
388         /* drop proc_owner privilege after open */
389         (void) __priv_bracket(PRIV_OFF);
390
391         if (procfd < 0)
392             continue;

```

```

393     if (read(procf, actinfo, sizeof (actinfo))
394         != sizeof (actinfo)) {
395         int err = errno;
396         (void) close(procf);
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(procf);
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_pgrpplink = pgrp->p_pgrpplink;
439     pgrp->p_pgrpplink = 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 && strcmp(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);
475             (void) printf("%-*.s ", NMAX, NMAX, ut->ut_name);
476
477             /* print tty user is on */
478             (void) printf("%-*.s", LINE_WIDTH, (int)LMAX,
479                 ut->ut_line);
480             (void) printf("%-*.s", LMAX, LMAX, ut->ut_line);
481
482             /* print when the user logged in */
483             tim = ut->ut_xtime;
484             (void) prtcat(&tim);
485
486             /* print idle time */
487             idle = findidle(ut->ut_line);
488             if (idle >= 36 * 60)
489                 (void) printf(dcgettext(NULL, "%2ddays ",
490                     LC_TIME), (idle + 12 * 60) / (24 * 60));
491             else
492                 prttime(idle, " ");
493             showtotals(findhash((pid_t)ut->ut_pid));
494             } else { /* standard whodo format */
495             tim = ut->ut_xtime;
496             tm = localtime(&tim);
497             (void) printf("\n%-*.s %-*.s %2.1d:%2.2d\n",
498                 LINE_WIDTH, (int)LMAX, ut->ut_line,
499                 LOGIN_WIDTH, (int)NMAX, ut->ut_name, tm->tm_hour,
500                 tm->tm_min);
501             LMAX, LMAX, ut->ut_line,
502             NMAX, NMAX, ut->ut_name, tm->tm_hour, tm->tm_min);
503             showproc(findhash((pid_t)ut->ut_pid));
504         }
505     }
506     return (0);
507 }
508 /*
509  * Used for standard whodo format.
510  * This is the recursive routine descending the process
511  * tree starting from the given process pointer(up).
512  * It used depth-first search strategy and also marked
513  * each node as printed as it traversed down the tree.
514  */
515 static void
516 showproc(struct uproc *up)
517 {
518     struct uproc *zp;
519     if (up->p_state == VISITED) /* we already been here */
520         return;
521     /* print the data for this process */

```

```
521     if (up->p_state == ZOMBIE)
522         (void) printf("    %-*s %5d %4.1ld:%2.2ld %s\n",
523             LINE_WIDTH, (int)LMAX, "  ?", (int)up->p_upid, 0L, 0L,
524             "<defunct>");
525     else if (up->p_state != NONE) {
526         (void) printf("    %-*s %5d %4.1ld:%2.2ld %s\n",
527             LINE_WIDTH, (int)LMAX, getty(up->p_ttyd), (int)up->p_upid,
528             LMAX, LMAX, getty(up->p_ttyd), (int)up->p_upid,
529             up->p_time / 60L, up->p_time % 60L,
530             up->p_comm);
531     }
532     up->p_state = VISITED;
533
534     /* descend for its children */
535     if (up->p_child) {
536         showproc(up->p_child);
537         for (zp = up->p_child->p_sibling; zp; zp = zp->p_sibling) {
538             showproc(zp);
539         }
540     }
541
542     /* print the pgrp relation */
543     if (up->p_pgrplink)
544         showproc(up->p_pgrplink);
545 }
546
547 _____
548 unchanged_portion_omitted
```

new/usr/src/cmd/zlogin/zlogin.c

1

```
*****
57228 Wed Apr  3 09:33:12 2013
new/usr/src/cmd/zlogin/zlogin.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
25 */

27 /*
28 * zlogin provides three types of login which allow users in the global
29 * zone to access non-global zones.
30 *
31 * - "interactive login" is similar to rlogin(1); for example, the user could
32 * issue 'zlogin my-zone' or 'zlogin -e ^ -l me my-zone'. The user is
33 * granted a new pty (which is then shoved into the zone), and an I/O
34 * loop between parent and child processes takes care of the interactive
35 * session. In this mode, login(1) (and its -c option, which means
36 * "already authenticated") is employed to take care of the initialization
37 * of the user's session.
38 *
39 * - "non-interactive login" is similar to su(1M); the user could issue
40 * 'zlogin my-zone ls -l' and the command would be run as specified.
41 * In this mode, zlogin sets up pipes as the communication channel, and
42 * 'su' is used to do the login setup work.
43 *
44 * - "console login" is the equivalent to accessing the tip line for a
45 * zone. For example, the user can issue 'zlogin -C my-zone'.
46 * In this mode, zlogin contacts the zoneadm process via unix domain
47 * socket. If zoneadm is not running, it starts it. This allows the
48 * console to be available anytime the zone is installed, regardless of
49 * whether it is running.
50 */

52 #include <sys/socket.h>
53 #include <sys/termios.h>
54 #include <sys/utsname.h>
55 #include <sys/stat.h>
56 #include <sys/types.h>
57 #include <sys/contract/process.h>
58 #include <sys/ctfs.h>
59 #include <sys/brand.h>
60 #include <sys/wait.h>
```

new/usr/src/cmd/zlogin/zlogin.c

2

```
61 #include <alloca.h>
62 #include <assert.h>
63 #include <ctype.h>
64 #include <door.h>
65 #include <errno.h>
66 #include <nss_dbdefs.h>
67 #include <poll.h>
68 #include <priv.h>
69 #include <pwd.h>
70 #include <unistd.h>
71 #include <utmpx.h>
72 #include <sac.h>
73 #include <signal.h>
74 #include <stdarg.h>
75 #include <stdio.h>
76 #include <stdlib.h>
77 #include <string.h>
78 #include <strings.h>
79 #include <stropts.h>
80 #include <wait.h>
81 #include <zone.h>
82 #include <fcntl.h>
83 #include <libdevinfo.h>
84 #include <libintl.h>
85 #include <locale.h>
86 #include <libzonecfg.h>
87 #include <libcontract.h>
88 #include <libbrand.h>
89 #include <auth_list.h>
90 #include <auth_attr.h>
91 #include <secdb.h>

93 static int masterfd;
94 static struct termios save_termios;
95 static struct termios effective_termios;
96 static int save_fd;
97 static struct winsize winsize;
98 static volatile int dead;
99 static volatile pid_t child_pid = -1;
100 static int interactive = 0;
101 static priv_set_t *dropprivs;

103 static int nocmdchar = 0;
104 static int failsafe = 0;
105 static char cmdchar = '~';

107 static int pollerr = 0;

109 static const char *pname;
110 static char *username;

112 /*
113 * When forced_login is true, the user is not prompted
114 * for an authentication password in the target zone.
115 */
116 static boolean_t forced_login = B_FALSE;

118 #if !defined(TEXT_DOMAIN) /* should be defined by cc -D */
119 #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it wasn't */
120 #endif

122 #define SUPATH "/usr/bin/su"
123 #define FAILSAFEHELL "/sbin/sh"
124 #define DEFAULTSHELL "/sbin/sh"
125 #define DEF_PATH "/usr/sbin:/usr/bin"
```

```

127 #define CLUSTER_BRAND_NAME      "cluster"

129 /*
130 * The ZLOGIN_BUFSIZ is larger than PIPE_BUF so we can be sure we're clearing
131 * out the pipe when the child is exiting. The ZLOGIN_RDBUFSIZ must be less
132 * than ZLOGIN_BUFSIZ (because we share the buffer in doio). This value is
133 * also chosen in conjunction with the HI_WATER setting to make sure we
134 * don't fill up the pipe. We can write FIFOHIWAT (16k) into the pipe before
135 * blocking. By having ZLOGIN_RDBUFSIZ set to 1k and HI_WATER set to 8k, we
136 * know we can always write a ZLOGIN_RDBUFSIZ chunk into the pipe when there
137 * is less than HI_WATER data already in the pipe.
138 */
139 #define ZLOGIN_BUFSIZ      8192
140 #define ZLOGIN_RDBUFSIZ  1024
141 #define HI_WATER          8192

143 /*
144 * See canonify() below. CANONIFY_LEN is the maximum length that a
145 * "canonical" sequence will expand to (backslash, three octal digits, NUL).
146 */
147 #define CANONIFY_LEN 5

149 static void
150 usage(void)
151 {
152     (void) fprintf(stderr, gettext("usage: %s [ -CES ] [ -e cmdchar ] "
153     "[-l user] zonename [command [args ...] ]\n"), pname);
154     exit(2);
155 }

unchanged_portion_omitted

1229 /*
1230 * Finish the preparation of the envp array for exec'd non-interactive
1231 * zlogins. This is called in the child process *after* we zone_enter(), since
1232 * it derives things we can only know within the zone, such as $HOME, $SHELL,
1233 * etc. We need only do this in the non-interactive, mode, since otherwise
1234 * login(1) will do it. We don't do this in failsafe mode, since it presents
1235 * additional ways in which the command could fail, and we'd prefer to avoid
1236 * that.
1237 */
1238 static char **
1239 prep_env_noninteractive(const char *user_cmd, char **env)
1240 {
1241     size_t size;
1242     char **new_env;
1243     int e, i;
1244     char *estr;
1245     char varmail[LOGNAME_MAX_ILLUMOS + 11]; /* strlen(/var/mail/) = */
1246     /* 10, NUL */
1247     char varmail[LOGNAME_MAX + 11]; /* strlen(/var/mail/) = 10, NUL */
1248     char pwbuf[NSS_BUFLEN_PASSWD + 1];
1249     struct passwd pwent;
1250     struct passwd *pw = NULL;

1251     assert(env != NULL);
1252     assert(failsafe == 0);

1254     /*
1255     * Exec the "user_cmd" brand hook to get a pwent for the
1256     * login user. If this fails, HOME will be set to "/", SHELL
1257     * will be set to $DEFAULTSHELL, and we will continue to exec
1258     * $SHELL <login> -c <cmd>.
1259     */
1260     pw = zone_get_user_pw(user_cmd, &pwent, pwbuf, sizeof(pwbuf));

1262     /*

```

```

1263     * Get existing envp size.
1264     */
1265     for (size = 0; env[size] != NULL; size++)
1266         ;

1268     e = size;

1270     /*
1271     * Finish filling out the environment; we duplicate the environment
1272     * setup described in login(1), for lack of a better precedent.
1273     */
1274     if (pw != NULL)
1275         size += 3; /* LOGNAME, HOME, MAIL */
1276     else
1277         size += 1; /* HOME */

1279     size++; /* always fill in SHELL */
1280     size++; /* terminating NULL */

1282     if ((new_env = malloc(sizeof(char *) * size)) == NULL)
1283         goto malloc_fail;

1285     /*
1286     * Copy existing elements of env into new_env.
1287     */
1288     for (i = 0; env[i] != NULL; i++) {
1289         if ((new_env[i] = strdup(env[i])) == NULL)
1290             goto malloc_fail;
1291     }
1292     assert(e == i);

1294     if (pw != NULL) {
1295         if ((estr = add_env("LOGNAME", pw->pw_name)) == NULL)
1296             goto malloc_fail;
1297         new_env[e++] = estr;

1299         if ((estr = add_env("HOME", pw->pw_dir)) == NULL)
1300             goto malloc_fail;
1301         new_env[e++] = estr;

1303         if (chdir(pw->pw_dir) != 0)
1304             zerror(gettext("Could not chdir to home directory "
1305             "%s: %s"), pw->pw_dir, strerror(errno));

1307         (void) snprintf(varmail, sizeof(varmail), "/var/mail/%s",
1308         pw->pw_name);
1309         if ((estr = add_env("MAIL", varmail)) == NULL)
1310             goto malloc_fail;
1311         new_env[e++] = estr;
1312     } else {
1313         if ((estr = add_env("HOME", "/")) == NULL)
1314             goto malloc_fail;
1315         new_env[e++] = estr;
1316     }

1318     if (pw != NULL && strlen(pw->pw_shell) > 0) {
1319         if ((estr = add_env("SHELL", pw->pw_shell)) == NULL)
1320             goto malloc_fail;
1321         new_env[e++] = estr;
1322     } else {
1323         if ((estr = add_env("SHELL", DEFAULTSHELL)) == NULL)
1324             goto malloc_fail;
1325         new_env[e++] = estr;
1326     }

1328     new_env[e++] = NULL; /* add terminating NULL */

```

```
1330     assert(e == size);
1331     return (new_env);

1333 malloc_fail:
1334     zerror(gettext("failed to allocate memory for process environment"));
1335     return (NULL);
1336 }
_____unchanged_portion_omitted_
```

new/usr/src/head/limits.h

1

```
*****
10568 Wed Apr 3 09:33:12 2013
new/usr/src/head/limits.h
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 */

22 /*
23  * Copyright (c) 2013 Gary Mills
24  *
25  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
26  * Use is subject to license terms.
27  */

29 /*      Copyright (c) 1988 AT&T */
30 /*      All Rights Reserved */

33 #ifndef _LIMITS_H
34 #define _LIMITS_H

34 #pragma ident "%Z%M% %I% %E% SMI" /* SVr4.0 1.34 */

36 #include <sys/feature_tests.h>
37 #include <sys/isa_defs.h>
38 #include <iso/limits_iso.h>

40 /*
41  * Include fixed width type limits as proposed by the ISO/JTC1/SC22/WG14 C
42  * committee's working draft for the revision of the current ISO C standard,
43  * ISO/IEC 9899:1990 Programming language - C. These are not currently
44  * required by any standard but constitute a useful, general purpose set
45  * of type definitions and limits which is namespace clean with respect to
46  * all standards.
47  */
48 #if defined(__EXTENSIONS__) || !defined(_STRICT_STDC) || \
49     defined(__XOPEN_OR_POSIX)
50 #include <sys/int_limits.h>
51 #endif

53 #ifdef __cplusplus
54 extern "C" {
55 #endif

57 #if defined(__EXTENSIONS__) || !defined(_STRICT_STDC) || \
58     defined(__XOPEN_OR_POSIX)
```

new/usr/src/head/limits.h

2

```
60 #define SSIZE_MAX        LONG_MAX        /* max value of an "ssize_t" */

62 /*
63  * ARG_MAX is calculated as follows:
64  * NCARGS - space for other stuff on initial stack
65  * like aux vectors, saved registers, etc..
66  */
67 #define _ARG_MAX32        1048320 /* max length of args to exec 32-bit program */
68 #define _ARG_MAX64        2096640 /* max length of args to exec 64-bit program */
69 #ifdef _LP64
70 #define ARG_MAX            _ARG_MAX64    /* max length of arguments to exec */
71 #else /* _LP64 */
72 #define ARG_MAX            _ARG_MAX32    /* max length of arguments to exec */
73 #endif /* _LP64 */

75 #ifndef MAX_CANON
76 #define MAX_CANON        256        /* max bytes in line for canonical processing */
77 #endif

79 #ifndef MAX_INPUT
80 #define MAX_INPUT        512        /* max size of a char input buffer */
81 #endif

83 #define NGROUPS_MAX        16        /* max number of groups for a user */

85 #ifndef PATH_MAX
86 #define PATH_MAX        1024        /* max # of characters in a path name */
87 #endif

89 #define SYMLINK_MAX        1024        /* max # of characters a symlink can contain */

91 #define PIPE_BUF        5120        /* max # bytes atomic in write to a pipe */

93 #ifndef TMP_MAX
94 #define TMP_MAX        17576        /* 26 * 26 * 26 */
95 #endif

97 /*
98  * POSIX conformant definitions - An implementation may define
99  * other symbols which reflect the actual implementation. Alternate
100 * definitions may not be as restrictive as the POSIX definitions.
101 */
102 #define _POSIX_AIO_LISTIO_MAX        2
103 #define _POSIX_AIO_MAX        1
104 #define _POSIX_ARG_MAX        4096
105 #ifdef _XPG6
106 #define _POSIX_CHILD_MAX        25
107 #else
108 #define _POSIX_CHILD_MAX        6        /* POSIX.1-1990 default */
109 #endif
110 #define _POSIX_CLOCKRES_MIN        20000000
111 #define _POSIX_DELAYTIMER_MAX        32
112 #define _POSIX_LINK_MAX        8
113 #define _POSIX_MAX_CANON        255
114 #define _POSIX_MAX_INPUT        255
115 #define _POSIX_MQ_OPEN_MAX        8
116 #define _POSIX_MQ_PRIO_MAX        32
117 #define _POSIX_NAME_MAX        14
118 #ifdef _XPG6
119 #define _POSIX_NGROUPS_MAX        8
120 #define _POSIX_OPEN_MAX        20
121 #define _POSIX_PATH_MAX        256
122 #else
123 #define _POSIX_NGROUPS_MAX        0        /* POSIX.1-1990 defaults */
124 #define _POSIX_OPEN_MAX        16
```

