```
**********************************************************
   20302 Thu Jun 13 01:09:55 2013
new/usr/src/cmd/sulogin/sulogin.c
3808 sulogin should reset console to text mode
Reviewed by: Jason King <jason.brian.king@gmail.com>
Reviewed by: Garrett D'Amore <garrett@damore.org>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T
 *      All rights reserved.
 *
 *      Copyright (c) 1987, 1988 Microsoft Corporation.
 *      All rights reserved.
 */

/*
 *      sulogin - special login program exec'd from init to let user
 *      come up single user, or go to default init state straight away.
 *
 *      Explain the scoop to the user, prompt for an authorized user
 *      name or ^D and then prompt for password or ^D.  If the password
 *      is correct, check if the user is authorized, if so enter
 *      single user. ^D exits sulogin, and init will go to default init state.
 *
 *      If /etc/passwd is missing, or there's no entry for root,
 *      go single user, no questions asked.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/param.h>
#include <sys/sysmsg_impl.h>
#include <sys/mkdev.h>
#include <sys/resource.h>
#include <sys/uadmin.h>
#include <sys/wait.h>
```



```
#include <sys/stermio.h>
#include <fcntl.h>
#include <termio.h>
#include <pwd.h>
#include <shadow.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <siginfo.h>
#include <utmpx.h>
#include <unistd.h>
#include <ucontext.h>
#include <string.h>
#include <strings.h>
#include <deflt.h>
#include <limits.h>
#include <errno.h>
#include <crypt.h>
#include <auth_attr.h>
#include <auth_list.h>
#include <nss_dbdefs.h>
#include <user_attr.h>
#include <sys/vt.h>
#include <sys/kd.h>

/*
 * Intervals to sleep after failed login
 */
#ifndef SLEEPTIME
#define SLEEPTIME       4       /* sleeptime before login incorrect msg */
#endif

#define SLEEPTIME_MAX   5       /* maximum sleeptime */

/*
 *      the name of the file containing the login defaults we deliberately
 *      use the same file as login(1)
 */

#define DEFAULT_LOGIN   "/etc/default/login"
#define DEFAULT_SULOGIN "/etc/default/sulogin"
#define DEFAULT_CONSOLE "/dev/console"

static char     shell[] = "/sbin/sh";
static char     su[]    = "/sbin/su.static";
static int      sleeptime       = SLEEPTIME;
static int      nchild = 0;
static pid_t    pidlist[10];
static pid_t    masterpid = 0;
static pid_t    originalpid = 0;
static struct sigaction sa;
static struct termio    ttymodes;

static char     *findttyname(int fd);
static char     *stripttyname(char *);
static char     *sulogin_getinput(char *, int);
static void     noop(int);
static void     single(const char *, char *);
static void     main_loop(char *, boolean_t);
static void     parenthandler();
static void     termhandler(int);
static void     setupsigs(void);
static int      pathcmp(char *, char *);
static void     doit(char *, char *);
static void     childcleanup(int);
```

```
126 #define ECHOON  0
127 #define ECHOOFF 1

129 /* ARGSUSED */
130 int
131 main(int argc, char **argv)
132 {
133         struct spwd     *shpw;
134         int             passreq = B_TRUE;
135         int             flags;
136         int             fd;
137         char            *infop, *ptr, *p;
138         pid_t           pid;
139         int             bufsize;
140         struct stat     st;
141         char            cttyname[100];
142         char            namedlist[500];
143         char            scratchlist[500];
144         dev_t           cttyd;

146         if (geteuid() != 0) {
147                 (void) fprintf(stderr, "%s: must be root\n", argv[0]);
148                 return (EXIT_FAILURE);
149         }

151         /* Do the magic to determine the children */
152         if ((fd = open(SYSMSG, 0)) < 0)
153                 return (EXIT_FAILURE);

155         /*
156          * If the console supports the CIOCTTYCONSOLE ioctl, then fetch
157          * its console device list.  If not, then we use the default
158          * console name.
