```
**********************************************************
    5939 Fri Jun 22 11:09:02 2012
new/usr/src/cmd/cron/Makefile
374 cron should send more useful mail
**********************************************************
```

```
#
# 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 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

DEFAULTFILES =  cron.dfl

include         ../Makefile.cmd

MANIFEST =      cron.xml

ROOTMANIFESTDIR = $(ROOTSVCSYSTEM)
ROOTMETHOD =    $(ROOTLIBSVCMETHOD)/svc-cron

CPPFLAGS +=     -D_FILE_OFFSET_BITS=64 -I $(SRC)/common/util

ROOTVAR =       $(ROOT)/var

ROOTSPCRON =    $(ROOTVAR)/spool/cron
ROOTCROND =     $(ROOTETC)/cron.d
ROOTCRONTABS =  $(ROOTSPCRON)/crontabs
ROOTATJOBS =    $(ROOTSPCRON)/atjobs
ROOTLIBCRON =   $(ROOTLIB)/cron

PROG1 =         cron
PROG2 =         at atq atrm crontab
XPG6PROG =      crontab
XPG4PROG =      at crontab
PROG =          $(PROG1) $(PROG2)

SCRIPT =        batch
XPG4SCRIPT =    batch.xpg4

POFILE=         $(PROG1)_cmd.po
POFILES1=       at.po crontab.po funcs.po batch.po
POFILES=        $(POFILES1) atrm.po
$(POFILES1) :=  XGETFLAGS= -a -x $(PROG1).xcl

ROOTDIRS =      $(ROOTSPCRON) $(ROOTCROND) \
                $(ROOTCRONTABS) $(ROOTATJOBS)
```



```
ROOTPROG =      $(PROG1:%=$(ROOTUSRSBIN)/%) $(PROG2:%=$(ROOTBIN)/%)     \
                $(SCRIPT:%=$(ROOTBIN)/%) \
                $(XPG6PROG:%=$(ROOTXPG6BIN)/%) \
                $(XPG4PROG:%=$(ROOTXPG4BIN)/%) \
                $(XPG4SCRIPT:%.xpg4=$(ROOTXPG4BIN)/%)

ROOTSYMLINK =   $(ROOTLIBCRON) $(ROOTETC)/cron

GETRESPSRC=     $(SRC)/common/util/getresponse.c
GETRESPOBJ=     getresponse.o
COMMONOBJ1=     permit.o
COMMONOBJ2=     funcs.o
COMMONOBJS=     $(COMMONOBJ1) $(COMMONOBJ2)
CRONOBJS=       cron.o elm.o cron_scf.o
CRONOBJS=       cron.o elm.o
ATOBJS=         at.o att1.o att2.o
XPG4OBJS=       values-xpg4.o
ATRMOBJS1=      atrm.o
ATRMOBJS=       $(ATRMOBJS1) $(GETRESPOBJ)
ATQOBJS=        atq.o
CRONTABOBJS1=   crontab.o
CRONTABOBJS=    $(CRONTABOBJS1) $(GETRESPOBJ)

# /usr/xpg*/bin/crontab isn't linked with values-xpg*.o since it isn't
# required by any specific behavior differences; this makes these
# setuid variants less likely to accidently trip over differences that
# could unintentionally open up a security hole.
XPG4COMMONOBJS= $(COMMONOBJS:%=objs.xpg4/%)
XPG4CTOBJS=     $(CRONTABOBJS:%=objs.xpg4/%)
XPG4ATOBJS=     $(ATOBJS:%=objs.xpg4/%) $(XPG4OBJS:%=objs.xpg4/%)
XPG6COMMONOBJS= $(COMMONOBJS:%=objs.xpg6/%)
XPG6CTOBJS=     $(CRONTABOBJS:%=objs.xpg6/%)

cron :=         POBJS = $(CRONOBJS) $(COMMONOBJ2)
at :=           POBJS = $(ATOBJS) $(COMMONOBJS)
at.xpg4 :=      POBJS = $(XPG4ATOBJS) $(XPG4COMMONOBJS)
atrm :=         POBJS = $(ATRMOBJS) $(COMMONOBJS)
atq :=          POBJS = $(ATQOBJS) $(COMMONOBJS)
crontab :=      POBJS = $(CRONTABOBJS) $(COMMONOBJS)
crontab.xpg4 := POBJS = $(XPG4CTOBJS) $(XPG4COMMONOBJS)
crontab.xpg6 := POBJS = $(XPG6CTOBJS) $(XPG6COMMONOBJS)

CFLAGS += $(CCVERBOSE)

NOBJS=          $(CRONOBJS) $(ATOBJS) $(ATRMOBJS1) $(ATQOBJS) $(CRONTABOBJS1) \
                $(COMMONOBJS)
OBJS =          $(NOBJS) $(XPG4COMMONOBJS) $(XPG4ATOBJS) $(XPG4CTOBJS) \
                $(XPG6COMMONOBJS) $(XPG6CTOBJS) $(GETRESPOBJ)

SRCS =          $(NOBJS:%.o=%.c) $(GETRESPSRC)

CLOBBERFILES += $(SCRIPT) $(XPG4SCRIPT)

$(ROOTLIBCRON) :=               SYMLNKDEST = ../../etc/cron.d
$(ROOTETC)/cron :=              SYMLNKDEST = ../usr/sbin/cron

$(ROOTBIN)/at :=                FILEMODE = 04755
$(ROOTXPG4BIN)/at :=            FILEMODE = 04755
$(ROOTBIN)/atrm :=              FILEMODE = 04755
$(ROOTBIN)/atq :=               FILEMODE = 04755
$(ROOTBIN)/crontab :=           FILEMODE = 04555
$(ROOTXPG6BIN)/crontab :=       FILEMODE = 04555
$(ROOTXPG4BIN)/crontab :=       FILEMODE = 04555
$(ROOTUSRSBIN)/cron :=          FILEMODE = 0555

LDLIBS +=       -lbsm
```

```
128 at :=           LDLIBS += -lproject -lsecdb
129 at.xpg4 :=      LDLIBS += -lproject -lsecdb
130 atq :=          LDLIBS += -lsecdb
131 atrm :=         LDLIBS += -lsecdb
132 cron :=         LDLIBS += -lpam -lproject -lcontract -lzoneinfo -lscf
132 cron :=         LDLIBS += -lpam -lproject -lcontract -lzoneinfo
133 crontab :=     LDLIBS += -lsecdb -lpam -lzoneinfo
134 crontab.xpg6 := LDLIBS += -lsecdb -lpam -lzoneinfo
135 crontab.xpg4 := LDLIBS += -lsecdb -lpam -lzoneinfo

137 lint :=         LDLIBS += -lproject -lsecdb -lcontract -lpam

139 $(XPG4) := CFLAGS += -DXPG4
140 $(XPG6) := CFLAGS += -DXPG6

142 LINTFLAGS += -u

144 $(ROOTSVCSYSTEM)/cron.xml := FILEMODE = 0444
145 $(ROOTLIBSVCMETHOD)/svc-cron := FILEMODE = 0555


148 .KEEP_STATE:

150 all :           $(PROG) $(XPG4) $(XPG6) $(SCRIPT) $(XPG4SCRIPT) $(FILES)

152 install :       all $(ROOTPROG) $(ROOTETCDEFAULTFILES) $(ROOTSYMLINK) \
153                 $(ROOTMANIFEST) $(ROOTMETHOD)

155 $(PROG) :       $$(POBJS)
156                 $(LINK.c) $(POBJS) -o $@ $(LDLIBS)
157                 $(POST_PROCESS)

159 $(XPG4) :       objs.xpg4 $$(POBJS)
160                 $(LINK.c) $(POBJS) -o $@ $(LDLIBS)
161                 $(POST_PROCESS)

163 $(XPG6) :       objs.xpg6 $$(POBJS)
164                 $(LINK.c) $(POBJS) -o $@ $(LDLIBS)
165                 $(POST_PROCESS)

167 objs.xpg6/%.o:          %.c
168         $(COMPILE.c) -o $@ $<

170 objs.xpg6:
171         -@mkdir -p $@

173 objs.xpg4/%.o:          %.c
174         $(COMPILE.c) -o $@ $<

176 objs.xpg4:
177         -@mkdir -p $@

179 objs.xpg4/values-xpg4.o:        ../../lib/common/common/values-xpg4.c
180         $(COMPILE.c) -o $@ ../../lib/common/common/values-xpg4.c

182 %.o:                    $(SRC)/common/util/%.c
183         $(COMPILE.c) $(OUTPUT_OPTION) $<
184         $(POST_PROCESS_O)

186 objs.xpg4/%.o:  $(SRC)/common/util/%.c
187         $(COMPILE.c) -o $@ $<
188         $(POST_PROCESS_O)

190 objs.xpg6/%.o:  $(SRC)/common/util/%.c
191         $(COMPILE.c) -o $@ $<
```



```
192         $(POST_PROCESS_O)

194 att1.c :        att1.y
195                 $(YACC.y) -d att1.y
196                 $(MV) y.tab.c att1.c
197                 $(MV) y.tab.h att1.h

199 att2.c :        att2.l att2.ed att1.c
200                 $(LEX) att2.l
201                 ed - lex.yy.c < att2.ed
202                 $(MV) lex.yy.c att2.c

204 # Don't re-install directories installed by Targetdirs
205 #$(ROOTDIRS):
206 #               $(INS.dir)

208 $(ROOTSYMLINK) :
209                 $(RM) $@; $(SYMLINK) $(SYMLNKDEST) $@

211 check:          $(CHKMANIFEST)

213 $(POFILE):      $(POFILES)