```

125 #define _POSIX_PATH_MAX          255
126 #endif
127 #define _POSIX_PIPE_BUF          512
128 #define _POSIX_RTSIG_MAX         8
129 #define _POSIX_SEM_NSEMS_MAX     256
130 #define _POSIX_SEM_VALUE_MAX     32767
131 #define _POSIX_SIGQUEUE_MAX     32
132 #define _POSIX_SSIZE_MAX        32767
133 #define _POSIX_STREAM_MAX       8
134 #define _POSIX_TIMER_MAX        32
135 #ifdef _XPG6
136 #define _POSIX_TZNAME_MAX        6
137 #else
138 #define _POSIX_TZNAME_MAX        3 /* POSIX.1-1990 default */
139 #endif
140 /* POSIX.1c conformant */
141 #define _POSIX_LOGIN_NAME_MAX    9
142 #define _POSIX_THREAD_DESTRUCTOR_ITERATIONS 4
143 #define _POSIX_THREAD_KEYS_MAX   128
144 #define _POSIX_THREAD_THREADS_MAX 64
145 #define _POSIX_TTY_NAME_MAX     9
146 /* UNIX 03 conformant */
147 #define _POSIX_HOST_NAME_MAX     255
148 #define _POSIX_RE_DUP_MAX        255
149 #define _POSIX_SYMLINK_MAX       255
150 #define _POSIX_SYMLINK_MAX       8

152 /*
153 * POSIX.2 and XPG4-XSH4 conformant definitions
154 */

156 #define _POSIX2_BC_BASE_MAX      99
157 #define _POSIX2_BC_DIM_MAX       2048
158 #define _POSIX2_BC_SCALE_MAX     99
159 #define _POSIX2_BC_STRING_MAX    1000
160 #define _POSIX2_COLL_WEIGHTS_MAX 2
161 #define _POSIX2_EXPR_NEST_MAX    32
162 #define _POSIX2_LINE_MAX         2048
163 #define _POSIX2_RE_DUP_MAX       255
164 /* UNIX 03 conformant */
165 #define _POSIX2_CHARCLASS_NAME_MAX 14

167 #define BC_BASE_MAX              _POSIX2_BC_BASE_MAX
168 #define BC_DIM_MAX               _POSIX2_BC_DIM_MAX
169 #define BC_SCALE_MAX            _POSIX2_BC_SCALE_MAX
170 #define BC_STRING_MAX           _POSIX2_BC_STRING_MAX
171 #define COLL_WEIGHTS_MAX        10
172 #define EXPR_NEST_MAX           _POSIX2_EXPR_NEST_MAX
173 #define LINE_MAX                 _POSIX2_LINE_MAX
174 #if !defined(_XPG6)
175 #define RE_DUP_MAX              _POSIX2_RE_DUP_MAX
176 #else
177 #define RE_DUP_MAX              _POSIX_RE_DUP_MAX
178 #endif /* !defined(_XPG6) */

180 #endif /* defined(__EXTENSIONS__) || !defined(_STRICT_STDC) ... */

182 #if defined(__EXTENSIONS__) || \
183 (!defined(_STRICT_STDC) && !defined(_POSIX_C_SOURCE)) || \
184 defined(_XOPEN_SOURCE)

186 /*
187 * For dual definitions for PASS_MAX and sysconf.c
188 */
189 #define _PASS_MAX_XPG           8 /* old standards PASS_MAX */
190 #define _PASS_MAX              256 /* modern Solaris PASS_MAX */

```

```

192 #if defined(_XPG3) && !defined(_XPG6)
193 #define PASS_MAX                _PASS_MAX_XPG /* max # of characters in a password */
194 #else /* XPG6 or just Solaris */
195 #define PASS_MAX                _PASS_MAX /* max # of characters in a password */
196 #endif /* defined(_XPG3) && !defined(_XPG6) */

198 #define CHARCLASS_NAME_MAX      _POSIX2_CHARCLASS_NAME_MAX

200 #define NL_ARGMAX               9 /* max value of "digit" in calls to the */
201 /* NLS printf() and scanf() */
202 #define NL_LANGMAX              14 /* max # of bytes in a LANG name */
203 #define NL_MSGMAX               32767 /* max message number */
204 #define NL_NMAX                  1 /* max # bytes in N-to-1 mapping characters */
205 #define NL_SETMAX                255 /* max set number */
206 #define NL_TEXTMAX              2048 /* max set number */
207 #define NZERO                    20 /* default process priority */

209 #define WORD_BIT                 32 /* # of bits in a "word" or "int" */
210 #if defined(_LP64)
211 #define LONG_BIT                 64 /* # of bits in a "long" */
212 #else /* _ILP32 */
213 #define LONG_BIT                 32 /* # of bits in a "long" */
214 #endif

216 /* Marked as LEGACY in SUSv2 and removed in UNIX 03 */
217 #ifndef _XPG6
218 #define DBL_DIG                  15 /* digits of precision of a "double" */
219 #define DBL_MAX                  1.7976931348623157081452E+308 /* max decimal value */
220 /* of a double */
221 #define FLT_DIG                   6 /* digits of precision of a "float" */
222 #define FLT_MAX                   3.4028234663852885981170E+38F /* max decimal value */
223 /* of a "float" */
224 #endif

226 /* Marked as LEGACY in SUSv1 and removed in SUSv2 */
227 #ifndef _XPG5
228 #define DBL_MIN                   2.2250738585072013830903E-308 /* min decimal value */
229 /* of a double */
230 #define FLT_MIN                   1.1754943508222875079688E-38F /* min decimal value */
231 /* of a float */
232 #endif

234 #endif /* defined(__EXTENSIONS__) || (!defined(_STRICT_STDC) ... */

236 #define _XOPEN_IOV_MAX           16 /* max # iovec/process with readv()/writev() */
237 #define _XOPEN_NAME_MAX         255 /* max # bytes in filename excluding null */
238 #define _XOPEN_PATH_MAX         1024 /* max # bytes in a pathname */

240 #define IOV_MAX                  _XOPEN_IOV_MAX

242 #if defined(__EXTENSIONS__) || \
243 (!defined(_STRICT_STDC) && !defined(_XOPEN_OR_POSIX))

245 #define FCHR_MAX                 1048576 /* max size of a file in bytes */
246 #define PID_MAX                  999999 /* max value for a process ID */

248 /*
249 * POSIX 1003.1a, section 2.9.5, table 2-5 contains [NAME_MAX] and the
250 * related text states:
251 *
252 * A definition of one of the values from Table 2-5 shall be omitted from the
253 * <limits.h> on specific implementations where the corresponding value is
254 * equal to or greater than the stated minimum, but where the value can vary
255 * depending on the file to which it is applied. The actual value supported for
256 * a specific pathname shall be provided by the pathconf() (5.7.1) function.

```



```

257 *
258 * This is clear that any machine supporting multiple file system types
259 * and/or a network can not include this define, regardless of protection
260 * by the _POSIX_SOURCE and _POSIX_C_SOURCE flags.
261 *
262 * #define      NAME_MAX      14
263 */

265 #define CHILD_MAX      25      /* max # of processes per user id */
266 #ifndef OPEN_MAX
267 #define OPEN_MAX      256      /* max # of files a process can have open */
268 #endif

270 #define PIPE_MAX      5120     /* max # bytes written to a pipe in a write */

272 #define STD_BLK      1024     /* # bytes in a physical I/O block */
273 #define UID_MAX      2147483647 /* max value for a user or group ID */
274 #define USI_MAX      4294967295 /* max decimal value of an "unsigned" */
275 #define SYSPID_MAX      1      /* max pid of system processes */

277 #ifndef SYS_NMLN
278 #define SYS_NMLN      257     /* also defined in sys/utsname.h */
279 #endif

281 #ifndef CLK_TCK
283 #if !defined(_CLOCK_T) || __cplusplus >= 199711L
284 #define _CLOCK_T
285 typedef long clock_t;
286 #endif /* !_CLOCK_T */

288 extern long _sysconf(int); /* System Private interface to sysconf() */
289 #define CLK_TCK ((clock_t)_sysconf(3)) /* 3 is _SC_CLK_TCK */

291 #endif /* CLK_TCK */

293 #define LOGNAME_MAX      8      /* max # of characters in a login name */
294 #define LOGNAME_MAX_ILLUMOS      32 /* max # of characters in an */
295 /* illumos login name */
296 #define LOGIN_NAME_MAX (LOGNAME_MAX_ILLUMOS + 1) /* max buffer size */
297 #define TTYNAME_MAX      128     /* max # of characters in a tty name */

299 #endif /* if defined(__EXTENSIONS__) || (!defined(_STRICT_STDC) ... */

301 #if defined(__EXTENSIONS__) || (_POSIX_C_SOURCE >= 199506L)
302 #include <sys/unistd.h>

304 #if !defined(_SIZE_T) || __cplusplus >= 199711L
305 #define _SIZE_T
306 #if defined(_LP64) || defined(_I32LPx)
307 typedef unsigned long size_t; /* size of something in bytes */
308 #else
309 typedef unsigned int size_t; /* (historical version) */
310 #endif
311 #endif /* _SIZE_T */

313 extern long _sysconf(int); /* System Private interface to sysconf() */

315 #define PTHREAD_STACK_MIN ((size_t)_sysconf(_SC_THREAD_STACK_MIN))
316 /* Added for UNIX98 conformance */
317 #define PTHREAD_DESTRUCTOR_ITERATIONS _POSIX_THREAD_DESTRUCTOR_ITERATIONS
318 #define PTHREAD_KEYS_MAX _POSIX_THREAD_KEYS_MAX
319 #define PTHREAD_THREADS_MAX _POSIX_THREAD_THREADS_MAX
320 #endif /* defined(__EXTENSIONS__) || (_POSIX_C_SOURCE >= 199506L) */

322 #ifdef __cplusplus

```

```

323 }
    unchanged_portion_omitted

```

```

*****
29037 Wed Apr 3 09:33:12 2013
new/usr/src/head/nss_dbdefs.h
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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  *
27  * Database-specific definitions for the getXXXbyYYY routines
28  * (e.g getpwuid_r(), ether_ntohost()) that use the name-service switch.
29  * Database-independent definitions are in <nss_common.h>
30  *
31  * Ideally, this is the only switch header file one would add things
32  * to in order to support a new database.
33  *
34  * NOTE: The interfaces documented in this file may change in a minor
35  * release. It is intended that in the future a stronger commitment
36  * will be made to these interface definitions which will guarantee
37  * them across minor releases.
38  */

40 #ifndef _NSS_DBDEFS_H
41 #define _NSS_DBDEFS_H

43 #include <sys/types.h>
44 #include <unistd.h>
45 #include <errno.h>
46 #include <netdb.h> /* MAXALIASES, MAXADDRS */
47 #include <limits.h> /* LOGNAME_MAX */
48 #include <nss_common.h>

50 #ifdef __cplusplus
51 extern "C" {
52 #endif

54 #ifndef NSS_INCLUDE_UNSAFE
55 #define NSS_INCLUDE_UNSAFE 1 /* Build old, MT-unsafe interfaces, */
56 #endif /* NSS_INCLUDE_UNSAFE */ /* e.g. getpwnam (c.f. getpwnam_r) */

58 /*
59  * Names of the well-known databases.
60  */

```

```

62 #define NSS_DBNAM_ALIASES "aliases" /* E-mail aliases, that is */
63 #define NSS_DBNAM_AUTOMOUNT "automount"
64 #define NSS_DBNAM_BOOTPARAMS "bootparams"
65 #define NSS_DBNAM_ETHERS "ethers"
66 #define NSS_DBNAM_GROUP "group"
67 #define NSS_DBNAM_HOSTS "hosts"
68 #define NSS_DBNAM_IPNODES "ipnodes"
69 #define NSS_DBNAM_NETGROUP "netgroup"
70 #define NSS_DBNAM_NETMASKS "netmasks"
71 #define NSS_DBNAM_NETWORKS "networks"
72 #define NSS_DBNAM_PASSWD "passwd"
73 #define NSS_DBNAM_PRINTERS "printers"
74 #define NSS_DBNAM_PROJECT "project"
75 #define NSS_DBNAM_PROTOCOLS "protocols"
76 #define NSS_DBNAM_PUBLICKEY "publickey"
77 #define NSS_DBNAM_RPC "rpc"
78 #define NSS_DBNAM_SERVICES "services"
79 #define NSS_DBNAM_AUDITUSER "audit_user"
80 #define NSS_DBNAM_AUTHATTR "auth_attr"
81 #define NSS_DBNAM_EXECATTR "exec_attr"
82 #define NSS_DBNAM_PROFATTR "prof_attr"
83 #define NSS_DBNAM_USERATTR "user_attr"

85 #define NSS_DBNAM_TSOL_TP "tnrhtp"
86 #define NSS_DBNAM_TSOL_RH "tnrhdb"
87 #define NSS_DBNAM_TSOL_ZC "tnzonecfg"

89 /* getsppnam() et al use the "passwd" config entry but the "shadow" backend */
90 #define NSS_DBNAM_SHADOW "shadow"

92 /* The "compat" backend gets config entries for these pseudo-databases */
93 #define NSS_DBNAM_PASSWD_COMPAT "passwd_compat"
94 #define NSS_DBNAM_GROUP_COMPAT "group_compat"

96 /*
97  * Default switch configuration, compiled into the front-ends.
98  *
99  * Absent good reasons to the contrary, this should be compatible with the
100 * default /etc/nsswitch.conf file.
101  */
102 #define NSS_FILES_ONLY "files"
103 #define NSS_FILES_NS "files nis"
104 #define NSS_NS_FALLBACK "nis [NOTFOUND=return] files"
105 #define NSS_NS_ONLY "nis"
106 #define NSS_TSOL_FALLBACK "files ldap"

108 #define NSS_DEFCONF_ALIASES NSS_FILES_NS
109 #define NSS_DEFCONF_AUTOMOUNT NSS_FILES_NS
110 #define NSS_DEFCONF_BOOTPARAMS NSS_NS_FALLBACK
111 #define NSS_DEFCONF_ETHERS NSS_NS_FALLBACK
112 #define NSS_DEFCONF_GROUP NSS_FILES_NS
113 #define NSS_DEFCONF_HOSTS NSS_NS_FALLBACK
114 #define NSS_DEFCONF_IPNODES NSS_NS_FALLBACK
115 #define NSS_DEFCONF_NETGROUP NSS_NS_ONLY
116 #define NSS_DEFCONF_NETMASKS NSS_NS_FALLBACK
117 #define NSS_DEFCONF_NETWORKS NSS_NS_FALLBACK
118 #define NSS_DEFCONF_PASSWD NSS_FILES_NS
119 #define NSS_DEFCONF_PRINTERS "user files nis"
120 #define NSS_DEFCONF_PROJECT NSS_FILES_NS
121 #define NSS_DEFCONF_PROTOCOLS NSS_NS_FALLBACK
122 #define NSS_DEFCONF_PUBLICKEY NSS_FILES_NS
123 #define NSS_DEFCONF_RPC NSS_NS_FALLBACK
124 #define NSS_DEFCONF_SERVICES NSS_FILES_NS /* speeds up byname() */

126 #define NSS_DEFCONF_GROUP_COMPAT NSS_NS_ONLY

```

new/usr/src/head/nss_dbdefs.h

