```
**********************************************************
   38941 Fri Jul 26 18:54:21 2013
new/usr/src/cmd/halt/halt.c
3913 there is no dialup, only zuul
**********************************************************
```

```
/*
 * 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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 * Portions contributed by Juergen Keil, <jk@tools.de>.
 */


/*
 * Common code for halt(1M), poweroff(1M), and reboot(1M).  We use
 * argv[0] to determine which behavior to exhibit.
 */

#include <stdio.h>
#include <procfs.h>
#include <sys/types.h>
#include <sys/elf.h>
#include <sys/systeminfo.h>
#include <sys/stat.h>
#include <sys/uadmin.h>
#include <sys/mntent.h>
#include <sys/mnttab.h>
#include <sys/mount.h>
#include <sys/fs/ufs_mount.h>
#include <alloca.h>
#include <assert.h>
```



```
#include <errno.h>
#include <fcntl.h>
#include <libgen.h>
#include <libscf.h>
#include <libscf_priv.h>
#include <limits.h>
#include <locale.h>
#include <libintl.h>
#include <syslog.h>
#include <signal.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <time.h>
#include <wait.h>
#include <ctype.h>
#include <utmpx.h>
#include <pwd.h>
#include <zone.h>
#include <spawn.h>

#include <libzfs.h>
#if defined(__i386)
#include <libgrubmgmt.h>
#endif

#if !defined(TEXT_DOMAIN)
#define TEXT_DOMAIN     "SYS_TEST"
#endif

#if defined(__sparc)
#define CUR_ELFDATA     ELFDATA2MSB
#elif defined(__i386)
#define CUR_ELFDATA     ELFDATA2LSB
#endif

static libzfs_handle_t *g_zfs;

extern int audit_halt_setup(int, char **);
extern int audit_halt_success(void);
extern int audit_halt_fail(void);

extern int audit_reboot_setup(void);
extern int audit_reboot_success(void);
extern int audit_reboot_fail(void);

static char *cmdname;   /* basename(argv[0]), the name of the command */

typedef struct ctidlist_struct {
        ctid_t ctid;
        struct ctidlist_struct *next;
} ctidlist_t;
______unchanged_portion_omitted_

int
main(int argc, char *argv[])
{
        char *ttyn = ttyname(STDERR_FILENO);

        int qflag = 0, needlog = 1, nosync = 0;
        int fast_reboot = 0;
        int prom_reboot = 0;
        uintptr_t mdep = NULL;
        int cmd, fcn, c, aval, r;
```

```
1262         const char *usage;
1263         const char *optstring;
1264         zoneid_t zoneid = getzoneid();
1265         int need_check_zones = 0;
1266         char bootargs_buf[BOOTARGS_MAX];
1267         char *bootargs_orig = NULL;
1268         char *bename = NULL;

1270         const char * const resetting = "/etc/svc/volatile/resetting";

1272         (void) setlocale(LC_ALL, "");
1273         (void) textdomain(TEXT_DOMAIN);

1275         cmdname = basename(argv[0]);

1277         if (strcmp(cmdname, "halt") == 0) {
1278                 (void) audit_halt_setup(argc, argv);
1279                 optstring = "dlnqy";
1280                 usage = gettext("usage: %s [ -dlnqy ]\n");
1281                 cmd = A_SHUTDOWN;
1282                 fcn = AD_HALT;
1283         } else if (strcmp(cmdname, "poweroff") == 0) {
1284                 (void) audit_halt_setup(argc, argv);
1285                 optstring = "dlnqy";
1286                 usage = gettext("usage: %s [ -dlnqy ]\n");
1287                 cmd = A_SHUTDOWN;
1288                 fcn = AD_POWEROFF;
1289         } else if (strcmp(cmdname, "reboot") == 0) {
1290                 (void) audit_reboot_setup();
1291 #if defined(__i386)
1292                 optstring = "dlnqpfe:";
1293                 usage = gettext("usage: %s [ -dlnq(p|fe:) ] [ boot args ]\n");
1294 #else
1295                 optstring = "dlnqfp";
1296                 usage = gettext("usage: %s [ -dlnq(p|f) ] [ boot args ]\n");
1297 #endif
1298                 cmd = A_SHUTDOWN;
1299                 fcn = AD_BOOT;
1300         } else {
1301                 (void) fprintf(stderr,
1302                     gettext("%s: not installed properly\n"), cmdname);
1303                 return (1);
1304         }

1306         while ((c = getopt(argc, argv, optstring)) != EOF) {
1307                 switch (c) {
1308                 case 'd':
1309                         if (zoneid == GLOBAL_ZONEID)
1310                                 cmd = A_DUMP;
1311                         else {
1312                                 (void) fprintf(stderr,
1313                                     gettext("%s: -d only valid from global"