214         $(RM) $@; cat $(POFILES) > $@

216 clean :
217                 $(RM) $(OBJS) att1.h att1.c att2.c

219 lint :          lint_SRCS

221 strip :
222                 $(STRIP) $(PROG) $(XPG4) $(XPG6)

224 include         ../Makefile.targ
```

```
**********************************************************
   87182 Fri Jun 22 11:09:03 2012
new/usr/src/cmd/cron/cron.c
374 cron should send more useful mail
**********************************************************
```

```
   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 2009 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  */

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

  29 /*      Copyright (c) 1987, 1988 Microsoft Corporation  */
  30 /*        All Rights Reserved   */

  32 #ifdef lint
  33 /* make lint happy */
  34 #define __EXTENSIONS__
  35 #endif

  37 #include <sys/contract/process.h>
  38 #include <sys/ctfs.h>
  39 #include <sys/param.h>
  40 #include <sys/resource.h>
  41 #include <sys/stat.h>
  42 #include <sys/task.h>
  43 #include <sys/time.h>
  44 #include <sys/types.h>
  45 #include <sys/utsname.h>
  46 #include <sys/wait.h>

  48 #include <security/pam_appl.h>

  50 #include <alloca.h>
  51 #include <ctype.h>
  52 #include <deflt.h>
  53 #include <dirent.h>
  54 #include <errno.h>
  55 #include <fcntl.h>
  56 #include <grp.h>
  57 #include <libcontract.h>
  58 #include <libcontract_priv.h>
  59 #include <limits.h>
  60 #include <locale.h>
  61 #include <poll.h>
```



```
  62 #include <project.h>
  63 #include <pwd.h>
  64 #include <signal.h>
  65 #include <stdarg.h>
  66 #include <stdio.h>
  67 #include <stdlib.h>
  68 #include <string.h>
  69 #include <stropts.h>
  70 #include <time.h>
  71 #include <unistd.h>
  72 #include <libzoneinfo.h>

  74 #include "cron.h"
  75 #include "cron_scf.h"

  77 /*
  78  * #define      DEBUG
  79  */

  81 #define MAIL            "/usr/bin/mail" /* mail program to use */
  82 #define CONSOLE         "/dev/console"  /* where messages go when cron dies */

  84 #define TMPINFILE       "/tmp/crinXXXXXX"  /* file to put stdin in for cmd  */
  85 #define TMPDIR          "/tmp"
  86 #define PFX             "crout"
  87 #define TMPOUTFILE      "/tmp/croutXXXXXX" /* file to place stdout, stderr */

  89 #define INMODE          00400           /* mode for stdin file  */
  90 #define OUTMODE         00600           /* mode for stdout file */
  91 #define ISUID           S_ISUID         /* mode for verifing at jobs */

  93 #define INFINITY        2147483647L     /* upper bound on time  */
  94 #define CUSHION         180L
  95 #define ZOMB            100             /* proc slot used for mailing output */

  97 #define JOBF            'j'
  98 #define NICEF           'n'
  99 #define USERF           'u'
 100 #define WAITF           'w'

 102 #define BCHAR           '>'
 103 #define ECHAR           '<'

 105 #define DEFAULT         0
 106 #define LOAD            1
 107 #define QBUFSIZ         80

 109 /* Defined actions for crabort() routine */
 110 #define NO_ACTION       000
 111 #define REMOVE_FIFO     001
 112 #define CONSOLE_MSG     002

 114 #define BADCD           "can't change directory to the crontab directory."
 115 #define NOREADDIR       "can't read the crontab directory."

 117 #define BADJOBOPEN      "unable to read your at job."
 118 #define BADSHELL        "because your login shell \
 119 isn't /usr/bin/sh, you can't use cron."

 121 #define BADSTAT         "can't access your crontab or at-job file. Resubmit it."
 122 #define BADPROJID       "can't set project id for your job."
 123 #define CANTCDHOME      "can't change directory to %s.\
 124 \nYour commands will not be executed."
 125 #define CANTEXECSH      "unable to exec the shell, %s, for one of your \
 126 commands."
 127 #define CANT_STR_LEN (sizeof (CANTEXECSH) > sizeof (CANTCDHOME) ? \
```

```
 128         sizeof (CANTEXECSH) : sizeof (CANTCDHOME))
 129 #define NOREAD          "can't read your crontab file.  Resubmit it."
 130 #define BADTYPE         "crontab or at-job file is not a regular file.\n"
 131 #define NOSTDIN         "unable to create a standard input file for \
 132 one of your crontab commands. \
 133 \nThat command was not executed."

 135 #define NOTALLOWED      "you are not authorized to use cron.  Sorry."
 136 #define STDERRMSG       "\n\n****************************************\
 137 *****\nCron: The previous message is the \
 138 standard output and standard error \
 139 \nof one of your cron commands.\n"

 141 #define STDOUTERR       "one of your commands generated output or errors, \
 142 but cron was unable to mail you this output.\
 143 \nRemember to redirect standard output and standard \
 144 error for each of your commands."

 146 #define CLOCK_DRIFT     "clock time drifted backwards after event!\n"
 147 #define PIDERR          "unexpected pid returned %d (ignored)"
 148 #define CRONTABERR      "Subject: Your crontab file has an error in it\n\n"
 149 #define CRONOUT         "Subject: Output from \"cron\" command\n\n"
 150 #define CRONOUTNEW      "Subject: Cron <%s@%s>: %s\n\n"
 151 #define MALLOCERR       "out of space, cannot create new string\n"

 153 #define DIDFORK didfork
 154 #define NOFORK !didfork

 156 #define MAILBUFLEN      (8*1024)
 157 #define LINELIMIT       80
 158 #define MAILBINITFREE   (MAILBUFLEN - (sizeof (cte_intro) - 1) \
 159             - (sizeof (cte_trail1) - 1) - (sizeof (cte_trail2) - 1) - 1)

 161 #define ERR_CRONTABENT  0       /* error in crontab file entry */
 162 #define ERR_UNIXERR     1       /* error in some system call */
 163 #define ERR_CANTEXECCRON 2      /* error setting up "cron" job environment */
 164 #define ERR_CANTEXECAT  3       /* error setting up "at" job environment */
 165 #define ERR_NOTREG      4       /* error not a regular file */

 167 #define PROJECT         "project="

 169 #define MAX_LOST_CONTRACTS      2048    /* reset if this many failed abandons */

 171 #define FORMAT  "%a %b %e %H:%M:%S %Y"
 172 static char     timebuf[80];

 174 static struct message msgbuf;

 176 struct shared {
 177         int count;                      /* usage count */
 178         void (*free)(void *obj);        /* routine that will free obj */
 179         void *obj;                      /* object */
 180 };
_____unchanged_portion_omitted_

 287 extern char **environ;

 289 #define DEFTZ           "GMT"
 290 static  int     log = 0;
 291 static  char    hzname[10];

 293 static void cronend(int);