```

127 #define NSS_DEFCONF_PASSWD_COMPAT      NSS_NS_ONLY
129 #define NSS_DEFCONF_ATTRDB            NSS_FILES_NS

131 #define NSS_DEFCONF_AUDITUSER          NSS_DEFCONF_PASSWD
132 #define NSS_DEFCONF_USERATTR          NSS_DEFCONF_PASSWD
133 #define NSS_DEFCONF_AUTHATTR          NSS_DEFCONF_ATTRDB
134 #define NSS_DEFCONF_PROFATTR          NSS_DEFCONF_ATTRDB
135 #define NSS_DEFCONF_EXECATTR          NSS_DEFCONF_PROFATTR

137 #define NSS_DEFCONF_TSOL_TP            NSS_TSOL_FALLBACK
138 #define NSS_DEFCONF_TSOL_RH            NSS_TSOL_FALLBACK
139 #define NSS_DEFCONF_TSOL_ZC            NSS_TSOL_FALLBACK

141 /*
142 * Line-lengths that the "files" and "compat" backends will try to support.
143 * It may be reasonable (even advisable) to use smaller values than these.
144 */

146 #define NSS_BUFSIZ                      1024

148 #define NSS_LINELEN_GROUP                ((NSS_BUFSIZ) * 8)
149 #define NSS_LINELEN_HOSTS                ((NSS_BUFSIZ) * 8)
150 #define NSS_LINELEN_IPNODES              ((NSS_BUFSIZ) * 8)
151 #define NSS_LINELEN_NETMASKS            NSS_BUFSIZ
152 #define NSS_LINELEN_NETWORKS            NSS_BUFSIZ
153 #define NSS_LINELEN_PASSWD               NSS_BUFSIZ
154 #define NSS_LINELEN_PRINTERS             NSS_BUFSIZ
155 #define NSS_LINELEN_PROJECT              ((NSS_BUFSIZ) * 4)
156 #define NSS_LINELEN_PROTOCOLS           NSS_BUFSIZ
157 #define NSS_LINELEN_PUBLICKEY           NSS_BUFSIZ
158 #define NSS_LINELEN_RPC                  NSS_BUFSIZ
159 #define NSS_LINELEN_SERVICES             NSS_BUFSIZ
160 #define NSS_LINELEN_SHADOW                NSS_BUFSIZ
161 #define NSS_LINELEN_ETHERS               NSS_BUFSIZ
162 #define NSS_LINELEN_BOOTPARAMS           NSS_BUFSIZ

164 #define NSS_LINELEN_ATTRDB              NSS_BUFSIZ

166 #define NSS_LINELEN_AUDITUSER            NSS_LINELEN_ATTRDB
167 #define NSS_LINELEN_AUTHATTR            NSS_LINELEN_ATTRDB
168 #define NSS_LINELEN_EXECATTR            NSS_LINELEN_ATTRDB
169 #define NSS_LINELEN_PROFATTR            NSS_LINELEN_ATTRDB
170 #define NSS_LINELEN_USERATTR            NSS_LINELEN_ATTRDB

172 #define NSS_MMAPLEN_EXECATTR            NSS_LINELEN_EXECATTR * 8

174 #define NSS_LINELEN_TSOL                 NSS_BUFSIZ

176 #define NSS_LINELEN_TSOL_TP              NSS_LINELEN_TSOL
177 #define NSS_LINELEN_TSOL_RH              NSS_LINELEN_TSOL
178 #define NSS_LINELEN_TSOL_ZC              NSS_LINELEN_TSOL

180 /*
181 * Reasonable defaults for 'buflen' values passed to _r functions.  The BSD
182 * and SunOS 4.x implementations of the getXXXbyYYY() functions used hard-
183 * coded array sizes; the values here are meant to handle anything that
184 * those implementations handled.
185 * == These might more reasonably go in <pwd.h>, <netdb.h> et al
186 */

188 #define NSS_BUFLLEN_GROUP                 NSS_LINELEN_GROUP
189 #define NSS_BUFLLEN_HOSTS                 \
190     (NSS_LINELEN_HOSTS + (MAXALIASES + MAXADDRS + 2) * sizeof (char *))
191 #define NSS_BUFLLEN_IPNODES              \
192     (NSS_LINELEN_IPNODES + (MAXALIASES + MAXADDRS + 2) * sizeof (char *))

```

3

new/usr/src/head/nss_dbdefs.h

```

193 #define NSS_BUFLLEN_NETGROUP              ((MAXHOSTNAMELEN * 2 + LOGNAME_MAX_ILLUMOS + 3)
194 #define NSS_BUFLLEN_NETGROUP              (MAXHOSTNAMELEN * 2 + LOGNAME_MAX + 3)
195 #define NSS_BUFLLEN_NETWORKS             NSS_LINELEN_NETWORKS /* == ? + 35 * 4 */
196 #define NSS_BUFLLEN_PASSWD               NSS_LINELEN_PASSWD
197 #define NSS_BUFLLEN_PROJECT               (NSS_LINELEN_PROJECT + 800 * sizeof (char *))
198 #define NSS_BUFLLEN_PROTOCOLS            NSS_LINELEN_PROTOCOLS /* == ? + 35 * 4 */
199 #define NSS_BUFLLEN_PUBLICKEY            NSS_LINELEN_PUBLICKEY
200 #define NSS_BUFLLEN_RPC                   NSS_LINELEN_RPC /* == ? + 35 * 4 */
201 #define NSS_BUFLLEN_SERVICES              NSS_LINELEN_SERVICES /* == ? + 35 * 4 */
202 #define NSS_BUFLLEN_SHADOW                NSS_LINELEN_SHADOW
203 #define NSS_BUFLLEN_ETHERS               NSS_LINELEN_ETHERS
204 #define NSS_BUFLLEN_BOOTPARAMS           NSS_LINELEN_BOOTPARAMS

205 #define NSS_BUFLLEN_ATTRDB               NSS_LINELEN_ATTRDB

207 #define NSS_BUFLLEN_AUDITUSER            NSS_BUFLLEN_ATTRDB
208 #define NSS_BUFLLEN_AUTHATTR            NSS_BUFLLEN_ATTRDB
209 #define NSS_BUFLLEN_EXECATTR            NSS_BUFLLEN_ATTRDB
210 #define NSS_BUFLLEN_PROFATTR            NSS_BUFLLEN_ATTRDB
211 #define NSS_BUFLLEN_USERATTR              ((NSS_BUFLLEN_ATTRDB) * 8)

213 #define NSS_BUFLLEN_TSOL                  NSS_LINELEN_TSOL
214 #define NSS_BUFLLEN_TSOL_TP              NSS_BUFLLEN_TSOL
215 #define NSS_BUFLLEN_TSOL_RH              NSS_BUFLLEN_TSOL
216 #define NSS_BUFLLEN_TSOL_ZC              NSS_BUFLLEN_TSOL

218 /*
219 * Default cache door buffer size (2x largest buffer)
220 */

222 #define NSS_BUFLLEN_DOOR                  ((NSS_BUFSIZ) * 16)

224 /*
225 * Arguments and results, passed between the frontends and backends for
226 * the well-known databases.  The getXbyY_r() and getXent_r() routines
227 * use a common format that is further described below; other routines
228 * use their own formats.
229 */

231 /*
232 * The nss_str2ent_t routine is the data marshaller for the nssswitch.
233 * it converts 'native files' format into 'entry' format as part of the
234 * return processing for a getXbyY interface.
235 *
236 * The nss_groupstr_t routine does the real work for any backend
237 * that can supply a netgroup entry as a string in /etc/group format
238 */
239 #if defined(__STDC__)
240 typedef int (*nss_str2ent_t)(const char *in, int inlen,
241 void *ent, char *buf, int buflen);
242 #endif

243 struct nss_groupstr_t; /* forward definition */
244 typedef nss_status_t (*nss_groupstr_t)(const char *instr, int inlen,
245 struct nss_groupstr_t *);
246 #else
247 typedef int (*nss_str2ent_t)();
248 typedef nss_status_t (*nss_groupstr_t)();
249 #endif

251 /*
252 * The initgroups() function [see initgroups(3c)] needs to find all the
253 * groups to which a given user belongs.  To do this it calls
254 * _getgroupsbymember(), which is part of the frontend for the "group"
255 * database.
256 * We want the same effect as if we used getgrent_r() to enumerate the
257 * entire groups database (possibly from multiple sources), but getgrent_r()

```

4

```
258 * is too inefficient. Most backends can do better if they know they're
259 * meant to scan all groups; hence there's a separate backend operation,
260 * NSS_DBOP_GROUP_BYMEMBER, which uses the nss_groupsbymem struct.
261 * Note that the normal return-value from such a backend, even when it
262 * successfully finds matching group entries, is NSS_NOTFOUND, because
263 * this tells the switch engine to keep searching in any more sources.
264 * In fact, the backends only return NSS_SUCCESS if they find enough
265 * matching entries that the gid_array is completely filled, in which
266 * case the switch engine should stop searching.
267 * If the force_slow_way field is set, the backend should eschew any cached
268 * information (e.g. the YP netid.byname map or the NIS+ cred.org_dir table)
269 * and should instead grind its way through the group map/table/whatever.
270 */

272 struct nss_groupsbymem {                               /* For _getgroupsbymember() */
273 /* in: */
274     const char    *username;
275     gid_t         *gid_array;
276     int           maxgids;
277     int           force_slow_way;
278     nss_str2ent_t str2ent;
279     nss_groupstr_t process_cstr;

281 /* in_out: */
282     int           numgids;
283 };
_____unchanged_portion_omitted_
```

```

*****
6486 Wed Apr 3 09:33:12 2013
new/usr/src/lib/libbssm/common/audit_ftpdc
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
25 */

27 #include <sys/types.h>
28 #include <sys/param.h>
29 #include <stdio.h>
30 #include <sys/fcntl.h>
31 #include <stdlib.h>
32 #include <string.h>
33 #include <syslog.h>
34 #include <unistd.h>

36 #include <sys/socket.h>
37 #include <sys/sockio.h>
38 #include <netinet/in.h>
39 #include <tsol/label.h>

41 #include <bsm/audit.h>
42 #include <bsm/audit_record.h>
43 #include <bsm/audit_uevents.h>
44 #include <bsm/libbssm.h>
45 #include <bsm/audit_private.h>

47 #include <locale.h>
48 #include <pwd.h>
49 #include <generic.h>

51 #define BAD_PASSWD (1)
52 #define UNKNOWN_USER (2)
53 #define EXCLUDED_USER (3)
54 #define NO_ANONYMOUS (4)
55 #define MISC_FAILURE (5)

57 static char luser[LOGNAME_MAX_ILLUMOS + 1];
58 static char luser[LOGNAME_MAX + 1];

59 static void generate_record(char *, int, char *);

```

```

60 static int selected(uid_t, char *, au_event_t, int);

62 void
63 audit_ftpdc_bad_pw(char *uname)
64 {
65     if (cannot_audit(0)) {
66         return;
67     }
68     (void) strncpy(luser, uname, LOGNAME_MAX_ILLUMOS);
69     (void) strncpy(luser, uname, LOGNAME_MAX);
70     generate_record(luser, BAD_PASSWD, dgettext(bsm_dom, "bad password"));
71 }

73 void
74 audit_ftpdc_unknown(char *uname)
75 {
76     if (cannot_audit(0)) {
77         return;
78     }
79     (void) strncpy(luser, uname, LOGNAME_MAX_ILLUMOS);
80     (void) strncpy(luser, uname, LOGNAME_MAX);
81     generate_record(luser, UNKNOWN_USER, dgettext(bsm_dom, "unknown user"));
82 }

84 void
85 audit_ftpdc_excluded(char *uname)
86 {
87     if (cannot_audit(0)) {
88         return;
89     }
90     (void) strncpy(luser, uname, LOGNAME_MAX_ILLUMOS);
91     (void) strncpy(luser, uname, LOGNAME_MAX);
92     generate_record(luser, EXCLUDED_USER, dgettext(bsm_dom,
93 "excluded user"));
94 }

_____unchanged_portion_omitted_____

114 void
115 audit_ftpdc_success(char *uname)
116 {
117     if (cannot_audit(0)) {
118         return;
119     }
120     (void) strncpy(luser, uname, LOGNAME_MAX_ILLUMOS);
121     (void) strncpy(luser, uname, LOGNAME_MAX);
122     generate_record(luser, 0, "");
123 }

_____unchanged_portion_omitted_____

```

new/usr/src/lib/libc/port/gen/getlogin.c

1

```
*****
2935 Wed Apr 3 09:33:12 2013
new/usr/src/lib/libc/port/gen/getlogin.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 */

22 /*
23  * Copyright (c) 2013 Gary Mills
24  *
25  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
26  * Use is subject to license terms.
27  */

29 /*      Copyright (c) 1988 AT&T */
30 /*      All Rights Reserved      */

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

32 #pragma weak _getlogin = getlogin
33 #pragma weak _getlogin_r = getlogin_r

35 #include "lint.h"
36 #include <sys/types.h>
37 #include <sys/stat.h>
38 #include <fcntl.h>
39 #include <string.h>
40 #include <stdlib.h>
41 #include <limits.h>
42 #include "utmpx.h"
43 #include <unistd.h>
44 #include <errno.h>
45 #include <thread.h>
46 #include <synch.h>
47 #include <mtlib.h>
48 #include "tsd.h"

50 /*
51  * Use the full length of a login name, from LOGIN_NAME_MAX .
52  * The utmpx interface provides for a 32 character login name.
53  * XXX - _POSIX_LOGIN_NAME_MAX limits the length of a login name. The utmpx
54  * interface provides for a 32 character login name, but for the sake of
55  * compatibility, we are still using the old utmp-imposed limit.
56  */
55 /*
```

new/usr/src/lib/libc/port/gen/getlogin.c

2

```
56 * POSIX.1c Draft-6 version of the function getlogin_r.
57 * It was implemented by Solaris 2.3.
58 */
59 char *
60 getlogin_r(char *answer, int namelen)
61 {
62     int                uf;
63     off64_t           me;
64     struct futmpx     ubuf;

66     /* Required minimum */
67     if (namelen < _POSIX_LOGIN_NAME_MAX) {
68         errno = ERANGE;
69         return (NULL);
70     }

72     if ((me = (off64_t)ttyslot()) < 0)
73         return (NULL);
74     if ((uf = open64(UTMPX_FILE, 0)) < 0)
75         return (NULL);
76     (void) lseek64(uf, me * sizeof (ubuf), SEEK_SET);
77     if (read(uf, &ubuf, sizeof (ubuf)) != sizeof (ubuf)) {
78         (void) close(uf);
79         return (NULL);
80     }
81     (void) close(uf);
82     if (ubuf.ut_user[0] == '\0')
83         return (NULL);

85     /* Insufficient buffer size */
86     if (namelen < strlen(&ubuf.ut_user[0], LOGIN_NAME_MAX - 1)) {
87         errno = ERANGE;
88         return (NULL);
89     }
90     (void) strncpy(&answer[0], &ubuf.ut_user[0],
91         LOGIN_NAME_MAX - 1);
92     answer[LOGIN_NAME_MAX - 1] = '\0';
85     _POSIX_LOGIN_NAME_MAX - 1);
86     answer[_POSIX_LOGIN_NAME_MAX - 1] = '\0';
93     return (&answer[0]);
94 }

unchanged_portion_omitted

117 char *
118 getlogin(void)
119 {
120     char *answer = tsdalloc(_T_LOGIN, LOGIN_NAME_MAX, NULL);
114     char *answer = tsdalloc(_T_LOGIN, _POSIX_LOGIN_NAME_MAX, NULL);

122     if (answer == NULL)
123         return (NULL);
124     return (getlogin_r(answer, LOGIN_NAME_MAX));
118     return (getlogin_r(answer, _POSIX_LOGIN_NAME_MAX));
125 }

unchanged_portion_omitted
```

```

*****
12091 Wed Apr  3 09:33:13 2013
new/usr/src/lib/libc/port/gen/sysconf.c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
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 */

22 /*
23 * Copyright (c) 2013 Gary Mills
24 *
25 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26 * Use is subject to license terms.
27 */