1314                                     " zone\n"), cmdname);
1315                                 return (1);
1316                         }
1317                         break;
1318                 case 'l':
1319                         needlog = 0;
1320                         break;
1321                 case 'n':
1322                         nosync = 1;
1323                         break;
1324                 case 'q':
1325                         qflag = 1;
1326                         break;
1327                 case 'y':
```



```
1328                         /*
1329                          * Option ignored for backwards compatibility.
1330                          */
1327                         ttyn = NULL;
1331                         break;
1332                 case 'f':
1333                         fast_reboot = 1;
1334                         break;
1335                 case 'p':
1336                         prom_reboot = 1;
1337                         break;
1338 #if defined(__i386)
1339                 case 'e':
1340                         bename = optarg;
1341                         break;
1342 #endif
1343                 default:
1344                         /*
1345                          * TRANSLATION_NOTE
1346                          * Don't translate the words "halt" or "reboot"
1347                          */
1348                         (void) fprintf(stderr, usage, cmdname);
1349                         return (1);
1350                 }
1351         }

1353         argc -= optind;
1354         argv += optind;

1356         if (argc != 0) {
1357                 if (fcn != AD_BOOT) {
1358                         (void) fprintf(stderr, usage, cmdname);
1359                         return (1);
1360                 }

1362                 /* Gather the arguments into bootargs_buf. */
1363                 if (gather_args(argv, bootargs_buf, sizeof (bootargs_buf)) !=
1364                     0) {
1365                         (void) fprintf(stderr,
1366                             gettext("%s: Boot arguments too long.\n"), cmdname);
1367                         return (1);
1368                 }

1370                 bootargs_orig = strdup(bootargs_buf);
1371                 mdep = (uintptr_t)bootargs_buf;
1372         } else {
1373                 /*
1374                  * Initialize it to 0 in case of fastboot, the buffer
1375                  * will be used.
1376                  */
1377                 bzero(bootargs_buf, sizeof (bootargs_buf));
1378         }

1380         if (geteuid() != 0) {
1381                 (void) fprintf(stderr,
1382                     gettext("%s: permission denied\n"), cmdname);
1383                 goto fail;
1384         }

1386         if (fast_reboot && prom_reboot) {
1387                 (void) fprintf(stderr,
1388                     gettext("%s: -p and -f are mutually exclusive\n"),
1389                     cmdname);
1390                 return (EINVAL);
1391         }
1392         /*
```

```
1393          * Check whether fast reboot is the default operating mode
1394          */
1395         if (fcn == AD_BOOT && !fast_reboot && !prom_reboot &&
1396             zoneid == GLOBAL_ZONEID) {
1397                 fast_reboot = scf_is_fastboot_default();

1399         }

1401         if (bename && !fast_reboot)     {
1402                 (void) fprintf(stderr, gettext("%s: -e only valid with -f\n"),
1403                     cmdname);
1404                 return (EINVAL);
1405         }

1407 #if defined(__sparc)
1408         if (fast_reboot) {
1409                 fast_reboot = 2;        /* need to distinguish each case */
1410         }
1411 #endif

1413         /*
1414          * If fast reboot, do some sanity check on the argument
1415          */
1416         if (fast_reboot == 1) {
1417                 int rc;
1418                 int is_dryrun = 0;

1420                 if (zoneid != GLOBAL_ZONEID)    {
1421                         (void) fprintf(stderr,
1422                             gettext("%s: Fast reboot only valid from global"
1423                             " zone\n"), cmdname);
1424                         return (EINVAL);
1425                 }

1427                 rc = parse_fastboot_args(bootargs_buf, sizeof (bootargs_buf),
1428                     &is_dryrun, bename);

1430                 /*
1431                  * If dry run, or if arguments are invalid, return.