 294 static void thaw_handler(int);
 295 static void child_handler(int);
 296 static void child_sigreset(void);
```



```
 298 static void mod_ctab(char *, time_t);
 299 static void mod_atjob(char *, time_t);
 300 static void add_atevent(struct usr *, char *, time_t, int);
 301 static void rm_ctevents(struct usr *);
 302 static void cleanup(struct runinfo *rn, int r);
 303 static void crabort(char *, int);
 304 static void msg(char *fmt, ...);
 305 static void ignore_msg(char *, char *, struct event *);
 306 static void logit(int, struct runinfo *, int);
 307 static void parsqdef(char *);
 308 static void defaults();
 309 static void initialize(int);
 310 static void quedefs(int);
 311 static int idle(long);
 312 static struct usr *find_usr(char *);
 313 static int ex(struct event *e);
 314 static void read_dirs(int);
 315 static void mail(char *, char *, int);
 316 static char *next_field(int, int);
 317 static void readcron(struct usr *, time_t);
 318 static int next_ge(int, char *);
 319 static void free_if_unused(struct usr *);
 320 static void del_atjob(char *, char *);
 321 static void del_ctab(char *);
 322 static void resched(int);
 323 static int msg_wait(long);
 324 static struct runinfo *rinfo_get(pid_t);
 325 static void rinfo_free(struct runinfo *rp);
 326 static void mail_result(struct usr *p, struct runinfo *pr, size_t filesize);
 327 static time_t next_time(struct event *, time_t);
 328 static time_t get_switching_time(int, time_t);
 329 static time_t xmktime(struct tm *);
 330 static void process_msg(struct message *, time_t);
 331 static void reap_child(void);
 332 static void miscpid_insert(pid_t);
 333 static int miscpid_delete(pid_t);
 334 static void contract_set_template(void);
 335 static void contract_clear_template(void);
 336 static void contract_abandon_latest(pid_t);

 338 static void cte_init(void);
 339 static void cte_add(int, char *);
 340 static void cte_valid(void);
 341 static int cte_istoomany(void);
 342 static void cte_sendmail(char *);

 344 static int set_user_cred(const struct usr *, struct project *);

 346 static struct shared *create_shared_str(char *str);
 347 static struct shared *dup_shared(struct shared *obj);
 348 static void rel_shared(struct shared *obj);
 349 static void *get_obj(struct shared *obj);
 350 /*
 351  * last_time is set immediately prior to exection of an event (via ex())
 352  * to indicate the last time an event was executed.  This was (surely)
 353  * it's original intended use.
 354  */
 355 static time_t last_time, init_time, t_old;
 356 static int reset_needed; /* set to 1 when cron(1M) needs to re-initialize */

 358 static int              refresh;
 359 static sigset_t         defmask, sigmask;

 361 /*
 362  * Configuration from smf(5)
 363  */
```

```
364 static int legacy_subject;
365 static int extra_headers;

367 /*
368  * BSM hooks
369  */
370 extern int      audit_cron_session(char *, char *, uid_t, gid_t, char *);
371 extern void     audit_cron_new_job(char *, int, void *);
372 extern void     audit_cron_bad_user(char *);
373 extern void     audit_cron_user_acct_expired(char *);
374 extern int      audit_cron_create_anc_file(char *, char *, char *, uid_t);
375 extern int      audit_cron_delete_anc_file(char *, char *);
376 extern int      audit_cron_is_anc_name(char *);
377 extern int      audit_cron_mode();

379 static int cron_conv(int, struct pam_message **,
380                 struct pam_response **, void *);

382 static struct pam_conv pam_conv = {cron_conv, NULL};
383 static pam_handle_t *pamh;      /* Authentication handle */

385 /*
386  * Function to help check a user's credentials.
387  */

389 static int verify_user_cred(struct usr *u);

391 /*
392  * Values returned by verify_user_cred and set_user_cred:
393  */

395 #define VUC_OK          0
396 #define VUC_BADUSER     1
397 #define VUC_NOTINGROUP  2
398 #define VUC_EXPIRED     3
399 #define VUC_NEW_AUTH    4

401 /*
402  * Modes of process_anc_files function
403  */
404 #define CRON_ANC_DELETE 1
405 #define CRON_ANC_CREATE 0

407 /*
408  * Functions to remove a user or job completely from the running database.
409  */
410 static void clean_out_atjobs(struct usr *u);
411 static void clean_out_ctab(struct usr *u);
412 static void clean_out_user(struct usr *u);
413 static void cron_unlink(char *name);
414 static void process_anc_files(int);

416 /*
417  * functions in elm.c
418  */
419 extern void el_init(int, time_t, time_t, int);
420 extern int el_add(void *, time_t, int);
421 extern void el_remove(int, int);
422 extern int el_empty(void);
423 extern void *el_first(void);
424 extern void el_delete(void);

426 static int valid_entry(char *, int);
427 static struct usr *create_ulist(char *, int);
428 static void init_cronevent(char *, int);
429 static void init_atevent(char *, time_t, int, int);
```



```
430 static void update_atevent(struct usr *, char *, time_t, int);

432 int
433 main(int argc, char *argv[])
434 {
435         time_t t;
436         time_t ne_time;         /* amt of time until next event execution */
437         time_t newtime, lastmtime = 0L;
438         struct usr *u;
439         struct event *e, *e2, *eprev;
440         struct stat buf;
441         pid_t rfork;
442         struct sigaction act;

444         /*
445          * reset_needed is set to 1 whenever el_add() finds out that a cron
446          * job is scheduled to be run before the time when cron(1M) daemon
447          * initialized.
448          * Other cases where a reset is needed is when ex() finds that the
449          * event to be executed is being run at the wrong time, or when idle()
450          * determines that time was reset.
451          * We immediately return to the top of the while (TRUE) loop in
452          * main() where the event list is cleared and rebuilt, and reset_needed
453          * is set back to 0.
454          */
455         reset_needed = 0;

457         /*
458          * Only the privileged user can run this command.
459          */
460         if (getuid() != 0)
461                 crabort(NOTALLOWED, 0);

463 begin:
464         (void) setlocale(LC_ALL, "");
465         /* fork unless 'nofork' is specified */
466         if ((argc <= 1) || (strcmp(argv[1], "nofork"))) {
467                 if (rfork = fork()) {
468                         if (rfork == (pid_t)-1) {
469                                 (void) sleep(30);
470                                 goto begin;
471                         }