29 /*      Copyright (c) 1988 AT&T */
30 /*      All Rights Reserved */

32 /* sysconf(3C) - returns system configuration information */

34 #pragma weak _sysconf = sysconf

36 #include "lint.h"
37 #include <mtlib.h>
38 #include <sys/types.h>
39 #include <unistd.h>
40 #include <sys/sysconfig.h>
41 #include <limits.h>
42 #include <time.h>
43 #include <errno.h>
44 #include <nss_dbdefs.h>
45 #include <thread.h>
46 #include <xti.h>
47 #include "libc.h"
48 #include "xpg6.h"

50 /* from nss_common.c */
51 extern size_t _nss_get_bufsizes(int);

53 long
54 sysconf(int name)
55 {
56     static int _pagesize = 0;
57     static int _hz = 0;
58     static pid_t _maxpid = 0;
59     static int _stackprot = 0;
60     static int _ngroups_max;

```

```

61     extern int __xpg4;

63     switch (name) {
64     default:
65         errno = EINVAL;
66         return (-1L);

68     case _SC_ARG_MAX:
69         return ((long)ARG_MAX);

71     case _SC_CLK_TCK:
72         if (_hz <= 0)
73             _hz = _sysconfig(_CONFIG_CLK_TCK);
74         return (_hz);

76     case _SC_JOB_CONTROL:
77         return ((long)_POSIX_JOB_CONTROL);

79     case _SC_SAVED_IDS:
80         return ((long)_POSIX_SAVED_IDS);

82     case _SC_CHILD_MAX:
83         return (_sysconfig(_CONFIG_CHILD_MAX));

85     case _SC_NGROUPS_MAX:
86         if (_ngroups_max <= 0)
87             _ngroups_max = _sysconfig(_CONFIG_NGROUPS);
88         return (_ngroups_max);

90     case _SC_OPEN_MAX:
91         return (_sysconfig(_CONFIG_OPEN_FILES));

93     case _SC_VERSION:
94         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
95             return (200112L);
96         else
97             return (199506L);

99     case _SC_PAGESIZE:
100         if (_pagesize <= 0)
101             _pagesize = _sysconfig(_CONFIG_PAGESIZE);
102         return (_pagesize);

104     case _SC_XOPEN_VERSION:
105         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
106             return (600L);
107         else if (__xpg4 == 0)
108             return (_sysconfig(_CONFIG_XOPEN_VER));
109         else
110             return (4L);

112     case _SC_XOPEN_XCU_VERSION:
113         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
114             return (600L);
115         else
116             return (4L);

118     /*
119     * old value for pre XPG5 conformant systems to match
120     * getpass() length.
121     * XPG5 special cased with __sysconf_xpg5()
122     * new value for default and modern XPG systems.
123     */
124     case _SC_PASS_MAX:
125         if ((__xpg4 == 1) &&
126             (!(__xpg6 & _C99SUSv3_XPG6_sysconf_version)))

```

```

127         return ((long)_PASS_MAX_XPG);
128     else
129         return ((long)_PASS_MAX);

131     case _SC_LOGNAME_MAX:
132         return ((long)LOGNAME_MAX);

134     case _SC_STREAM_MAX:
135         return (_sysconf(_CONFIG_OPEN_FILES));

137     case _SC_TZNAME_MAX:
138         return (-1L);

140     case _SC_NPROCESSORS_CONF:
141         return (_sysconf(_CONFIG_NPROC_CONF));

143     case _SC_NPROCESSORS_ONLN:
144         return (_sysconf(_CONFIG_NPROC_ONLN));

146     case _SC_NPROCESSORS_MAX:
147         return (_sysconf(_CONFIG_NPROC_MAX));

149     case _SC_STACK_PROT:
150         if (_stackprot == 0)
151             _stackprot = _sysconf(_CONFIG_STACK_PROT);
152         return (_stackprot);

154     /* POSIX.4 names */

156     /*
157     * Each of the following also have _POSIX_* symbols
158     * defined in <unistd.h>. Values here should align
159     * with values in the header. Up until the SUSv3 standard
160     * we defined these simply as 1. With the introduction
161     * of the new revision, these were changed to 200112L.
162     * The standard allows us to change the value, however,
163     * we have kept both values in case application programs
164     * are relying on the previous value even though an
165     * application doing so is technically wrong.
166     */
167     case _SC_ASYNCHRONOUS_IO:
168     case _SC_FSYNC:
169     case _SC_MAPPED_FILES:
170     case _SC_MEMLOCK:
171     case _SC_MEMLOCK_RANGE:
172     case _SC_MEMORY_PROTECTION:
173     case _SC_MESSAGE_PASSING:
174     case _SC_PRIORITY_SCHEDULING:
175     case _SC_REALTIME_SIGNALS:
176     case _SC_SEMAPHORES:
177     case _SC_SHARED_MEMORY_OBJECTS:
178     case _SC_SYNCHRONIZED_IO:
179     case _SC_TIMERS:
180         if (__xpg6 & _C99SUSv3_mode_ON)
181             return (200112L);
182         else
183             return (1L);

185     case _SC_PRIORITIZED_IO:
186 #ifdef _POSIX_PRIORITIZED_IO
187         return (1L);
188 #else
189         return (-1L);
190 #endif

192     case _SC_AIO_LISTIO_MAX:

```

```

193         return (_sysconf(_CONFIG_AIO_LISTIO_MAX));

195     case _SC_AIO_MAX:
196         return (_sysconf(_CONFIG_AIO_MAX));

198     case _SC_AIO_PRIO_DELTA_MAX:
199         return (_sysconf(_CONFIG_AIO_PRIO_DELTA_MAX));

201     case _SC_DELAYTIMER_MAX:
202         return (_sysconf(_CONFIG_DELAYTIMER_MAX));

204     case _SC_MQ_OPEN_MAX:
205         return (_sysconf(_CONFIG_MQ_OPEN_MAX));

207     case _SC_MQ_PRIO_MAX:
208         return (_sysconf(_CONFIG_MQ_PRIO_MAX));

210     case _SC_RTSIG_MAX:
211         return (_sysconf(_CONFIG_RTSIG_MAX));

213     case _SC_SEM_NSEMS_MAX:
214         return (_sysconf(_CONFIG_SEM_NSEMS_MAX));

216     case _SC_SEM_VALUE_MAX:
217         return (_sysconf(_CONFIG_SEM_VALUE_MAX));

219     case _SC_SIGQUEUE_MAX:
220         return (_sysconf(_CONFIG_SIGQUEUE_MAX));

222     case _SC_SIGRT_MAX:
223         return (_sysconf(_CONFIG_SIGRT_MAX));

225     case _SC_SIGRT_MIN:
226         return (_sysconf(_CONFIG_SIGRT_MIN));

228     case _SC_TIMER_MAX:
229         return (_sysconf(_CONFIG_TIMER_MAX));

231     case _SC_PHYS_PAGES:
232         return (_sysconf(_CONFIG_PHYS_PAGES));

234     case _SC_AVPHYS_PAGES:
235         return (_sysconf(_CONFIG_AVPHYS_PAGES));

237     /* XPG4/POSIX.1-1990/POSIX.2-1992 names */
238     case _SC_2_C_BIND:
239         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
240             return (200112L);
241         else
242             return (1L);

244     case _SC_2_CHAR_TERM:
245         return ((long)_POSIX2_CHAR_TERM);

247     case _SC_2_C_DEV:
248         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
249             return (200112L);
250         else
251             return (1L);

253     case _SC_2_C_VERSION:
254         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
255             return (200112L);
256         else
257             return (199209L);

```



```

259     case _SC_2_FORT_DEV:
260         return (-1L);

262     case _SC_2_FORT_RUN:
263         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
264             return (200112L);
265         else
266             return (1L);

268     case _SC_2_LOCALEDEF:
269         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
270             return (200112L);
271         else
272             return (1L);

274     case _SC_2_SW_DEV:
275         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
276             return (200112L);
277         else
278             return (1L);

280     case _SC_2_UPE:
281         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
282             return (200112L);
283         else
284             return (1L);

286     case _SC_2_VERSION:
287         if (__xpg6 & _C99SUSv3_XPG6_sysconf_version)
288             return (200112L);
289         else
290             return (199209L);

292     case _SC_BC_BASE_MAX:
293         return ((long)BC_BASE_MAX);

295     case _SC_BC_DIM_MAX:
296         return ((long)BC_DIM_MAX);

298     case _SC_BC_SCALE_MAX:
299         return ((long)BC_SCALE_MAX);

301     case _SC_BC_STRING_MAX:
302         return ((long)BC_STRING_MAX);

304     case _SC_COLL_WEIGHTS_MAX:
305         return ((long)COLL_WEIGHTS_MAX);

307     case _SC_EXPR_NEST_MAX:
308         return ((long)EXPR_NEST_MAX);

310     case _SC_LINE_MAX:
311         return ((long)LINE_MAX);

313     case _SC_RE_DUP_MAX:
314         return ((long)RE_DUP_MAX);

316     case _SC_XOPEN_CRYPT:
317         return (1L);

319     case _SC_XOPEN_ENH_I18N:
320         return ((long)_XOPEN_ENH_I18N);

322     case _SC_XOPEN_SHM:
323         return ((long)_XOPEN_SHM);

```

```

325         /* XPG4v2 (SUS) names */
326     case _SC_XOPEN_UNIX:
327         return (1L);

329     case _SC_XOPEN_LEGACY:
330         return (1L);

332     case _SC_ATEXIT_MAX:
333         return (-1L);

335     case _SC_IOV_MAX:
336         return ((long)IOV_MAX);

338     case _SC_T_IOV_MAX:
339         return ((long)T_IOV_MAX);

341         /* XPG5 (SUSv2) names */
342     case _SC_XOPEN_REALTIME:
343         return (1L);

345     case _SC_XOPEN_REALTIME_THREADS:
346 #if defined(_POSIX_THREAD_PRIORITY_SCHEDULING) && \
347     defined(_POSIX_THREAD_PRIO_INHERIT) && \
348     defined(_POSIX_THREAD_PRIO_PROTECT)
349         return (1L);
350 #else
351         return (-1L);
352 #endif

354     case _SC_XBS5_ILP32_OFF32:
355         return (1L);

357     case _SC_XBS5_ILP32_OFFBIG:
358         return (1L);

360     case _SC_XBS5_LP64_OFF64:
361         return (1L);

363     case _SC_XBS5_LPBIG_OFFBIG:
364         return (1L);

366         /* POSIX.1c names */
367     case _SC_THREAD_DESTRUCTOR_ITERATIONS:
368         return (-1L);

370     case _SC_GETGR_R_SIZE_MAX:
371         return ((long)_nss_get_bufsizes(_SC_GETGR_R_SIZE_MAX));

373     case _SC_GETPW_R_SIZE_MAX:
374         return ((long)_NSS_BUFLEN_PASSWD);

376     case _SC_LOGIN_NAME_MAX:
377         return ((long)(LOGIN_NAME_MAX));
375         return ((long)(LOGNAME_MAX + 1));

379     case _SC_THREAD_KEYS_MAX:
380         return (-1L);

382     case _SC_THREAD_STACK_MIN:
383         return ((long)thr_min_stack());

385     case _SC_THREAD_THREADS_MAX:
386         return (-1L);

388     case _SC_TTY_NAME_MAX:
389         return ((long)TTYNAME_MAX);

```

```

391     case _SC_BARRIERS:
392         return ((long)_POSIX_BARRIERS);

394     case _SC_CLOCK_SELECTION:
395         return ((long)_POSIX_CLOCK_SELECTION);

397     case _SC_MONOTONIC_CLOCK:
398         return ((long)_POSIX_MONOTONIC_CLOCK);

400     case _SC_SPAWN:
401         return ((long)_POSIX_SPAWN);

403     case _SC_SPIN_LOCKS:
404         return ((long)_POSIX_SPIN_LOCKS);

406     case _SC_THREADS:
407     case _SC_THREAD_ATTR_STACKADDR:
408     case _SC_THREAD_ATTR_STACKSIZE:
409     case _SC_THREAD_PRIORITY_SCHEDULING:
410     case _SC_THREAD_PRIO_INHERIT:
411     case _SC_THREAD_PRIO_PROTECT:
412     case _SC_THREAD_PROCESS_SHARED:
413     case _SC_THREAD_SAFE_FUNCTIONS:
414         if (__xpg6 & _C99SUSv3_mode_ON)
415             return (200112L);
416         else
417             return (1L);

419     case _SC_TIMEOUTS:
420         return ((long)_POSIX_TIMEOUTS);

422     /* 1216676 - cache info */
423     case _SC_COHER_BLKSZ:
424         return (_sysconf(_CONFIG_COHERENCY));

426     case _SC_SPLIT_CACHE:
427         return (_sysconf(_CONFIG_SPLIT_CACHE));

429     case _SC_ICACHE_SZ:
430         return (_sysconf(_CONFIG_ICACHESZ));

432     case _SC_DCACHE_SZ:
433         return (_sysconf(_CONFIG_DCACHESZ));

435     case _SC_ICACHE_LINESZ:
436         return (_sysconf(_CONFIG_ICACHELINESZ));

438     case _SC_DCACHE_LINESZ:
439         return (_sysconf(_CONFIG_DCACHELINESZ));

441     case _SC_ICACHE_BLKSZ:
442         return (_sysconf(_CONFIG_ICACHEBLKSZ));

444     case _SC_DCACHE_BLKSZ:
445         return (_sysconf(_CONFIG_DCACHEBLKSZ));

447     case _SC_DCACHE_TBLKSZ:
448         return (_sysconf(_CONFIG_DCACHE_TBLKSZ));

450     case _SC_ICACHE_ASSOC:
451         return (_sysconf(_CONFIG_ICACHE_ASSOC));

453     case _SC_DCACHE_ASSOC:
454         return (_sysconf(_CONFIG_DCACHE_ASSOC));

```

```

456     case _SC_MAXPID:
457         if (_maxpid <= 0)
458             _maxpid = _sysconf(_CONFIG_MAXPID);
459         return (_maxpid);

461     case _SC_CPUID_MAX:
462         return (_sysconf(_CONFIG_CPUID_MAX));

464     case _SC_EPHID_MAX:
465         return (_sysconf(_CONFIG_EPHID_MAX));

467     /* UNIX 03 names - XPG6/SUSv3/POSIX.1-2001 */

469     case _SC_REGEX:
470         return ((long)_POSIX_REGEX);

472     case _SC_SHELL:
473         return ((long)_POSIX_SHELL);

475     case _SC_ADVISORY_INFO:
476         return ((long)_POSIX_ADVISORY_INFO);

478     case _SC_HOST_NAME_MAX:
479         return ((long)_POSIX_HOST_NAME_MAX);

481     case _SC_READER_WRITER_LOCKS:
482         return ((long)_POSIX_READER_WRITER_LOCKS);

484     case _SC_IPV6:
485         return ((long)_POSIX_IPV6);

487     case _SC_RAW_SOCKETS:
488         return ((long)_POSIX_RAW_SOCKETS);

490     case _SC_XOPEN_STREAMS:
491         return ((long)_XOPEN_STREAMS);

493     case _SC_SYMLINK_MAX:
494         return (_sysconf(_CONFIG_SYMLINK_MAX));

496     case _SC_V6_ILP32_OFF32:
497         return (1L);

499     case _SC_V6_ILP32_OFFBIG:
500         return (1L);

502     case _SC_V6_LP64_OFF64:
503         return (1L);

505     case _SC_V6_LP64_OFFBIG:
506         return (1L);

508     /* Unsupported UNIX 03 options */
509     case _SC_2_PBS:
510     case _SC_2_PBS_ACCOUNTING:
511     case _SC_2_PBS_CHECKPOINT:
512     case _SC_2_PBS_LOCATE:
513     case _SC_2_PBS_MESSAGE:
514     case _SC_2_PBS_TRACK:
515     case _SC_CPUTIME:
516     case _SC_SPORADIC_SERVER:
517     case _SC_SS_REPL_MAX:
518     case _SC_THREAD_CPUTIME:
519     case _SC_THREAD_SPORADIC_SERVER:
520     case _SC_TRACE:
521     case _SC_TRACE_EVENT_FILTER:

```