159          */
160         if (ioctl(fd, CIOCTTYCONSOLE, &cttyd) == 0) {
161                 if ((bufsize = ioctl(fd, CIOCGETCONSOLE, NULL)) < 0)
162                         return (EXIT_FAILURE);

164                 if (bufsize > 0) {
165                         if ((infop = calloc(bufsize, sizeof (char))) == NULL)
166                                 return (EXIT_FAILURE);

168                         if (ioctl(fd, CIOCGETCONSOLE, infop) < 0)
169                                 return (EXIT_FAILURE);

171                         (void) snprintf(namedlist, sizeof (namedlist), "%s %s",
172                             DEFAULT_CONSOLE, infop);
173                 } else
174                         (void) snprintf(namedlist, sizeof (namedlist), "%s",
175                             DEFAULT_CONSOLE);
176         } else {
177                 (void) snprintf(namedlist, sizeof (namedlist), "%s",
178                     DEFAULT_CONSOLE);
179                 cttyd = NODEV;
180         }

182         /*
183          * The attempt to turn the controlling terminals dev_t into a string
184          * may not be successful, thus leaving the variable cttyname as a
185          * NULL.  This occurs if during boot we find
186          * the root partition (or some other partition)
187          * requires manual fsck, thus resulting in sulogin
188          * getting invoked.  The ioctl for CIOCTTYCONSOLE
189          * called above returned NODEV for cttyd
190          * in these cases.  NODEV gets returned when the vnode pointer
191          * in our session structure is NULL.  In these cases it
```



```
192          * must be assumed that the default console is used.
193          *
194          * See uts/common/os/session.c:cttydev().
195          */
196         (void) strcpy(cttyname, DEFAULT_CONSOLE);
197         (void) strcpy(scratchlist, namedlist);
198         ptr = scratchlist;
199         while (ptr != NULL) {
200                 p = strchr(ptr, ' ');
201                 if (p == NULL) {
202                         if (stat(ptr, &st))
203                                 return (EXIT_FAILURE);
204                         if (st.st_rdev == cttyd)
205                                 (void) strcpy(cttyname, ptr);
206                         break;
207                 }
208                 *p++ = '\0';
209                 if (stat(ptr, &st))
210                         return (EXIT_FAILURE);
211                 if (st.st_rdev == cttyd) {
212                         (void) strcpy(cttyname, ptr);
213                         break;
214                 }
215                 ptr = p;
216         }

218         /*
219          * Use the same value of SLEEPTIME that login(1) uses.  This
220          * is obtained by reading the file /etc/default/login using
221          * the def*() functions.
222          */

224         if (defopen(DEFAULT_LOGIN) == 0) {

226                 /* ignore case */

228                 flags = defcntl(DC_GETFLAGS, 0);
229                 TURNOFF(flags, DC_CASE);
230                 (void) defcntl(DC_SETFLAGS, flags);

232                 if ((ptr = defread("SLEEPTIME=")) != NULL)
233                         sleeptime = atoi(ptr);

235                 if (sleeptime < 0 || sleeptime > SLEEPTIME_MAX)
236                         sleeptime = SLEEPTIME;

238                 (void) defopen(NULL);   /* closes DEFAULT_LOGIN */
239         }

241         /*
242          * Use our own value of PASSREQ, separate from the one login(1) uses.
243          * This is obtained by reading the file /etc/default/sulogin using
244          * the def*() functions.