472                         return (0);
473                 }
474                 didfork++;
475                 (void) setpgrp();       /* detach cron from console */
476         }

478         (void) umask(022);
479         (void) signal(SIGHUP, SIG_IGN);
480         (void) signal(SIGINT, SIG_IGN);
481         (void) signal(SIGQUIT, SIG_IGN);
482         (void) signal(SIGTERM, cronend);

484         defaults();
485         initialize(1);
486         quedefs(DEFAULT);       /* load default queue definitions */
487         cron_pid = getpid();
488         msg("*** cron started ***   pid = %d", cron_pid);

490         /* setup THAW handler */
491         act.sa_handler = thaw_handler;
492         act.sa_flags = 0;
493         (void) sigemptyset(&act.sa_mask);
494         (void) sigaction(SIGTHAW, &act, NULL);
```

```
496         /* setup CHLD handler */
497         act.sa_handler = child_handler;
498         act.sa_flags = 0;
499         (void) sigemptyset(&act.sa_mask);
500         (void) sigaddset(&act.sa_mask, SIGCLD);
501         (void) sigaction(SIGCLD, &act, NULL);

503         (void) sigemptyset(&defmask);
504         (void) sigemptyset(&sigmask);
505         (void) sigaddset(&sigmask, SIGCLD);
506         (void) sigaddset(&sigmask, SIGTHAW);
507         (void) sigprocmask(SIG_BLOCK, &sigmask, NULL);

509         t_old = init_time;
510         last_time = t_old;
511         for (;;) {              /* MAIN LOOP */
512                 t = time(NULL);
513                 if ((t_old > t) || (t-last_time > CUSHION) || reset_needed) {
514                         reset_needed = 0;
515                         /*
516                          * the time was set backwards or forward or
517                          * refresh is requested.
518                          */
519                         if (refresh)
520                                 msg("re-scheduling jobs");
521                         else
522                                 msg("time was reset, re-initializing");
523                         el_delete();
524                         u = uhead;
525                         while (u != NULL) {
526                                 rm_ctevents(u);
527                                 e = u->atevents;
528                                 while (e != NULL) {
529                                         free(e->cmd);
530                                         e2 = e->link;
531                                         free(e);
532                                         e = e2;
533                                 }
534                                 u->atevents = NULL;
535                                 u = u->nextusr;
536                         }
537                         (void) close(msgfd);
538                         initialize(0);
539                         t = time(NULL);
540                         last_time = t;
541                         /*
542                          * reset_needed might have been set in the functions
543                          * call path from initialize()
544                          */
545                         if (reset_needed) {
546                                 continue;
547                         }
548                 }
549                 t_old = t;

551                 if (next_event == NULL && !el_empty()) {
552                         next_event = (struct event *)el_first();
553                 }
554                 if (next_event == NULL) {
555                         ne_time = INFINITY;
556                 } else {
557                         ne_time = next_event->time - t;
558 #ifdef DEBUG
559                         cftime(timebuf, "%C", &next_event->time);
560                         (void) fprintf(stderr, "next_time=%ld %s\n",
561                             next_event->time, timebuf);
```



```
562 #endif
563                 }
564                 if (ne_time > 0) {
565                         /*
566                          * reset_needed may be set in the functions call path
567                          * from idle()
568                          */
569                         if (idle(ne_time) || reset_needed) {
570                                 reset_needed = 1;
571                                 continue;
572                         }
573                 }

575                 if (stat(QUEDEFS, &buf)) {
576                         msg("cannot stat QUEDEFS file");
577                 } else if (lastmtime != buf.st_mtime) {
578                         quedefs(LOAD);
579                         lastmtime = buf.st_mtime;
580                 }

582                 last_time = next_event->time; /* save execution time */

584                 /*
585                  * reset_needed may be set in the functions call path
586                  * from ex()
587                  */
588                 if (ex(next_event) || reset_needed) {
589                         reset_needed = 1;
590                         continue;
591                 }

593                 switch (next_event->etype) {
594                 case CRONEVENT:
595                         /* add cronevent back into the main event list */
596                         if (delayed) {
597                                 delayed = 0;
598                                 break;
599                         }

601                         /*
602                          * check if time(0)< last_time. if so, then the
603                          * system clock has gone backwards. to prevent this
604                          * job from being started twice, we reschedule this
605                          * job for the >>next time after last_time<<, and
606                          * then set next_event->time to this. note that
607                          * crontab's resolution is 1 minute.
608                          */

610                         if (last_time > time(NULL)) {
611                                 msg(CLOCK_DRIFT);
612                                 /*
613                                  * bump up to next 30 second
614                                  * increment
615                                  * 1 <= newtime <= 30
616                                  */
617                                 newtime = 30 - (last_time % 30);
618                                 newtime += last_time;

620                                 /*
621                                  * get the next scheduled event,
622                                  * not the one that we just