1432                  */
1433                 if (is_dryrun)
1434                         return (rc);
1435                 else if (rc == EINVAL)
1436                         goto fail;
1437                 else if (rc != 0)
1438                         fast_reboot = 0;

1440                 /*
1441                  * For all the other errors, we continue on in case user
1442                  * user want to force fast reboot, or fall back to regular
1443                  * reboot.
1444                  */
1445                 if (strlen(bootargs_buf) != 0)
1446                         mdep = (uintptr_t)bootargs_buf;
1447         }

1449 #if 0   /* For debugging */
1450         if (mdep != NULL)
1451                 (void) fprintf(stderr, "mdep = %s\n", (char *)mdep);
1452 #endif

1451         if (fcn != AD_BOOT && ttyn != NULL &&
1452             strncmp(ttyn, "/dev/term/", strlen("/dev/term/")) == 0) {
1453                 /*
1454                  * TRANSLATION_NOTE
1455                  * Don't translate ''halt -y''
```



```
1456                  */
1457                 (void) fprintf(stderr,
1458                     gettext("%s: dangerous on a dialup;"), cmdname);
1459                 (void) fprintf(stderr,
1460                     gettext("use ''%s -y'' if you are really sure\n"), cmdname);
1461                 goto fail;
1462         }

1454         if (needlog) {
1455                 char *user = getlogin();
1456                 struct passwd *pw;
1457                 char *tty;

1459                 openlog(cmdname, 0, LOG_AUTH);
1460                 if (user == NULL && (pw = getpwuid(getuid())) != NULL)
1461                         user = pw->pw_name;
1462                 if (user == NULL)
1463                         user = "root";

1465                 tty = ttyname(1);

1467                 if (tty == NULL)
1468                         syslog(LOG_CRIT, "initiated by %s", user);
1469                 else
1470                         syslog(LOG_CRIT, "initiated by %s on %s", user, tty);
1471         }

1473         /*
1474          * We must assume success and log it before auditd is terminated.
1475          */
1476         if (fcn == AD_BOOT)
1477                 aval = audit_reboot_success();
1478         else
1479                 aval = audit_halt_success();

1481         if (aval == -1) {
1482                 (void) fprintf(stderr,
1483                     gettext("%s: can't turn off auditd\n"), cmdname);
1484                 if (needlog)
1485                         (void) sleep(5); /* Give syslogd time to record this */
1486         }

1488         (void) signal(SIGHUP, SIG_IGN); /* for remote connections */

1490         /*
1491          * We start to fork a bunch of zoneadms to halt any active zones.
1492          * This will proceed with halt in parallel until we call
1493          * check_zone_haltedness later on.
1494          */
1495         if (zoneid == GLOBAL_ZONEID && cmd != A_DUMP) {
1496                 need_check_zones = halt_zones();
1497         }

1499 #if defined(__i386)
1500         /* set new default entry in the GRUB entry */
1501         if (fbarg_entnum != GRUB_ENTRY_DEFAULT) {
1502                 char buf[32];
1503                 (void) snprintf(buf, sizeof (buf), "default=%u", fbarg_entnum);
1504                 (void) halt_exec(BOOTADM_PROG, "set-menu", buf, NULL);
1505         }
1506 #endif  /* __i386 */

1508         /* if we're dumping, do the archive update here and don't defer it */
1509         if (cmd == A_DUMP && zoneid == GLOBAL_ZONEID && !nosync)
1510                 do_archives_update(fast_reboot);
```

```
1512         /*
1513          * If we're not forcing a crash dump, mark the system as quiescing for
1514          * smf(5)'s benefit, and idle the init process.