```
522         case _SC_TRACE_EVENT_NAME_MAX:
523         case _SC_TRACE_INHERIT:
524         case _SC_TRACE_LOG:
525         case _SC_TRACE_NAME_MAX:
526         case _SC_TRACE_SYS_MAX:
527         case _SC_TRACE_USER_EVENT_MAX:
528         case _SC_TYPED_MEMORY_OBJECTS:
529             return (-1L);
530     }
531 }
unchanged_portion_omitted
```

new/usr/src/lib/nsswitch/ldap/common/getnetgrent.c

1

23360 Wed Apr 3 09:33:13 2013

new/usr/src/lib/nsswitch/ldap/common/getnetgrent.c

2989 Eliminate use of LOGNAME_MAX in ON

1166 useradd have warning with name more 8 chars

```
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 */

29 #include <syslog.h>
30 #include "ldap_common.h"

32 /* netgroup attributes filters */
33 #define _N_TRIPLE "nisnetgrouptriple"
34 #define _N_MEMBER "membernisnetgroup"

36 #define PRINT_VAL(a) (((a).argc == 0) || ((a).argv == NULL) || \
37 ((a).argv[0] == NULL)) ? "*" : (a).argv[0]
38 #define ISNULL(a) (a == NULL ? "<NULL>" : a)
39 #define MAX_DOMAIN_LEN 1024
40 #define MAX_TRIPLE_LEN (MAXHOSTNAMELEN + LOGNAME_MAX_ILLUMOS + \
41 (MAXHOSTNAMELEN + LOGNAME_MAX + \
42 MAX_DOMAIN_LEN + 5))

43 #define _F_SETMEMBER "(&(objectClass=nisNetGroup)(cn=%s))"
44 #define _F_SETMEMBER_SSD "(&(%s)(cn=%s))"

46 #define N_HASH 257
47 #define COMMA ','

49 static const char *netgrent_attrs[] = {
50     _N_TRIPLE,
51     _N_MEMBER,
52     (char *)NULL
53 };
54
55 unchanged_portion_omitted
```

16720 Wed Apr 3 09:33:13 2013

new/usr/src/man/man1m/prstat.1m

2989 Eliminate use of LOGNAME_MAX in ON

1166 useradd have warning with name more 8 chars

```

1  \" te
2  .\" Copyright (c) 2013 Gary Mills
3  .\" Copyright (c) 2006, 2009 Sun Microsystems, Inc. All Rights Reserved.
4  .\" The contents of this file are subject to the terms of the Common Development
5  .\" See the License for the specific language governing permissions and limitat
6  .\" the fields enclosed by brackets \"[]\" replaced with your own identifying info
7  .TH PRSTAT 1M \"Jun 25, 2009\"
8  .SH NAME
9  prstat \- report active process statistics
10 .SH SYNOPSIS
11 .LP
12 .nf
13 \fBprstat\fR [\fB-acHJLmRrtTv\fR] [\fB-d\fR u | d] [\fB-C\fR \fIpsrsetlist\fR] [
14  [\fB-j\fR \fIprojlist\fR] [\fB-k\fR \fItasklist\fR] [\fB-n\fR \fIntop\fR[, \fB-p\fR
15  [\fB-p\fR \fIpidlist\fR] [\fB-P\fR \fIcpulist\fR] [\fB-s\fR \fIkey\fR | \fB
16  [\fB-u\fR \fIeuidlist\fR] [\fB-U\fR \fIuidlist\fR] [\fB-z\fR \fIzoneidlist\
17  [\fIinterval\fR [\fIcount\fR]]
18 .fi
20 .SH DESCRIPTION
21 .sp
22 .LP
23 The \fBprstat\fR utility iteratively examines all active processes on the
24 system and reports statistics based on the selected output mode and sort order.
25 \fBprstat\fR provides options to examine only processes matching specified
26 \fBPID\fRs, \fBUID\fRs, zone \fBID\fRs, \fBBCPU\fR \fBID\fRs, and processor set
27 \fBID\fRs.
28 .sp
29 .LP
30 The \fB-j\fR, \fB-k\fR, \fB-C\fR, \fB-p\fR, \fB-P\fR, \fB-u\fR, \fB-U\fR, and
31 \fB-z\fR options accept lists as arguments. Items in a list can be either
32 separated by commas or enclosed in quotes and separated by commas or spaces.
33 .sp
34 .LP
35 If you do not specify an option, \fBprstat\fR examines all processes and
36 reports statistics sorted by \fBCPU\fR usage.
37 .SH OPTIONS
38 .sp
39 .LP
40 The following options are supported:
41 .sp
42 .ne 2
43 .na
44 \fB\fB-a\fR\fR
45 .ad
46 .sp .6
47 .RS 4n
48 Report information about processes and users. In this mode \fBprstat\fR
49 displays separate reports about processes and users at the same time.
50 .RE
52 .sp
53 .ne 2
54 .na
55 \fB\fB-c\fR\fR
56 .ad
57 .sp .6
58 .RS 4n
59 Print new reports below previous reports instead of overprinting them.
60 Long names are not truncated in this mode.

```

```

61 .RE
63 .sp
64 .ne 2
65 .na
66 \fB\fB-C\fR \fIpsrsetlist\fR\fR
67 .ad
68 .sp .6
69 .RS 4n
70 Report only processes or lwps that are bound to processor sets in the given
71 list. Each processor set is identified by an integer as reported by
72 \fBpsrset\fR(1M). The load averages displayed are the sum of the load averages
73 of the specified processor sets (see \fBpsrset_getloadavg\fR(3C)). Processes with
74 one or more LWPs bound to processor sets in the given list are reported even
75 when the \fB-L\fR option is not used.
76 .RE
78 .sp
79 .ne 2
80 .na
81 \fB\fB-d\fR \fBu | d\fR\fR
82 .ad
83 .sp .6
84 .RS 4n
85 Specify \fBu\fR for a printed representation of the internal representation of
86 time. See \fBtime\fR(2). Specify \fBd\fR for standard date format. See
87 \fBdate\fR(1).
88 .RE
90 .sp
91 .ne 2
92 .na
93 \fB\fB-h\fR \fIilgrplist\fR\fR
94 .ad
95 .sp .6
96 .RS 4n
97 Report only processes or lwps whose home \fIilgroup\fR is in the given list of
98 \fIilgroups\fR. No processes or lwps will be listed for invalid \fIilgroups\fR.
99 .RE
101 .sp
102 .ne 2
103 .na
104 \fB\fB-H\fR\fR
105 .ad
106 .sp .6
107 .RS 4n
108 Report information about home \fIilgroup\fR. In this mode, \fBprstat\fR adds an
109 extra column showing process or lwps home \fIilgroup\fR with the header LGRP.
110 .RE
112 .sp
113 .ne 2
114 .na
115 \fB\fB-j\fR \fIprojlist\fR\fR
116 .ad
117 .sp .6
118 .RS 4n
119 Report only processes or lwps whose project \fBID\fR is in the given list. Each
120 project \fBID\fR can be specified as either a project name or a numerical
121 project \fBID\fR. See \fBproject\fR(4).
122 .RE
124 .sp
125 .ne 2
126 .na

```

```

127 \fB\fB-J\fR\fR
128 .ad
129 .sp .6
130 .RS 4n
131 Report information about processes and projects. In this mode \fBprstat\fR
132 displays separate reports about processes and projects at the same time.
133 A trailing asterisk marks a long name that has been truncated
134 to fit the column.
135 .RE

137 .sp
138 .ne 2
139 .na
140 \fB\fB-k\fR \fItasklist\fR\fR
141 .ad
142 .sp .6
143 .RS 4n
144 Report only processes or lwps whose task \fBID\fR is in \fItasklist\fR.
145 .RE

147 .sp
148 .ne 2
149 .na
150 \fB\fB-L\fR\fR
151 .ad
152 .sp .6
153 .RS 4n
154 Report statistics for each light-weight process (\fBLWP\fR). By default,
155 \fBprstat\fR reports only the number of \fBLWP\fRs for each process.
156 .RE

158 .sp
159 .ne 2
160 .na
161 \fB\fB-m\fR\fR
162 .ad
163 .sp .6
164 .RS 4n
165 Report microstate process accounting information. In addition to all fields
166 listed in \fB-v\fR mode, this mode also includes the percentage of time the
167 process has spent processing system traps, text page faults, data page faults,
168 waiting for user locks and waiting for \fBCPU\fR (latency time).
169 .RE

171 .sp
172 .ne 2
173 .na
174 \fB\fB-n\fR \fIntop\fR[\fI,nbottom\fR]\fR
175 .ad
176 .sp .6
177 .RS 4n
178 Restrict number of output lines. The \fIntop\fR argument determines how many
179 lines of process or \fBlwp\fR statistics are reported, and the \fInbottom\fR
180 argument determines how many lines of user, task, or projects statistics are
181 reported if the \fB-a\fR, \fB-t\fR, \fB-T\fR, or \fB-J\fR options are
182 specified. By default, \fBprstat\fR displays as many lines of output that fit
183 in a window or terminal. When you specify the \fB-c\fR option or direct the
184 output to a file, the default values for \fBntop\fR and \fBnbottom\fR are
185 \fB15\fR and \fB5\fR.
186 .RE

188 .sp
189 .ne 2
190 .na
191 \fB\fB-p\fR \fIpidlist\fR\fR
192 .ad

```

```

193 .sp .6
194 .RS 4n
195 Report only processes whose process \fBID\fR is in the given list.
196 .RE

198 .sp
199 .ne 2
200 .na
201 \fB\fB-P\fR \fIcpulist\fR\fR
202 .ad
203 .sp .6
204 .RS 4n
205 Report only processes or \fBlwp\fRs which have most recently executed on a
206 \fBCPU\fR in the given list. Each \fBCPU\fR is identified by an integer as
207 reported by \fBpsrinfo\fR(1M).
208 .RE

210 .sp
211 .ne 2
212 .na
213 \fB\fB-R\fR\fR
214 .ad
215 .sp .6
216 .RS 4n
217 Put \fBprstat\fR in the real time scheduling class. When this option is used,
218 \fBprstat\fR is given priority over time-sharing and interactive processes.
219 This option is available only for superuser.
220 .RE

222 .sp
223 .ne 2
224 .na
225 \fB\fB-r\fR\fR
226 .ad
227 .sp .6
228 .RS 4n
229 Disable lookups for user names and project names. (Note that this does not
230 apply to lookups for the \fB-j\fR, \fB-u\fR, or \fB-U\fR options.)
231 .RE

233 .sp
234 .ne 2
235 .na
236 \fB\fB-s\fR \fIkey\fR\fR
237 .ad
238 .sp .6
239 .RS 4n
240 Sort output lines (that is, processes, \fBlwp\fRs, or users) by \fIkey\fR in
241 descending order. Only one \fIkey\fR can be used as an argument.
242 .sp
243 There are five possible key values:
244 .sp
245 .ne 2
246 .na
247 \fBcpcu\fR
248 .ad
249 .sp .6
250 .RS 4n
251 Sort by process \fBCPU\fR usage. This is the default.
252 .RE

254 .sp
255 .ne 2
256 .na
257 \fBpri\fR
258 .ad

```

```

259 .sp .6
260 .RS 4n
261 Sort by process priority.
262 .RE

264 .sp
265 .ne 2
266 .na
267 \fBrs\fr
268 .ad
269 .sp .6
270 .RS 4n
271 Sort by resident set size.
272 .RE

274 .sp
275 .ne 2
276 .na
277 \fBsize\fr
278 .ad
279 .sp .6
280 .RS 4n
281 Sort by size of process image.
282 .RE

284 .sp
285 .ne 2
286 .na
287 \fBtime\fr
288 .ad
289 .sp .6
290 .RS 4n
291 Sort by process execution time.
292 .RE

294 .RE

296 .sp
297 .ne 2
298 .na
299 \fB-s\fr \fikey\fr\fr
300 .ad
301 .sp .6
302 .RS 4n
303 Sort output lines by \fikey\fr in ascending order. Possible \fikey\fr values
304 are the same as for the \fB-s\fr option. See \fB-s\fr.
305 .RE

307 .sp
308 .ne 2
309 .na
310 \fB-t\fr\fr
311 .ad
312 .sp .6
313 .RS 4n
314 Report total usage summary for each user. The summary includes the total number
315 of processes or \fBLWP\frs owned by the user, total size of process images,
316 total resident set size, total cpu time, and percentages of recent cpu time and
317 system memory.
318 .RE

320 .sp
321 .ne 2
322 .na
323 \fB-T\fr\fr
324 .ad

```

```

325 .sp .6
326 .RS 4n
327 Report information about processes and tasks. In this mode \fBprstat\fr
328 displays separate reports about processes and tasks at the same time.
329 .RE

331 .sp
332 .ne 2
333 .na
334 \fB-u\fr \fIeuidlist\fr\fr
335 .ad
336 .sp .6
337 .RS 4n
338 Report only processes whose effective user \fBID\fr is in the given list. Each
339 user \fBID\fr may be specified as either a login name or a numerical user
340 \fBID\fr.
341 .RE

343 .sp
344 .ne 2
345 .na
346 \fB-U\fr \fIuidlis\frt\fr
347 .ad
348 .sp .6
349 .RS 4n
350 Report only processes whose real user \fBID\fr is in the given list. Each user
351 \fBID\fr may be specified as either a login name or a numerical user \fBID\fr.
352 .RE

354 .sp
355 .ne 2
356 .na
357 \fB-v\fr\fr
358 .ad
359 .sp .6
360 .RS 4n
361 Report verbose process usage. This output format includes the percentage of
362 time the process has spent in user mode, in system mode, and sleeping. It also
363 includes the number of voluntary and involuntary context switches, system calls
364 and the number of signals received. Statistics that are not reported are marked
365 with the \fB-\fr sign.
366 .RE

368 .sp
369 .ne 2
370 .na
371 \fB-z\fr \fIzoneidlist\fr\fr
372 .ad
373 .sp .6
374 .RS 4n
375 Report only processes or LWPs whose zone ID is in the given list. Each zone ID
376 can be specified as either a zone name or a numerical zone ID. See
377 \fBzones\fr(5).
378 .RE

380 .sp
381 .ne 2
382 .na
383 \fB-Z\fr\fr
384 .ad
385 .sp .6
386 .RS 4n
387 Report information about processes and zones. In this mode, \fBprstat\fr
388 displays separate reports about processes and zones at the same time.
389 A trailing asterisk marks a long name that has been truncated
390 to fit the column.

```

```

391 .RE
393 .SH OUTPUT
394 .sp
395 .LP
396 The following list defines the column headings and the meanings of a
397 \fBprstat\fR report:
398 .sp
399 .ne 2
400 .na
401 \fBPID\fR
402 .ad
403 .sp .6
404 .RS 4n
405 The process \fBID\fR of the process.
406 .RE
408 .sp
409 .ne 2
410 .na
411 \fBUSERNAME\fR
412 .ad
413 .sp .6
414 .RS 4n
415 The real user (login) name or real user \fBID\fR.
416 A trailing asterisk marks a long name that has been truncated
417 to fit the column.
418 .RE
420 .sp
421 .ne 2
422 .na
423 \fBSWAP\fR
424 .ad
425 .sp .6
426 .RS 4n
427 The total virtual memory size of the process, including all mapped files and
428 devices, in kilobytes (\fBK\fR), megabytes (\fBM\fR), or gigabytes (\fBG\fR).
429 .RE
431 .sp
432 .ne 2
433 .na
434 \fBRSS\fR
435 .ad
436 .sp .6
437 .RS 4n
438 The resident set size of the process (\fBRSS\fR), in kilobytes (\fBK\fR),
439 megabytes (\fBM\fR), or gigabytes (\fBG\fR). The RSS value is an estimate
440 provided by \fBproc(4) that might underestimate the actual resident set
441 size. Users who want to get more accurate usage information for capacity
442 planning should use the \fB-x\fR option to \fBpmap(1) instead.
443 .RE
445 .sp
446 .ne 2
447 .na
448 \fBSTATE\fR
449 .ad
450 .sp .6
451 .RS 4n
452 The state of the process:
453 .sp
454 .ne 2
455 .na
456 \fBcpu\fIN\fR

```