623                                  * kicked off!
624                                  */
625                                 next_event->time =
626                                     next_time(next_event, newtime);
627                                 t_old = time(NULL);
```

```
628                         } else {
629                                 next_event->time =
630                                     next_time(next_event, (time_t)0);
631                         }
632 #ifdef DEBUG
633                         cftime(timebuf, "%C", &next_event->time);
634                         (void) fprintf(stderr,
635                             "pushing back cron event %s at %ld (%s)\n",
636                             next_event->cmd, next_event->time, timebuf);
637 #endif

639                         switch (el_add(next_event, next_event->time,
640                             (next_event->u)->ctid)) {
641                         case -1:
642                                 ignore_msg("main", "cron", next_event);
643                                 break;
644                         case -2: /* event time lower than init time */
645                                 reset_needed = 1;
646                                 break;
647                         }
648                         break;
649                 default:
650                         /* remove at or batch job from system */
651                         if (delayed) {
652                                 delayed = 0;
653                                 break;
654                         }
655                         eprev = NULL;
656                         e = (next_event->u)->atevents;
657                         while (e != NULL) {
658                                 if (e == next_event) {
659                                         if (eprev == NULL)
660                                                 (e->u)->atevents = e->link;
661                                         else
662                                                 eprev->link = e->link;
663                                         free(e->cmd);
664                                         free(e);
665                                         break;
666                                 } else {
667                                         eprev = e;
668                                         e = e->link;
669                                 }
670                         }
671                         break;
672                 }
673                 next_event = NULL;
674         }

676         /*NOTREACHED*/
677 }

679 static void
680 initialize(int firstpass)
681 {
682 #ifdef DEBUG
683         (void) fprintf(stderr, "in initialize\n");
684 #endif
685         if (firstpass) {
686                 int val;
687                 /* for mail(1), make sure messages come from root */
688                 if (putenv("LOGNAME=root") != 0) {
689                         crabort("cannot expand env variable",
690                             REMOVE_FIFO|CONSOLE_MSG);
691                 }
692                 if (access(FIFO, R_OK) == -1) {
693                         if (errno == ENOENT) {
```



```
694                                 if (mknod(FIFO, S_IFIFO|0600, 0) != 0)
695                                         crabort("cannot create fifo queue",
696                                             REMOVE_FIFO|CONSOLE_MSG);
697                         } else {
698                                 if (NOFORK) {
699                                         /* didn't fork... init(1M) is waiting */
700                                         (void) sleep(60);
701                                 }
702                                 perror("FIFO");
703                                 crabort("cannot access fifo queue",
704                                     REMOVE_FIFO|CONSOLE_MSG);
705                         }
706                 } else {
707                         if (NOFORK) {
708                                 /* didn't fork... init(1M) is waiting */
709                                 (void) sleep(60);
710                                 /*
711                                  * the wait is painful, but we don't want
712                                  * init respawning this quickly
713                                  */
714                         }
715                         crabort("cannot start cron; FIFO exists", CONSOLE_MSG);
716                 }
717                 switch (init_scf()) {
718                 case 0:
719                         val = get_config_boolean("legacy_subject");
720                         legacy_subject = (val < 0 ? 0 : val);
721                         val = get_config_boolean("extra_headers");
722                         extra_headers = (val < 0 ? 1 : val);
723                         break;
724                 case -2:
725                         crabort("cron not running under smf(5)",
726                             REMOVE_FIFO|CONSOLE_MSG);
727                         break;
728                 default:
729                         crabort("could not initialise libscf",
730                             REMOVE_FIFO|CONSOLE_MSG);
731                 }
732         }

734         if ((msgfd = open(FIFO, O_RDWR)) < 0) {
735                 perror("! open");
736                 crabort("cannot open fifo queue", REMOVE_FIFO|CONSOLE_MSG);
737         }

739         init_time = time(NULL);
740         el_init(8, init_time, (time_t)(60*60*24), 10);

742         init_time = time(NULL);
743         el_init(8, init_time, (time_t)(60*60*24), 10);

745         /*
746          * read directories, create users list, and add events to the
747          * main event list. Only zero user list on firstpass.
748          */
749         if (firstpass)
750                 uhead = NULL;
751         read_dirs(firstpass);
752         next_event = NULL;

754         if (!firstpass)
755                 return;

757         /* stdout is log file */
758         if (freopen(ACCTFILE, "a", stdout) == NULL)
759                 (void) fprintf(stderr, "cannot open %s\n", ACCTFILE);
```

```
 761         /* log should be root-only */
 762         (void) fchmod(1, S_IRUSR|S_IWUSR);

 764         /* stderr also goes to ACCTFILE */
 765         (void) close(fileno(stderr));
 766         (void) dup(1);
 767         /* null for stdin */
 768         (void) freopen("/dev/null", "r", stdin);

 770         contract_set_template();
 771 }
```