245          */

247         if (defopen(DEFAULT_SULOGIN) == 0) {
248                 if ((ptr = defread("PASSREQ=")) != NULL)
249                         if (strcmp("NO", ptr) == 0)
250                                 passreq = B_FALSE;

252                 (void) defopen(NULL);   /* closes DEFAULT_SULOGIN */
253         }

255         if (passreq == B_FALSE)
256                 single(shell, NULL);
```

```
258         /*
259          * if no 'root' entry in /etc/shadow, give maint. mode single
260          * user shell prompt
261          */
262         setspent();
263         if ((shpw = getspnam("root")) == NULL) {
264                 (void) fprintf(stderr, "\n*** Unable to retrieve 'root' entry "
265                     "in shadow password file ***\n\n");
266                 single(shell, NULL);
267         }
268         endspent();
269         /*
270          * if no 'root' entry in /etc/passwd, give maint. mode single
271          * user shell prompt
272          */
273         setpwent();
274         if (getpwnam("root") == NULL) {
275                 (void) fprintf(stderr, "\n*** Unable to retrieve 'root' entry "
276                     "in password file ***\n\n");
277                 single(shell, NULL);
278         }
279         endpwent();
280         /* process with controlling tty treated special */
281         if ((pid = fork()) != (pid_t)0) {
282                 if (pid == -1)
283                         return (EXIT_FAILURE);
284                 else {
285                         setupsigs();
286                         masterpid = pid;
287                         originalpid = getpid();
288                         /*
289                          * init() was invoked from a console that was not
290                          * the default console, nor was it an auxiliary.
291                          */
292                         if (cttyname[0] == NULL)
293                                 termhandler(0);
294                                 /* Never returns */

296                         main_loop(cttyname, B_TRUE);
297                         /* Never returns */
298                 }
299         }
300         masterpid = getpid();
301         originalpid = getppid();
302         pidlist[nchild++] = originalpid;

304         sa.sa_handler = childcleanup;
305         sa.sa_flags = 0;
306         (void) sigemptyset(&sa.sa_mask);
307         (void) sigaction(SIGTERM, &sa, NULL);
308         (void) sigaction(SIGHUP, &sa, NULL);
309         sa.sa_handler = parenthandler;
310         sa.sa_flags = SA_SIGINFO;
311         (void) sigemptyset(&sa.sa_mask);
312         (void) sigaction(SIGUSR1, &sa, NULL);

314         sa.sa_handler = SIG_IGN;
315         sa.sa_flags = 0;
316         (void) sigemptyset(&sa.sa_mask);
317         (void) sigaction(SIGCHLD, &sa, NULL);
318         /*
319          * If there isn't a password on root, then don't permit
320          * the fanout capability of sulogin.
321          */
322         if (*shpw->sp_pwdp != '\0') {
323                 ptr = namedlist;
```



```
324                 while (ptr != NULL) {
325                         p = strchr(ptr, ' ');
326                         if (p == NULL) {
327                                 doit(ptr, cttyname);
328                                 break;
329                         }
330                         *p++ = '\0';
331                         doit(ptr, cttyname);
332                         ptr = p;
333                 }
334         }
335         if (pathcmp(cttyname, DEFAULT_CONSOLE) != 0) {
336                 if ((pid = fork()) == (pid_t)0) {
337                         setupsigs();
338                         main_loop(DEFAULT_CONSOLE, B_FALSE);
339                 } else if (pid == -1)
340                         return (EXIT_FAILURE);
341                 pidlist[nchild++] = pid;
342         }
343         /*
344          * When parent is all done, it pauses until one of its children
345          * signals that its time to kill the underpriviledged.
346          */
347         (void) wait(NULL);

349         return (0);
350 }
```

*_____unchanged_portion_omitted_*

```
448 static void
449 main_loop(char *devname, boolean_t cttyflag)
450 {
451         int             fd, fb, i;
445         int             fd, i;
452         char            *user = NULL;           /* authorized user */
453         char            *pass;                  /* password from user */
454         char            *cpass;                 /* crypted password */
455         struct spwd     spwd;
456         struct spwd     *lshpw;                 /* local shadow */
457         char            shadow[NSS_BUFLEN_SHADOW];
458         FILE            *sysmsgfd;

460         for (i = 0; i < 3; i++)
461                 (void) close(i);
462         if (cttyflag == B_FALSE) {
463                 if (setsid() == -1)
464                         exit(EXIT_FAILURE);
465         }
466         if ((fd = open(devname, O_RDWR)) < 0)
467                 exit(EXIT_FAILURE);

469         /*
470          * In system maintenance mode, all virtual console instances
471          * of the svc:/system/console-login service are not available
472          * any more, and only the system console is available. So here
473          * we always switch to the system console in case at the moment
474          * the active console isn't it.