```

457 .ad
458 .sp .6
459 .RS 4n
460 Process is running on \fBCPU\fR \fIN\fR.
461 .RE
463 .sp
464 .ne 2
465 .na
466 \fBsleeeep\fR
467 .ad
468 .sp .6
469 .RS 4n
470 Sleeping: process is waiting for an event to complete.
471 .RE
473 .sp
474 .ne 2
475 .na
476 \fBwait\fR
477 .ad
478 .sp .6
479 .RS 4n
480 Waiting: process is waiting for CPU usage to drop to the CPU-caps enforced
481 limits. See the description of \fBCPU-caps\fR in \fBresource_controls(5).
482 .RE
484 .sp
485 .ne 2
486 .na
487 \fBrun\fR
488 .ad
489 .sp .6
490 .RS 4n
491 Runnable: process in on run queue.
492 .RE
494 .sp
495 .ne 2
496 .na
497 \fBzombie\fR
498 .ad
499 .sp .6
500 .RS 4n
501 Zombie state: process terminated and parent not waiting.
502 .RE
504 .sp
505 .ne 2
506 .na
507 \fBstop\fR
508 .ad
509 .sp .6
510 .RS 4n
511 Process is stopped.
512 .RE
514 .RE
516 .sp
517 .ne 2
518 .na
519 \fBPRI\fR
520 .ad
521 .sp .6
522 .RS 4n

```



```

523 The priority of the process. Larger numbers mean higher priority.
524 .RE

526 .sp
527 .ne 2
528 .na
529 \fBNICE\fR
530 .ad
531 .sp .6
532 .RS 4n
533 Nice value used in priority computation. Only processes in certain scheduling
534 classes have a nice value.
535 .RE

537 .sp
538 .ne 2
539 .na
540 \fBTIME\fR
541 .ad
542 .sp .6
543 .RS 4n
544 The cumulative execution time for the process.
545 .RE

547 .sp
548 .ne 2
549 .na
550 \fBCPU\fR
551 .ad
552 .sp .6
553 .RS 4n
554 The percentage of recent \fBCPU\fR time used by the process. If executing in a
555 non-global \fBzone\fR and the pools facility is active, the percentage will be
556 that of the processors in the processor set in use by the pool to which the
557 \fBzone\fR is bound.
558 .RE

560 .sp
561 .ne 2
562 .na
563 \fBPROCESS\fR
564 .ad
565 .sp .6
566 .RS 4n
567 The name of the process (name of executed file).
568 .RE

570 .sp
571 .ne 2
572 .na
573 \fBLWPID\fR
574 .ad
575 .sp .6
576 .RS 4n
577 The \fBlwp\fR \fBID\fR of the \fBlwp\fR being reported.
578 .RE

580 .sp
581 .ne 2
582 .na
583 \fBNLWP\fR
584 .ad
585 .sp .6
586 .RS 4n
587 The number of \fBlwp\fRs in the process.
588 .RE

```

```

590 .sp
591 .LP
592 With the some options, in addition to a number of the column headings shown
593 above, there are:
594 .sp
595 .ne 2
596 .na
597 \fBNPROC\fR
598 .ad
599 .sp .6
600 .RS 4n
601 Number of processes in a specified collection.
602 .RE

604 .sp
605 .ne 2
606 .na
607 \fBMEMORY\fR
608 .ad
609 .sp .6
610 .RS 4n
611 Percentage of memory used by a specified collection of processes.
612 .RE

614 .sp
615 .LP
616 The following columns are displayed when the \fB-v\fR or \fB-m\fR option is
617 specified
618 .sp
619 .ne 2
620 .na
621 \fBUSR\fR
622 .ad
623 .sp .6
624 .RS 4n
625 The percentage of time the process has spent in user mode.
626 .RE

628 .sp
629 .ne 2
630 .na
631 \fBSYS\fR
632 .ad
633 .sp .6
634 .RS 4n
635 The percentage of time the process has spent in system mode.
636 .RE

638 .sp
639 .ne 2
640 .na
641 \fBTRP\fR
642 .ad
643 .sp .6
644 .RS 4n
645 The percentage of time the process has spent in processing system traps.
646 .RE

648 .sp
649 .ne 2
650 .na
651 \fBTFL\fR
652 .ad
653 .sp .6
654 .RS 4n

```

```

655 The percentage of time the process has spent processing text page faults.
656 .RE

658 .sp
659 .ne 2
660 .na
661 \fBDFL\fR
662 .ad
663 .sp .6
664 .RS 4n
665 The percentage of time the process has spent processing data page faults.
666 .RE

668 .sp
669 .ne 2
670 .na
671 \fBBLCK\fR
672 .ad
673 .sp .6
674 .RS 4n
675 The percentage of time the process has spent waiting for user locks.
676 .RE

678 .sp
679 .ne 2
680 .na
681 \fBSLPL\fR
682 .ad
683 .sp .6
684 .RS 4n
685 The percentage of time the process has spent sleeping.
686 .RE

688 .sp
689 .ne 2
690 .na
691 \fBBLAT\fR
692 .ad
693 .sp .6
694 .RS 4n
695 The percentage of time the process has spent waiting for CPU.
696 .RE

698 .sp
699 .ne 2
700 .na
701 \fBVVCX\fR
702 .ad
703 .sp .6
704 .RS 4n
705 The number of voluntary context switches.
706 .RE

708 .sp
709 .ne 2
710 .na
711 \fBICX\fR
712 .ad
713 .sp .6
714 .RS 4n
715 The number of involuntary context switches.
716 .RE

718 .sp
719 .ne 2
720 .na

```

```

721 \fBSCL\fR
722 .ad
723 .sp .6
724 .RS 4n
725 The number of system calls.
726 .RE

728 .sp
729 .ne 2
730 .na
731 \fBSIG\fR
732 .ad
733 .sp .6
734 .RS 4n
735 The number of signals received.
736 .RE

738 .sp
739 .LP
740 Under the \fB-L\fR option, one line is printed for each \fBlwp\fR in the
741 process and some reporting fields show the values for the \fBlwp\fR, not the
742 process.
743 .sp
744 .LP
745 The following column is displayed when the \fB-H\fR option is specified:
746 .sp
747 .ne 2
748 .na
749 \fBBLGRP\fR
750 .ad
751 .sp .6
752 .RS 4n
753 The home \fIgroup\fR of the process or lwp.
754 .RE

756 .SH OPERANDS
757 .sp
758 .LP
759 The following operands are supported:
760 .sp
761 .ne 2
762 .na
763 \fB\fIcount\fR\fR
764 .ad
765 .sp .6
766 .RS 4n
767 Specifies the number of times that the statistics are repeated. By default,
768 \fBprstat\fR reports statistics until a termination signal is received.
769 .RE

771 .sp
772 .ne 2
773 .na
774 \fB\fIinterval\fR\fR
775 .ad
776 .sp .6
777 .RS 4n
778 Specifies the sampling interval in seconds; the default interval is \fB5\fR
779 seconds.
780 .RE

782 .SH EXAMPLES
783 .LP
784 \fBExample 1\fR Reporting the Five Most Active Super-User Processes
785 .sp
786 .LP

```

787 The following command reports the five most active super-user processes running
788 on \fBCPU1\fR and \fBCPU2\fR:

```
790 .sp
791 .in +2
792 .nf
793 example% prstat -u root -n 5 -P 1,2 1 1
```

795 PID	USERNAME	SWAP	RSS	STATE	PRI	NICE	TIME	CPU	PROCESS/LWP
796 306	root	3024K	1448K	sleep	58	0	0:00.00	0.3%	sendmail/1
797 102	root	1600K	592K	sleep	59	0	0:00.00	0.1%	in.rdisc/1
798 250	root	1000K	552K	sleep	58	0	0:00.00	0.0%	utmpd/1
799 288	root	1720K	1032K	sleep	58	0	0:00.00	0.0%	sac/1
800 1	root	744K	168K	sleep	58	0	0:00.00	0.0%	init/1
801 TOTAL:		25,	load averages:	0.05,	0.08,	0.12			

```
802 .fi
803 .in -2
804 .sp
```

```
806 .LP
807 \fBExample 2 \fRDisplaying Verbose Process Usage Information
808 .sp
809 .LP
810 The following command displays verbose process usage information about
811 processes with lowest resident set sizes owned by users \fBroot\fR and
812 \fBjohn\fR.
```

```
814 .sp
815 .in +2
816 .nf
817 example% prstat -S rss -n 5 -vc -u root,john
```

819 PID	USERNAME	USR	SYS	TRP	TFL	DFL	LCK	SLP	LAT	VCX	ICX	SCL	SIG	PROCESS/LWP
820 1	root	0.0	0.0	-	-	-	-	100	-	0	0	0	0	init/1
821 102	root	0.0	0.0	-	-	-	-	100	-	0	0	3	0	in.rdisc/1
822 250	root	0.0	0.0	-	-	-	-	100	-	0	0	0	0	utmpd/1
823 1185	john	0.0	0.0	-	-	-	-	100	-	0	0	0	0	csch/1
824 240	root	0.0	0.0	-	-	-	-	100	-	0	0	0	0	powerd/4
825 TOTAL:		71,	load averages:	0.02,	0.04,	0.08								

```
827 .fi
828 .in -2
829 .sp
```

```
831 .SH EXIT STATUS
832 .sp
833 .LP
834 The following exit values are returned:
835 .sp
836 .ne 2
837 .na
838 \fB0\fR
839 .ad
840 .sp .6
841 .RS 4n
842 Successful completion.
843 .RE
```

```
845 .sp
846 .ne 2
847 .na
848 \fB1\fR
849 .ad
850 .sp .6
851 .RS 4n
852 An error occurred.
```

853 .RE

```
855 .SH SEE ALSO
856 .sp
857 .LP
858 \fBdate\fR(1), \fBlgrpinfo\fR(1), \fBplgrp\fR(1), \fBproc\fR(1), \fBps\fR(1),
859 \fBtime\fR(2), \fBpsrinfo\fR(1M), \fBpsrset\fR(1M), \fBsar\fR(1M),
860 \fBpset_getloadavg\fR(3C), \fBproc\fR(4), \fBproject\fR(4),
861 \fBattributes\fR(5), \fBresource_controls\fR(5), \fBzones\fR(5)
862 .SH NOTES
863 .sp
864 .LP
865 The snapshot of system usage displayed by \fBprstat\fR is true only for a
866 split-second, and it may not be accurate by the time it is displayed. When the
867 \fB-m\fR option is specified, \fBprstat\fR tries to turn on microstate
868 accounting for each process; the original state is restored when \fBprstat\fR
869 exits. See \fBproc\fR(4) for additional information about the microstate
870 accounting facility.
871 .sp
872 .LP
873 The total memory size reported in the SWAP and RSS columns for groups of
874 processes can sometimes overestimate the actual amount of memory used by
875 processes with shared memory segments.
```

```

*****
28316 Wed Apr  3 09:33:13 2013
new/usr/src/man/man3c/sysconf.3c
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
1 \" te
2 .\" Copyright (c) 2013 Gary Mills
3 .\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved.
4 .\" Portions Copyright (c) 1992, X/Open Company Limited. All Rights Reserved.
5 .\" Copyright 1989 AT&T
6 .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
7 .\" http://www.opengroup.org/bookstore/.
8 .\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
9 .\" This notice shall appear on any product containing this material.
10 .\" The contents of this file are subject to the terms of the Common Development
11 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
12 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
13 .TH SYSCONF 3C "Mar 26, 2008"
14 .SH NAME
15 sysconf \- get configurable system variables
16 .SH SYNOPSIS
17 .LP
18 .nf
19 #include <unistd.h>

21 \fBlong\fR \fBsysconf\fR(\fBint\fR \fBname\fR);
22 .fi

24 .SH DESCRIPTION
25 .sp
26 .LP
27 The \fBsysconf()\fR function provides a method for an application to determine
28 the current value of a configurable system limit or option (variable).
29 .sp
30 .LP
31 The \fBname\fR argument represents the system variable to be queried. The
32 following table lists the minimal set of system variables from \fB<limits.h>\fR
33 and \fB<unistd.h>\fR that can be returned by \fBsysconf()\fR and the symbolic
34 constants defined in \fB<unistd.h>\fR that are the corresponding values used
35 for \fBname\fR on the SPARC and x86 platforms.
36 .sp
37 .in +2
38 .nf
39      Name                      Return Value          Meaning
40 _____
41 _SC_2_C_BIND                  _POSIX2_C_BIND        Supports the C lang-
42                               _POSIX2_C_DEV         uage binding option
43                               _POSIX2_C_DEV         Supports the C lang-
44                               _POSIX2_C_DEV         uage development
45                               _POSIX2_C_DEV         utilities option
46 _SC_2_C_VERSION                _POSIX2_C_VERSION     Integer value
47                               _POSIX2_C_VERSION     indicates version
48                               _POSIX2_C_VERSION     of ISO POSIX-2
49                               _POSIX2_C_VERSION     standard (Commands)
50 _SC_2_CHAR_TERM                _POSIX2_CHAR_TERM     Supports at least
51                               _POSIX2_CHAR_TERM     one terminal
52 _SC_2_FORT_DEV                 _POSIX2_FORT_DEV      Supports FORTRAN
53                               _POSIX2_FORT_DEV      Development
54                               _POSIX2_FORT_DEV      Utilities Option
55 _SC_2_FORT_RUN                 _POSIX2_FORT_RUN      Supports FORTRAN
56                               _POSIX2_FORT_RUN      Run-time Utilities
57                               _POSIX2_FORT_RUN      Option
58 _SC_2_LOCALEDEF                _POSIX2_LOCALEDEF     Supports creation
59                               _POSIX2_LOCALEDEF     of locales by the
60                               _POSIX2_LOCALEDEF     localedef utility

```