*unchanged_portion_omitted_*

```
2723 /*
2724  * Mail stdout and stderr of a job to user. Get uid for real user and become
2725  * that person. We do this so that mail won't come from root since this
2726  * could be a security hole. If failure, quit - don't send mail as root.
2727  */
2728 static void
2729 mail_result(struct usr *p, struct runinfo *pr, size_t filesize)
2730 {
2731         struct  passwd  *ruser_ids;
2732         FILE    *mailpipe;
2733         FILE    *st;
2734         struct utsname  name;
2735         int     nbytes;
2736         char    iobuf[BUFSIZ];
2737         char    *cmd;

2739         (void) uname(&name);
2740         if ((ruser_ids = getpwnam(p->name)) == NULL)
2741                 exit(0);
2742         (void) setuid(ruser_ids->pw_uid);

2744         cmd = xmalloc(strlen(MAIL) + strlen(p->name)+2);
2745         (void) sprintf(cmd, "%s %s", MAIL, p->name);
2746         mailpipe = popen(cmd, "w");
2747         free(cmd);
2748         if (mailpipe == NULL)
2749                 exit(127);
2750         (void) fprintf(mailpipe, "To: %s\n", p->name);
2751         if (extra_headers) {
2752                 (void) fprintf(mailpipe, "X-Mailer: cron (%s %s)\n",
2753                     name.sysname, name.release);
2754                 (void) fprintf(mailpipe, "X-Cron-User: %s\n", p->name);
2755                 (void) fprintf(mailpipe, "X-Cron-Host: %s\n", name.nodename);
2756                 (void) fprintf(mailpipe, "X-Cron-Job-Name: %s\n", pr->jobname);
2757                 (void) fprintf(mailpipe, "X-Cron-Job-Type: %s\n",
2758                     (pr->jobtype == CRONEVENT ? "cron" : "at"));
2759         }
2760         if (pr->jobtype == CRONEVENT) {
2761                 if (legacy_subject)
2762                         (void) fprintf(mailpipe, CRONOUT);
2763                 else
2764                         (void) fprintf(mailpipe, CRONOUTNEW,
2765                             p->name, name.nodename, pr->jobname);
2766                 (void) fprintf(mailpipe, "Your \"cron\" job on %s\n",
2767                     name.nodename);
2768                 if (pr->jobname != NULL) {
2769                         (void) fprintf(mailpipe, "%s\n\n", pr->jobname);
2770                 }
2771         } else {
2772                 (void) fprintf(mailpipe, "Subject: Output from \"at\" job\n\n");
2773                 (void) fprintf(mailpipe, "Your \"at\" job on %s\n",
2774                     name.nodename);
```



```
2775                 if (pr->jobname != NULL) {
2776                         (void) fprintf(mailpipe, "\"%s\"\n\n", pr->jobname);
2777                 }
2778         }
2779         /* Tmp. file is fopen'ed w/ "r",  secure open */
2780         if (filesize > 0 &&
2781             (st = fopen(pr->outfile, "r")) != NULL) {
2782                 (void) fprintf(mailpipe,
2783                     "produced the following output:\n\n");
2784                 while ((nbytes = fread(iobuf, sizeof (char), BUFSIZ, st)) != 0)
2785                         (void) fwrite(iobuf, sizeof (char), nbytes, mailpipe);
2786                 (void) fclose(st);
2787         } else {
2788                 (void) fprintf(mailpipe, "completed.\n");
2789         }
2790         (void) pclose(mailpipe);
2791         exit(0);
2792 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    3329 Fri Jun 22 11:09:03 2012
new/usr/src/cmd/cron/cron.xml
374 cron should send more useful mail
**********************************************************
```