475          */
476         (void) ioctl(fd, VT_ACTIVATE, 1);

478         if (fd != 0)
479                 (void) dup2(fd, STDIN_FILENO);
480         if (fd != 1)
481                 (void) dup2(fd, STDOUT_FILENO);
482         if (fd != 2)
483                 (void) dup2(fd, STDERR_FILENO);
```

```
484         if (fd > 2)
485                 (void) close(fd);
487         /* Stop progress bar and reset console mode to text */
488         if ((fb = open("/dev/fb", O_RDONLY)) >= 0) {
489                 (void) ioctl(fb, KDSETMODE, KD_RESETTEXT);
490                 (void) close(fb);
491         }
493         sysmsgfd = fopen("/dev/sysmsg", "w");
495         sanitize_tty(fileno(stdin));
497         for (;;) {
498                 do {
499                         (void) printf("\nEnter user name for system "
500                             "maintenance (control-d to bypass): ");
501                         user = sulogin_getinput(devname, ECHOON);
502                         if (user == NULL) {
503                                 /* signal other children to exit */
504                                 (void) sigsend(P_PID, masterpid, SIGUSR1);
505                                 /* ^D, so straight to default init state */
506                                 exit(EXIT_FAILURE);
507                         }
508                 } while (user[0] == '\0');
509                 (void) printf("Enter %s password (control-d to bypass): ",
510                     user);
512                 if ((pass = sulogin_getinput(devname, ECHOOFF)) == NULL) {
513                         /* signal other children to exit */
514                         (void) sigsend(P_PID, masterpid, SIGUSR1);
515                         /* ^D, so straight to default init state */
516                         free(user);
517                         exit(EXIT_FAILURE);
518                 }
519                 lshpw = getspnam_r(user, &spwd, shadow, sizeof (shadow));
520                 if (lshpw == NULL) {
521                         /*
522                          * the user entered doesn't exist, too bad.
523                          */
524                         goto sorry;
525                 }
527                 /*
528                  * There is a special case error to catch here:
529                  * If the password is hashed with an algorithm
530                  * other than the old unix crypt the call to crypt(3c)
531                  * could fail if /usr is corrupt or not available
532                  * since by default /etc/security/crypt.conf will
533                  * have the crypt_ modules located under /usr/lib.
534                  * Or it could happen if /etc/security/crypt.conf
535                  * is corrupted.
536                  *
537                  * If this happens crypt(3c) will return NULL and
538                  * set errno to ELIBACC for the former condition or
539                  * EINVAL for the latter, in this case we bypass
540                  * authentication and just verify that the user is
541                  * authorized.
542                  */
544                 errno = 0;
545                 cpass = crypt(pass, lshpw->sp_pwdp);
546                 if (((cpass == NULL) && (lshpw->sp_pwdp[0] == '$')) &&
547                     ((errno == ELIBACC) || (errno == EINVAL))) {
548                         goto checkauth;
549                 } else if ((cpass == NULL) ||
```



```
550                     (strcmp(cpass, lshpw->sp_pwdp) != 0)) {
551                         goto sorry;
552                 }
554 checkauth:
555                 /*
556                  * There is a special case error here as well.
557                  * If /etc/user_attr is corrupt, getusernam("root")
558                  * returns NULL.
559                  * In this case, we just give access because this is similar
560                  * to the case of root not existing in /etc/passwd.
561                  */
563                 if ((getusernam("root") != NULL) &&
564                     (chkauthattr(MAINTENANCE_AUTH, user) != 1)) {
565                         goto sorry;
566                 }
567                 (void) fprintf(sysmsgfd, "\nsingle-user privilege "
568                     "assigned to %s on %s.\n", user, devname);
569                 (void) sigsend(P_PID, masterpid, SIGUSR1);
570                 (void) wait(NULL);
571                 free(user);
572                 free(pass);
573                 single(su, devname);
574                 /* single never returns */
576 sorry:
577                 (void) printf("\nLogin incorrect or user %s not authorized\n",
578                     user);
579                 free(user);
580                 free(pass);
581                 (void) sleep(sleeptime);
582         }
583 }
```

*_____unchanged_portion_omitted_*