```

61 _SC_2_SW_DEV                   _POSIX2_SW_DEV        Supports Software
62                               _POSIX2_SW_DEV        Development Utility
63                               _POSIX2_SW_DEV        Option
64 _SC_2_UPE                      _POSIX2_UPE           Supports User
65                               _POSIX2_UPE           Portability
66                               _POSIX2_UPE           Utilities Option
67 _SC_2_VERSION                  _POSIX2_VERSION       Integer value
68                               _POSIX2_VERSION       indicates version
69                               _POSIX2_VERSION       of ISO POSIX-2
70                               _POSIX2_VERSION       standard (C language
71                               _POSIX2_VERSION       binding)
72 _SC_AIO_LISTIO_MAX             AIO_LISTIO_MAX        Max number of I/O
73                               AIO_LISTIO_MAX        operations in a
74                               AIO_LISTIO_MAX        single list I/O call
75                               AIO_LISTIO_MAX        supported
76 _SC_AIO_MAX                    AIO_MAX               Max number of
77                               AIO_MAX               outstanding
78                               AIO_MAX               asynchronous I/O
79                               AIO_MAX               operations supported
80 _SC_AIO_Prio_DELTA_MAX         AIO_Prio_DELTA_MAX    Max amount by which
81                               AIO_Prio_DELTA_MAX    process can decrease
82                               AIO_Prio_DELTA_MAX    its asynchronous
83                               AIO_Prio_DELTA_MAX    I/O priority level
84                               AIO_Prio_DELTA_MAX    from its own
85                               AIO_Prio_DELTA_MAX    scheduling priority
86 _SC_ARG_MAX                    ARG_MAX               Max size of argv[]
87                               ARG_MAX               plus envp[]
88 _SC_ASYNCHRONOUS_IO           _POSIX2_ASYNCHRONOUS Supports
89                               _POSIX2_ASYNCHRONOUS Asynchronous I/O
90 _SC_ATEXIT_MAX                 ATEXIT_MAX           Max number of
91                               ATEXIT_MAX           functions that can
92                               ATEXIT_MAX           be registered with
93                               ATEXIT_MAX           atexit()
94 _SC_AVPHYS_PAGES               _POSIX2_AVPHYS_PAGES Number of physical
95                               _POSIX2_AVPHYS_PAGES memory pages not
96                               _POSIX2_AVPHYS_PAGES currently in use by
97                               _POSIX2_AVPHYS_PAGES system
98 _SC_BARRIERS                   _POSIX2_BARRIERS     Supports Barriers
99                               _POSIX2_BARRIERS     option
100 _SC_BC_BASE_MAX                BC_BASE_MAX           Maximum obase values
101                               BC_BASE_MAX           allowed by bc
102 _SC_BC_DIM_MAX                 BC_DIM_MAX            Max number of
103                               BC_DIM_MAX            elements permitted
104                               BC_DIM_MAX            in array by bc
105 _SC_BC_SCALE_MAX               BC_SCALE_MAX          Max scale value
106                               BC_SCALE_MAX          allowed by bc
107 _SC_BC_STRING_MAX              BC_STRING_MAX         Max length of string
108                               BC_STRING_MAX         constant allowed by
109                               BC_STRING_MAX         bc
110 _SC_CHILD_MAX                  CHILD_MAX             Max processes
111                               CHILD_MAX             allowed to a UID
112 _SC_CLK_TCK                    CLK_TCK              Ticks per second
113                               CLK_TCK              (clock_t)
114 _SC_CLOCK_SELECTION            _POSIX2_CLOCK_SELECTION Supports Clock
115                               _POSIX2_CLOCK_SELECTION Selection option
116 _SC_COLL_WEIGHTS_MAX           COLL_WEIGHTS_MAX      Max number of
117                               COLL_WEIGHTS_MAX      weights that can be
118                               COLL_WEIGHTS_MAX      assigned to entry of
119                               COLL_WEIGHTS_MAX      the LC_COLLATE order
120                               COLL_WEIGHTS_MAX      keyword in locale
121                               COLL_WEIGHTS_MAX      definition file
122 _SC_CPUID_MAX                  CPUID_MAX            Max possible
123                               CPUID_MAX            processor ID
124 _SC_DELAYTIMER_MAX             DELAYTIMER_MAX        Max number of timer
125                               DELAYTIMER_MAX        expiration overruns
126 _SC_EXPR_NEST_MAX              EXPR_NEST_MAX        Max number of

```

127			parentheses by expr
128	_SC_FSYNC	_POSIX_FSYNC	Supports File Synchronization
129			Max size of group entry buffer
130	_SC_GETGR_R_SIZE_MAX		Max size of password entry buffer
131			Maximum length of a host name (excluding terminating null)
132	_SC_GETPW_R_SIZE_MAX		Max number of iovec structures available to one process for use with readv() and writev()
133			Job control supported?
134	_SC_HOST_NAME_MAX	_POSIX_HOST_NAME_MAX	Max length of input line
135			Max length of login name
136	_SC_LOGIN_NAME_MAX	LOGIN_NAME_MAX	Max length of login name
137	_SC_LOGIN_NAME_MAX	LOGIN_NAME_MAX	Max length of login name
138			Supports Memory Mapped Files
139	_SC_JOB_CONTROL	_POSIX_JOB_CONTROL	Max pid value
140			Supports Process Memory Locking
141	_SC_LINE_MAX	LINE_MAX	Supports Range Memory Locking
142			Supports Memory Protection
143	_SC_LOGNAME_MAX	LOGNAME_MAX	Supports Message Passing
144	_SC_MAPPED_FILES	_POSIX_MAPPED_FILES	Supports Monotonic Clock option
145			Max number of open message queues a process can hold
146	_SC_MAXPID		Max number of message priorities supported
147	_SC_MEMLOCK	_POSIX_MEMLOCK	Max simultaneous groups to which one can belong
148			Number of processors configured
149	_SC_MEMLOCK_RANGE	_POSIX_MEMLOCK_RANGE	Max number of processors supported by platform
150			Number of processors online
151	_SC_MEMORY_PROTECTION	_POSIX_MEMORY_PROTECTION	Max open files per process
152	_SC_MESSAGE_PASSING	_POSIX_MESSAGE_PASSING	System memory page size
153			Same as _SC_PAGESIZE
154	_SC_MONOTONIC_CLOCK	_POSIX_MONOTONIC_CLOCK	Max number of significant bytes in a password
155			Total number of pages of physical memory in system
156	_SC_MQ_OPEN_MAX	MQ_OPEN_MAX	Supports Prioritized I/O
157			Supports Process Scheduling
158	_SC_MQ_PRIO_MAX	MQ_PRIO_MAX	
159			
160	_SC_NGROUPS_MAX	NGROUPS_MAX	
161			
162	_SC_NPROCESSORS_CONF		
163			
164	_SC_NPROCESSORS_MAX		
165			
166	_SC_NPROCESSORS_ONLN		
167			
168	_SC_OPEN_MAX	OPEN_MAX	
169			
170	_SC_PAGESIZE	PAGESIZE	
171			
172	_SC_PAGE_SIZE	PAGESIZE	
173			
174	_SC_PASS_MAX	PASS_MAX	
175			
176	_SC_PHYS_PAGES		
177			
178	_SC_PRIORITY_SCHEDULING	_POSIX_PRIORITY_SCHEDULING	
179			
180			
181			
182			
183			
184			
185			
186			
187			
188			
189			
190			
191			

192			Scheduling
193	_SC_RAW_SOCKETS	_POSIX_RAW_SOCKETS	Supports Raw Sockets option
194			Max number of repeated occurrences of a regular expression permitted when using interval notation $\{m,n\}$
195	_SC_RE_DUP_MAX	RE_DUP_MAX	Supports IPV6 option
196			Supports Realtime Signals
197			Supports Regular Expression Handling option
198			Max number of realtime signals reserved for application use
199			Supported?
200			Max number of POSIX semaphores a process can have
201	_SC_READER_WRITER_LOCKS	_POSIX_READER_WRITER_LOCKS	Max value a POSIX semaphore can have
202	_SC_REALTIME_SIGNALS	_POSIX_REALTIME_SIGNALS	Supports Semaphores
203			Supports Shared Memory Objects
204	_SC_REGEXP	_POSIX_REGEXP	Supports POSIX shell
205			Max number of queued signals that a process can send and have pending at receiver(s) at a time
206			Supports Spawn option
207	_SC_RTSIG_MAX	RTSIG_MAX	Supports Spin Locks option
208			Default stack protection
209			Number of streams one process can have open at a time
210			Max number of symbolic links that can be reliably traversed in the resolution of a pathname in the absence of a loop
211	_SC_SAVED_IDS	_POSIX_SAVED_IDS	Supports Synchronized I/O
212			Supports Thread Stack Address Attribute option
213			Number of attempts made to destroy thread-specific data on thread exit
214	_SC_SEM_NSEMS_MAX	SEM_NSEMS_MAX	Max number of data keys per process
215			Supports Priority
216			
217	_SC_SEM_VALUE_MAX	SEM_VALUE_MAX	
218			
219	_SC_SEMAPHORES	_POSIX_SEMAPHORES	
220	_SC_SHARED_MEMORY_	_POSIX_SHARED_MEMORY_	
221	OBJECTS	OBJECTS	
222	_SC_SHELL	_POSIX_SHELL	
223	_SC_SIGQUEUE_MAX	SIGQUEUE_MAX	
224			
225			
226			
227			
228			
229	_SC_SPAWN	_POSIX_SPAWN	
230	_SC_SPIN_LOCKS	_POSIX_SPIN_LOCKS	
231			
232	_SC_STACK_PROT		
233			
234	_SC_STREAM_MAX	STREAM_MAX	
235			
236			
237	_SC_SYMLINK_MAX	_POSIX_SYMLINK_MAX	
238			
239			
240			
241			
242			
243	_SC_SYNCHRONIZED_IO	_POSIX_SYNCHRONIZED_IO	
244			
245	_SC_THREAD_ATTR_	_POSIX_THREAD_ATTR_	
246	STACKADDR	STACKADDR	
247			
248	_SC_THREAD_ATTR_	_POSIX_THREAD_ATTR_	
249	STACKSIZE	STACKSIZE	
250			
251	_SC_THREAD_DESTRUCTOR_	PTHREAD_DESTRUCTOR_	
252	ITERATIONS	ITERATIONS	
253			
254			
255	_SC_THREAD_KEYS_MAX	PTHREAD_KEYS_MAX	
256			
257	_SC_THREAD_PRIO_	_POSIX_THREAD_PRIO_	

```

258 INHERIT INHERIT Inheritance option
259 _SC_THREAD_PRIO_ _POSIX_THREAD_PRIO_ Supports Priority
260 PROTECT PROTECT Protection option
261 _SC_THREAD_PRIORITY_ _POSIX_THREAD_PRIORITY_ Supports Thread
262 SCHEDULING SCHEDULING Execution
263 Scheduling option
264 _SC_THREAD_PROCESS_ _POSIX_THREAD_PROCESS_ Supports
265 SHARED SHARED Process-Shared
266 Synchronization
267 option
268 _SC_THREAD_SAFE_ _POSIX_THREAD_SAFE_ Supports Thread-Safe
269 FUNCTIONS FUNCTIONS Functions option
270 _SC_THREAD_STACK_MIN PTHREAD_STACK_MIN Min byte size of
271 thread stack storage
272 _SC_THREAD_THREADS_MAX PTHREAD_THREADS_MAX Max number of
273 threads per process
274 _SC_THREADS _POSIX_THREADS Supports Threads
275 option
276 _SC_TIMEOITS _POSIX_TIMEOITS Supports Timeouts
277 option
278 _SC_TIMER_MAX TIMER_MAX Max number of timer
279 per process
280 supported
281 _SC_TIMERS _POSIX_TIMERS Supports Timers
282 _SC_TTY_NAME_MAX TTYNAME_MAX Max length of tty
283 device name
284 _SC_TZNAME_MAX TZNAME_MAX Max number of bytes
285 supported for name
286 of a time zone
287 _SC_V6_ILP32_OFF32 _POSIX_V6_ILP32_OFF32 Supports X/Open
288 ILP32 w/32-bit
289 offset build
290 environment
291 _SC_V6_ILP32_OFFBIG _POSIX_V6_ILP32_OFFBIG Supports X/Open
292 ILP32 w/64-bit
293 offset build
294 environment
295 _SC_V6_LP64_OFF64 _POSIX_V6_LP64_OFF64 Supports X/Open
296 LP64 w/64-bit
297 offset build
298 environment
299 _SC_V6_LPBIG_OFFBIG _POSIX_V6_LPBIG_OFFBIG Same as
300 _SC_V6_LP64_OFF64
301 _SC_VERSION _POSIX_VERSION POSIX.1 version
302 supported
303 _SC_XBS5_ILP32_OFF32 _XBS_ILP32_OFF32 Indicates support
304 for X/Open ILP32
305 w/32-bit offset
306 build environment
307 _SC_XBS5_ILP32_OFFBIG _XBS5_ILP32_OFFBIG Indicates support
308 for X/Open ILP32
309 w/64-bit offset
310 build environment
311 _SC_XBS5_LP64_OFF64 _XBS5_LP64_OFF64 Indicates support of
312 X/Open LP64,
313 64-bit offset
314 build environment
315 _SC_XBS5_LPBIG_OFFBIG _XBS5_LP64_OFF64 Same as
316 _SC_XBS5_LP64_OFF64
317 _SC_XOPEN_CRYPT _XOPEN_CRYPT Supports X/Open
318 Encryption Feature
319 Group
320 _SC_XOPEN_ENH_I18N _XOPEN_ENH_I18N Supports X/Open
321 Enhanced
322 Internationalization
323 Feature Group

```

```

324 _SC_XOPEN_LEGACY _XOPEN_LEGACY Supports X/Open
325 Legacy Feature Group
326 _SC_XOPEN_REALTIME _XOPEN_REALTIME Supports X/Open
327 POSIX Realtime
328 Feature Group
329 _SC_XOPEN_REALTIME_ _XOPEN_REALTIME_THREADS Supports X/Open
330 THREADS POSIX Realtime
331 Threads Feature
332 Group
333 _SC_XOPEN_SHM _XOPEN_SHM Supports X/Open
334 Shared Memory
335 Feature Group
336 _SC_XOPEN_STREAMS _POSIX_XOPEN_STREAMS Supports XSI Streams
337 option group
338 _SC_XOPEN_UNIX _XOPEN_UNIX Supports X/Open CAE
339 Specification,
340 August 1994, System
341 Interfaces and
342 Headers, Issue 4,
343 Version 2
344 _SC_XOPEN_VERSION _XOPEN_VERSION Integer value
345 indicates version of
346 X/Open Portability
347 Guide to which
348 implementation
349 conforms
350 _SC_XOPEN_XCU_VERSION _XOPEN_XCU_VERSION Integer value
351 indicates version of
352 XCU specification to
353 which implementation
354 conforms
355 .fi
356 .in -2
357 .sp
358
359 .sp
360 .LP
361 The following options are not supported and return \(\mil:
362 .sp
363
364 .sp
365 .TS
366 l l
367 l l .
368 \fb_SC_2_PBS\fr \fb_POSIX2_PBS\fr
369 \fb_SC_2_PBS_ACCOUNTING\fr \fb_POSIX2_PBS_ACCOUNTING\fr
370 \fb_SC_2_PBS_CHECKPOINT\fr \fb_POSIX2_PBS_CHECKPOINT\fr
371 \fb_SC_2_PBS_LOCATE\fr \fb_POSIX2_PBS_LOCATE\fr
372 \fb_SC_2_PBS_MESSAGE\fr \fb_POSIX2_PBS_MESSAGE\fr
373 \fb_SC_2_PBS_TRACK\fr \fb_POSIX2_PBS_TRACK\fr
374 \fb_SC_ADVISORY_INFO\fr \fb_POSIX_ADVISORY_INFO\fr
375 \fb_SC_CPUTIME\fr \fb_POSIX_CPUTIME\fr
376 \fb_SC_SPORADIC_SERVER\fr \fb_POSIX_SPORADIC_SERVER\fr
377 \fb_SC_SS_REPL_MAX\fr \fb_POSIX_SS_REPL_MAX\fr
378 \fb_SC_THREAD_CPUTIME\fr \fb_POSIX_THREAD_CPUTIME\fr
379 \fb_SC_THREAD_SPORADIC_SERVER\fr \fb_POSIX_THREAD_SPORADIC_SERVER\fr
380 \fb_SC_TRACE\fr \fb_POSIX_TRACE\fr
381 \fb_SC_TRACE_EVENT_FILTER\fr \fb_POSIX_TRACE_EVENT_FILTER\fr
382 \fb_SC_TRACE_EVENT_NAME_MAX\fr \fb_POSIX_TRACE_EVENT_NAME_MAX\fr
383 \fb_SC_TRACE_INHERIT\fr \fb_POSIX_TRACE_INHERIT\fr
384 \fb_SC_TRACE_LOG\fr \fb_POSIX_TRACE_LOG\fr
385 \fb_SC_TRACE_NAME_MAX\fr \fb_POSIX_TRACE_NAME_MAX\fr
386 \fb_SC_TRACE_SYS_MAX\fr \fb_POSIX_TRACE_SYS_MAX\fr
387 \fb_SC_TRACE_USER_EVENT_MAX\fr \fb_POSIX_TRACE_USER_EVENT_MAX\fr
388 \fb_SC_TYPED_MEMORY_OBJECTS\fr \fb_POSIX_TYPED_MEMORY_OBJECTS\fr
389 .TE

```

```

391 .SH RETURN VALUES
392 .sp
393 .LP
394 Upon successful completion, \fBsysconf()\fR returns the current variable value
395 on the system. The value returned will not be more restrictive than the
396 corresponding value described to the application when it was compiled with the
397 implementation's <\fBlimits.h\fR>, <\fBunistd.h\fR> or <\fBtime.h\fR>. With
398 only a few obvious exceptions such as \fB_SC_AVPHYS_PAGES\fR and
399 \fB_SC_NPROCESSORS_ONLN\fR, the value will not change during the lifetime of
400 the calling process.
401 .sp
402 .LP
403 If \fIname\fR is an invalid value, \fBsysconf()\fR returns \fB(mil)\fR and sets
404 \fBerrno\fR to indicate the error. If the variable corresponding to \fIname\fR
405 is associated with functionality that is not supported by the system,
406 \fBsysconf()\fR returns \fB(mil)\fR without changing the value of \fBerrno\fR.
407 .sp
408 .LP
409 Calling \fBsysconf()\fR with the following returns \fB(mil)\fR without setting
410 \fBerrno\fR, because no maximum limit can be determined. The system supports at
411 least the minimum values and can support higher values depending upon system
412 resources.
413 .sp
414 .in +2
415 .nf
416 Variable                Minimum supported value
417 _SC_AIO_MAX              _POSIX_AIO_MAX
418 _SC_ATEXIT_MAX          32
419 _SC_MQ_OPEN_MAX         32
420 _SC_THREAD_THREADS_MAX  _POSIX_THREAD_THREADS_MAX
421 _SC_THREAD_KEYS_MAX     _POSIX_THREAD_KEYS_MAX
422 _SC_THREAD_DESTRUCTOR_ITERATIONS  _POSIX_THREAD_DESTRUCTOR_ITERATIONS
423 .fi
424 .in -2