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<!--
 Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 Use is subject to license terms.

 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

    NOTE:  This service manifest is not editable; its contents will
    be overwritten by package or patch operations, including
    operating system upgrade.  Make customizations in a different
    file.
-->

<service_bundle type='manifest' name='SUNWcsr:cron'>

<service
        name='system/cron'
        type='service'
        version='1'>

        <single_instance />

        <dependency
                name='usr'
                type='service'
                grouping='require_all'
                restart_on='none'>
                <service_fmri value='svc:/system/filesystem/local' />
        </dependency>

        <dependency
                name='ns'
                type='service'
                grouping='require_all'
                restart_on='none'>
                <service_fmri value='svc:/milestone/name-services' />
        </dependency>

        <dependent
                name='cron_multi-user'
                grouping='optional_all'
                restart_on='none'>
                <service_fmri value='svc:/milestone/multi-user' />
        </dependent>

        <exec_method
                type='method'
                name='start'
                exec='/lib/svc/method/svc-cron'
                timeout_seconds='60'>
                <method_context>
                        <method_credential user='root' group='root' />
                </method_context>
        </exec_method>

        <exec_method
                type='method'
                name='stop'
                exec=':kill'
                timeout_seconds='60'>
        </exec_method>

        <exec_method
                type='method'
                name='refresh'
                exec=':kill -THAW'
                timeout_seconds='60'>
        </exec_method>

        <property_group name='startd' type='framework'>
                <!-- sub-process core dumps shouldn't restart session -->
                <propval name='ignore_error' type='astring'
                    value='core,signal' />
        </property_group>

        <property_group name='general' type='framework'>
                <!-- to start stop cron -->
                <propval name='action_authorization' type='astring'
                        value='solaris.smf.manage.cron' />
        </property_group>

        <property_group name='config' type='application'>
                <!-- use legacy (static) subject string in mail? -->
                <propval name='legacy_subject' type='boolean'
                        value='false' />
                <!-- include X-Mailer, X-Cron-* headers in mail? -->
                <propval name='extra_headers' type='boolean'
                        value='true' />
        </property_group>

        <instance name='default' enabled='false' />

        <stability value='Unstable' />

        <template>
                <common_name>
                        <loctext xml:lang='C'>
                        clock daemon (cron)
                        </loctext>
                </common_name>
                <documentation>
                        <manpage title='cron' section='1M' manpath='/usr/share/m
                        <manpage title='crontab' section='1' manpath='/usr/share
                </documentation>
        </template>
</service>

</service_bundle>
```

```
**********************************************************
    3614 Fri Jun 22 11:09:04 2012
new/usr/src/cmd/cron/cron_scf.c
374 cron should send more useful mail
**********************************************************
```

```
/*
 * 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 2012 Joshua M. Clulow <josh@sysmgr.org>
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <err.h>

#include <libscf.h>

#include "cron_scf.h"

extern void *xmalloc(size_t); /* from funcs.c */

static char *fmri;
static scf_handle_t *scf;
static scf_instance_t *inst;
static scf_propertygroup_t *pg;

/*
 * Initialise our connection to smf(5).  Returns:
 *    0 on success
 *   <0 for any failure
 *   -2 if we believe we're not actually running under smf(5)
 */
int
init_scf(void)
{
        int rc = -1;
        size_t fmrisz;

        scf = scf_handle_create(SCF_VERSION);
        if (scf == NULL)
                return (-1);
        if (scf_handle_bind(scf) != 0)
                goto cleanup0;

#ifdef DEBUG
        if (getenv("SMF_TEST_FMRI") != NULL) {
```



```
                printf("DEBUG: smf test mode engaged\n");
                fmri = strdup(getenv("SMF_TEST_FMRI"));
                goto have_fmri;
        }
#endif /* DEBUG */

        fmrisz = scf_myname(scf, NULL, 0);
        if (fmrisz == -1) {
                if (scf_error() == SCF_ERROR_NOT_SET) {
                        rc = -2;
                }
                goto cleanup1;
        }
        fmri = xmalloc(fmrisz + 1);
        fmrisz = scf_myname(scf, fmri, fmrisz + 1);
        if (fmrisz == -1)
                goto cleanup2;

have_fmri:

        if ((inst = scf_instance_create(scf)) == NULL ||
            (pg = scf_pg_create(scf)) == NULL)
                goto cleanup3;

        if (scf_handle_decode_fmri(scf, fmri, NULL, NULL, inst,
            NULL, NULL, SCF_DECODE_FMRI_EXACT) != 0)
                goto cleanup3;

        if (scf_instance_get_pg_composed(inst, NULL, "config", pg) != 0)
                goto cleanup3;

        return (0);

cleanup3:
        if (pg != NULL) {
                scf_pg_destroy(pg);
                pg = NULL;
        }
        if (inst != NULL) {
                scf_instance_destroy(inst);
                inst = NULL;
        }
cleanup2:
        free(fmri);
        fmri = NULL;
cleanup1:
        (void) scf_handle_unbind(scf);
cleanup0:
        (void) scf_handle_destroy(scf);
        scf = NULL;
        return (rc);
}

void
fini_scf(void)
{
        (void) scf_pg_destroy(pg);
        pg = NULL;
        (void) scf_instance_destroy(inst);
        inst = NULL;
        free(fmri);
        fmri = NULL;
        (void) scf_handle_unbind(scf);
        (void) scf_handle_destroy(scf);
        scf = NULL;
}
```

```
129 /*
130  * Fetch the boolean value of a property from the 'config' property
131  * group in our smf(5) instance.
132  * Returns:
133  *   <0 on failure
134  *    0 if found and false
135  *    1 if found and true
136  */
137 int
138 get_config_boolean(char *name)
139 {
140         scf_property_t *prop = NULL;
141         scf_value_t *val = NULL;
142         uint8_t out;
143         int rc = -1;

145         prop = scf_property_create(scf);
146         val = scf_value_create(scf);
147         if (prop == NULL || val == NULL)
148                 goto cleanup0;

150         if (scf_pg_get_property(pg, name, prop) != 0)
151                 goto cleanup0;

153         if (scf_property_is_type(prop, SCF_TYPE_BOOLEAN) != 0)
154                 goto cleanup0;

156         if (scf_property_get_value(prop, val) != 0)
157                 goto cleanup0;

159         if (scf_value_get_boolean(val, &out) != 0)
160                 goto cleanup0;

162         rc = out;

164 cleanup0:
165         if (val != NULL)
166                 scf_value_destroy(val);
167         if (prop != NULL)
168                 scf_property_destroy(prop);

170         return (rc);
171 }
```

```
**********************************************************
    1090 Fri Jun 22 11:09:04 2012
new/usr/src/cmd/cron/cron_scf.h
374 cron should send more useful mail
**********************************************************
```

```
/*
 * 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 2012 Joshua M. Clulow <josh@sysmgr.org>
 */

#ifndef _CRON_SCF_H
#define _CRON_SCF_H

#ifdef  __cplusplus
extern "C" {
#endif

int init_scf(void);
void fini_scf(void);
int get_config_boolean(char *);

#ifdef  __cplusplus
}
#endif

#endif  /* _CRON_SCF_H */
```