426 .sp
427 .LP
428 The following SPARC and x86 platform variables return \fBEINVAL\fR:
429 .sp
430 .in +2
431 .nf
432 _SC_COHER_BLKSZ         _SC_DCACHE_ASSOC
433 _SC_DCACHE_BLKSZ       _SC_DCACHE_LINESZ
434 _SC_DCACHE_SZ           _SC_DCACHE_TBLKSZ
435 _SC_ICACHE_ASSOC       _SC_ICACHE_BLKSZ
436 _SC_ICACHE_LINESZ     _SC_ICACHE_SZ
437 _SC_SPLIT_CACHE
438 .fi
439 .in -2

441 .SH ERRORS
442 .sp
443 .LP
444 The \fBsysconf()\fR function will fail if:
445 .sp
446 .ne 2
447 .na
448 \fBEINVAL\fR
449 .ad
450 .RS 10n
451 The value of the \fIname\fR argument is invalid.
452 .RE

454 .SH ATTRIBUTES
455 .sp

```

```

456 .LP
457 See \fBattributes\fR(5) for descriptions of the following attributes:
458 .sp

460 .sp
461 .TS
462 box;
463 c | c
464 l | l .
465 ATTRIBUTE TYPE    ATTRIBUTE VALUE
466 -
467 Architecture      SPARC and x86
468 -
469 Interface Stability    Committed
470 -
471 MT-Level           MT-Safe, Async-Signal-Safe
472 -
473 Standard          See \fBstandards\fR(5).
474 .TE

476 .SH SEE ALSO
477 .sp
478 .LP
479 \fBpooladm\fR(1M), \fBzoneadm\fR(1M), \fBfpathconf\fR(2), \fBseteuid\fR(2),
480 \fBsetrlimit\fR(2), \fBconfstr\fR(3C), \fBattributes\fR(5), \fBstandards\fR(5)
481 .SH NOTES
482 .sp
483 .LP
484 A call to \fBsetrlimit()\fR can cause the value of \fBOPEN_MAX\fR to change.
485 .sp
486 .LP
487 Multiplying \fBsysconf\fR(\fB_SC_PHYS_PAGES\fR) or
488 \fBsysconf\fR(\fB_SC_AVPHYS_PAGES\fR) by \fBsysconf\fR(\fB_SC_PAGESIZE\fR) to
489 determine memory amount in bytes can exceed the maximum values representable in
490 a 32-bit signed or unsigned integer.
491 .sp
492 .LP
493 The value of \fBCLK_TCK\fR can be variable and it should not be assumed that
494 \fBCLK_TCK\fR is a compile-time constant.
495 .sp
496 .LP
497 If the caller is in a non-global zone and the pools facility is active,
498 \fBsysconf\fR(\fB_SC_NPROCESSORS_CONF\fR) and
499 \fBsysconf\fR(\fB_SC_NPROCESSORS_ONLN\fR) return the number of processors in
500 the processor set of the pool to which the zone is bound.

```

```

*****
9883 Wed Apr 3 09:33:13 2013
new/usr/src/man/man4/passwd.4
2989 Eliminate use of LOGNAME_MAX in ON
1166 useradd have warning with name more 8 chars
*****
1 \" te
2 .\" Copyright (c) 2013 Gary Mills
3 .\" Copyright (c) 2004, 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 PASSWD 4 \"Jul 28, 2004\"
9 .SH NAME
10 passwd \- password file
11 .SH SYNOPSIS
12 .LP
13 .nf
14 \fB/etc/passwd\fR
15 .fi

17 .SH DESCRIPTION
18 .sp
19 .LP
20 The file \fB/etc/passwd\fR is a local source of information about users'
21 accounts. The password file can be used in conjunction with other naming
22 sources, such as the \fBNIS\fR maps \fBpasswd.byname\fR and \fBpasswd.bygid\fR,
23 data from the \fBNIS+\fR \fBpasswd\fR table, or password data stored on an LDAP
24 server. Programs use the \fBgetpwnam\fR(3C) routines to access this
25 information.
26 .sp
27 .LP
28 Each \fBpasswd\fR entry is a single line of the form:
29 .sp
30 .in +2
31 .nf
32 \fIusername\fR:\fB:\fR:\fIpassword\fR:\fB:\fR:\fIuid\fR:\fB:\fR
33 \fIgid\fR:\fB:\fR:\fIgcosp-field\fR:\fB:\fR:\fIhome-dir\fR:\fB:\fR
34 \fIlogin-shell\fR
35 .fi
36 .in -2
37 .sp

39 .sp
40 .LP
41 where
42 .sp
43 .ne 2
44 .na
45 \fB\fIusername\fR\fR
46 .ad
47 .RS 15n
48 is the user's login name.
49 .sp
50 The login (\fBlogin\fR) and role (\fBrole\fR) fields accept a string of no more
51 than 32 bytes consisting of characters from the set of alphabetic
52 than eight bytes consisting of characters from the set of alphabetic
53 characters, numeric characters, period (\fB.&\fR), underscore (\fB_\fR), and
54 hyphen (\fB-\fR). The first character should be alphabetic and the field should
55 contain at least one lower case alphabetic character. A warning message is
56 displayed if these restrictions are not met.
57 .sp
57 The \fBlogin\fR and \fBrole\fR fields must contain at least one character and
58 must not contain a colon (\fB:\fR) or a newline (\fB\n\fR).
59 .RE

```

```

61 .sp
62 .ne 2
63 .na
64 \fB\fIpassword\fR\fR
65 .ad
66 .RS 15n
67 is an empty field. The encrypted password for the user is in the corresponding
68 entry in the \fB/etc/shadow\fR file. \fBpwconv\fR(1M) relies on a special value
69 of '\fBx\fR' in the password field of \fB/etc/passwd\fR. If this value
70 of '\fBx\fR' exists in the password field of \fB/etc/passwd\fR, this indicates
71 that the password for the user is already in \fB/etc/shadow\fR and should not
72 be modified.
73 .RE

75 .sp
76 .ne 2
77 .na
78 \fB\fIuid\fR\fR
79 .ad
80 .RS 15n
81 is the user's unique numerical \fBID\fR for the system.
82 .RE

84 .sp
85 .ne 2
86 .na
87 \fB\fIgid\fR\fR
88 .ad
89 .RS 15n
90 is the unique numerical \fBID\fR of the group that the user belongs to.
91 .RE

93 .sp
94 .ne 2
95 .na
96 \fB\fIgcosp-field\fR\fR
97 .ad
98 .RS 15n
99 is the user's real name, along with information to pass along in a mail-message
100 heading. (It is called the gcosp-field for historical reasons.) An '\fB&\fR'
101 (ampersand) in this field stands for the login name (in cases where the login
102 name appears in a user's real name).
103 .RE

105 .sp
106 .ne 2
107 .na
108 \fB\fIhome-dir\fR\fR
109 .ad
110 .RS 15n
111 is the pathname to the directory in which the user is initially positioned upon
112 logging in.
113 .RE

115 .sp
116 .ne 2
117 .na
118 \fB\fIlogin-shell\fR\fR
119 .ad
120 .RS 15n
121 is the user's initial shell program. If this field is empty, the default shell
122 is \fB/usr/bin/sh\fR.
123 .RE

125 .sp

```


126 .LP
 127 The maximum value of the `\fluid` and `\igid` fields is `\B2147483647`. To
 128 maximize interoperability and compatibility, administrators are recommended to
 129 assign users a range of `\BUID`s and `\BGID`s below `\B60000` where
 130 possible. (`\BUID`s from `\B0` to `\B99` inclusive are reserved by the
 131 operating system vendor for use in future applications. Their use by end system
 132 users or vendors of layered products is not supported and may cause security
 133 related issues with future applications.)
 134 .sp
 135 .LP
 136 The password file is an `\BASCII` file that resides in the `\B/etc`
 137 directory. Because the encrypted passwords on a secure system are always kept
 138 in the `\Bshadow` file, `\B/etc/passwd` has general read permission on all
 139 systems and can be used by routines that map between numerical user `\BID`s
 140 and user names.
 141 .sp
 142 .LP
 143 Blank lines are treated as malformed entries in the `\Bpasswd` file and cause
 144 consumers of the file, such as `\Bgetpwnam(3C)`, to fail.
 145 .sp
 146 .LP
 147 The password file can contain entries beginning with a '+' (plus sign) or '-'
 148 (minus sign) to selectively incorporate entries from another naming service
 149 source, such as NIS, NIS+, or LDAP.
 150 .sp
 151 .LP
 152 A line beginning with a '+' means to incorporate entries from the naming
 153 service source. There are three styles of the '+' entries in this file. A
 154 single + means to insert all the entries from the alternate naming service
 155 source at that point, while a `+filename` means to insert the specific entry,
 156 if one exists, from the naming service source. A `+@fInetgroup` means to
 157 insert the entries for all members of the network group `\fInetgroup` from the
 158 alternate naming service. If a `+filename` entry has a non-null `\Bpassword`,
 159 `\fIgcoss`, `\fIhome-dir`, or `\fIlogin-shell` field, the value of that field
 160 overrides what is contained in the alternate naming service. The `\fluid` and
 161 `\igid` fields cannot be overridden.
 162 .sp
 163 .LP
 164 A line beginning with a '\(mi' means to disallow entries from the alternate
 165 naming service. There are two styles of '-' entries in this file. `-filename`
 166 means to disallow any subsequent entries (if any) for `\fIname` (in this file
 167 or in a naming service), and `-@fInetgroup` means to disallow any subsequent
 168 entries for all members of the network group `\fInetgroup`.
 169 .sp
 170 .LP
 171 This is also supported by specifying `'passwd : compat'` in
 172 `\Bnsswitch.conf(4)`. The "compat" source might not be supported in future
 173 releases. The preferred sources are `\Bfiles` followed by the identifier of a
 174 name service, such as `\Bnis` or `\Bldap`. This has the effect of
 175 incorporating the entire contents of the naming service's `\Bpasswd` database
 176 or password-related information after the `\Bpasswd` file.
 177 .sp
 178 .LP
 179 Note that in compat mode, for every `\B/etc/passwd` entry, there must be a
 180 corresponding entry in the `\B/etc/shadow` file.
 181 .sp
 182 .LP
 183 Appropriate precautions must be taken to lock the `\B/etc/passwd` file
 184 against simultaneous changes if it is to be edited with a text editor;
 185 `\Bvipw(1B)` does the necessary locking.
 186 .SH EXAMPLES
 187 .LP
 188 `\BExample 1 \fRSample \fBpasswd` File
 189 .sp
 190 .LP
 191 The following is a sample `\Bpasswd` file:

193 .sp
 194 .in +2
 195 .nf
 196 root:x:0:1:Super-User:/:/sbin/sh
 197 fred:6k/7KCFRPNVXg:508:10:& Fredericks:/usr2/fred:/bin/csh
 198 .fi
 199 .in -2
 200 .sp
 202 .sp
 203 .LP
 204 and the sample password entry from `\Bnsswitch.conf` is:
 206 .sp
 207 .in +2
 208 .nf
 209 passwd: files ldap
 210 .fi
 211 .in -2
 212 .sp
 214 .sp
 215 .LP
 216 In this example, there are specific entries for users `\Broot` and `\Bfred`
 217 to assure that they can login even when the system is running single-user. In
 218 addition, anyone whose password information is stored on an LDAP server will be
 219 able to login with their usual password, shell, and home directory.
 221 .sp
 222 .LP
 223 If the password file is:
 225 .sp
 226 .in +2
 227 .nf
 228 root:x:0:1:Super-User:/:/sbin/sh
 229 fred:6k/7KCFRPNVXg:508:10:& Fredericks:/usr2/fred:/bin/csh
 230 +
 231 .fi
 232 .in -2
 233 .sp
 235 .sp
 236 .LP
 237 and the password entry in `\Bnsswitch.conf` is:
 239 .sp
 240 .in +2
 241 .nf
 242 passwd: compat
 243 .fi
 244 .in -2
 245 .sp
 247 .sp
 248 .LP
 249 then all the entries listed in the `\BNIS` `\Bpasswd.byuid` and
 250 `\Bpasswd.byname` maps will be effectively incorporated after the entries for
 251 `\Broot` and `\Bfred`. If the password entry in `\Bnsswitch.conf` is:
 253 .sp
 254 .in +2
 255 .nf
 256 passwd_compat: ldap
 257 passwd: compat

```

258 .fi
259 .in -2

261 .sp
262 .LP
263 then all password-related entries stored on the LDAP server will be
264 incorporated after the entries for \fBroot\fR and \fBfred\fR.

266 .sp
267 .LP
268 The following is a sample \fBpasswd\fR file when \fBshadow\fR does not exist:

270 .sp
271 .in +2
272 .nf
273 root:q.mJzTnu8icf.:0:1:Super-User:/:/sbin/sh
274 fred:6k/7KCFRPNVXg:508:10:& Fredericks:/usr2/fred:/bin/csh
275 +john:
276 +@documentation:no-login:
277 +:::Guest
278 .fi
279 .in -2
280 .sp

282 .sp
283 .LP
284 The following is a sample \fBpasswd\fR file when \fBshadow\fR does exist:

286 .sp
287 .in +2
288 .nf
289 root:##root:0:1:Super-User:/:/sbin/sh
290 fred:##fred:508:10:& Fredericks:/usr2/fred:/bin/csh
291 +john:
292 +@documentation:no-login:
293 +:::Guest
294 .fi
295 .in -2
296 .sp

298 .sp
299 .LP
300 In this example, there are specific entries for users \fBroot\fR and
301 \fBfred\fR, to assure that they can log in even when the system is running
302 standalone. The user \fBjohn\fR will have his password entry in the naming
303 service source incorporated without change, anyone in the netgroup
304 \fBdocumentation\fR will have their password field disabled, and anyone else
305 will be able to log in with their usual password, shell, and home directory,
306 but with a \fBgcscos\fR field of \fBGuest\fR

308 .SH FILES
309 .sp
310 .ne 2
311 .na
312 \fB/etc/nsswitch.conf\fR
313 .ad
314 .RS 22n

316 .RE

318 .sp
319 .ne 2
320 .na
321 \fB/etc/passwd\fR
322 .ad
323 .RS 22n

```

```

325 .RE

327 .sp
328 .ne 2
329 .na
330 \fB/etc/shadow\fR
331 .ad
332 .RS 22n

334 .RE

336 .SH SEE ALSO
337 .sp
338 .LP
339 \fBchgrp\fR(1), \fBchown\fR(1), \fBfinger\fR(1), \fBgroups\fR(1),
340 \fBlogin\fR(1), \fBnewgrp\fR(1), \fBnispasswd\fR(1), \fBpasswd\fR(1),
341 \fBsh\fR(1), \fBsort\fR(1), \fBdomainname\fR(1M), \fBgetent\fR(1M),
342 \fBin.ftpd\fR(1M), \fBpassmgmt\fR(1M), \fBpwck\fR(1M), \fBpwconv\fR(1M),
343 \fBsu\fR(1M), \fBuseradd\fR(1M), \fBuserdel\fR(1M), \fBusermod\fR(1M),
344 \fBa64l\fR(3C), \fBcrypt\fR(3C), \fBgetpw\fR(3C), \fBgetpwnam\fR(3C),
345 \fBgetspnam\fR(3C), \fBputpwent\fR(3C), \fBgroup\fR(4), \fBhosts.equiv\fR(4),
346 \fBnsswitch.conf\fR(4), \fBshadow\fR(4), \fBenviron\fR(5),
347 \fBunistd.h\fR(3HEAD)
348 .sp
349 .LP
350 \fISystem Administration Guide: Basic Administration\fR

```