1515          */
1516         if (cmd != A_DUMP) {
1517                 if (direct_init(PCDSTOP) == -1) {
1518                         /*
1519                          * TRANSLATION_NOTE
1520                          * Don't translate the word "init"
1521                          */
1522                         (void) fprintf(stderr,
1523                             gettext("%s: can't idle init\n"), cmdname);
1524                         goto fail;
1525                 }

1527                 if (creat(resetting, 0755) == -1)
1528                         (void) fprintf(stderr,
1529                             gettext("%s: could not create %s.\n"),
1530                             cmdname, resetting);
1531         }

1533         /*
1534          * Make sure we don't get stopped by a jobcontrol shell
1535          * once we start killing everybody.
1536          */
1537         (void) signal(SIGTSTP, SIG_IGN);
1538         (void) signal(SIGTTIN, SIG_IGN);
1539         (void) signal(SIGTTOU, SIG_IGN);
1540         (void) signal(SIGPIPE, SIG_IGN);
1541         (void) signal(SIGTERM, SIG_IGN);

1543         /*
1544          * Try to stop gdm so X has a chance to return the screen and
1545          * keyboard to a sane state.
1546          */
1547         if (fast_reboot == 1 && stop_gdm() != 0) {
1548                 (void) fprintf(stderr,
1549                     gettext("%s: Falling back to regular reboot.\n"), cmdname);
1550                 fast_reboot = 0;
1551                 mdep = (uintptr_t)bootargs_orig;
1552         } else if (bootargs_orig) {
1553                 free(bootargs_orig);
1554         }

1556         if (cmd != A_DUMP) {
1557                 /*
1558                  * Stop all restarters so they do not try to restart services
1559                  * that are terminated.
1560                  */
1561                 stop_restarters();

1563                 /*
1564                  * Wait a little while for zones to shutdown.
1565                  */
1566                 if (need_check_zones) {
1567                         check_zones_haltedness();

1569                         (void) fprintf(stderr,
1570                             gettext("%s: Completing system halt.\n"),
1571                             cmdname);
1572                 }
1573         }

1575         /*
1576          * If we're not forcing a crash dump, give everyone 5 seconds to
1577          * handle a SIGTERM and clean up properly.
```



```
1578          */
1579         if (cmd != A_DUMP) {
1580                 int     start, end, delta;

1582                 (void) kill(-1, SIGTERM);
1583                 start = time(NULL);

1585                 if (zoneid == GLOBAL_ZONEID && !nosync)
1586                         do_archives_update(fast_reboot);

1588                 end = time(NULL);
1589                 delta = end - start;
1590                 if (delta < 5)
1591                         (void) sleep(5 - delta);
1592         }

1594         (void) signal(SIGINT, SIG_IGN);

1596         if (!qflag && !nosync) {
1597                 struct utmpx wtmpx;

1599                 bzero(&wtmpx, sizeof (struct utmpx));
1600                 (void) strcpy(wtmpx.ut_line, "~");
1601                 (void) time(&wtmpx.ut_tv.tv_sec);

1603                 if (cmd == A_DUMP)
1604                         (void) strcpy(wtmpx.ut_name, "crash dump");
1605                 else
1606                         (void) strcpy(wtmpx.ut_name, "shutdown");

1608                 (void) updwtmpx(WTMPX_FILE, &wtmpx);
1609                 sync();
1610         }

1612         if (cmd == A_DUMP && nosync != 0)
1613                 (void) uadmin(A_DUMP, AD_NOSYNC, NULL);

1615         if (fast_reboot)
1616                 fcn = AD_FASTREBOOT;

1618         if (uadmin(cmd, fcn, mdep) == -1)
1619                 (void) fprintf(stderr, "%s: uadmin failed: %s\n",
1620                     cmdname, strerror(errno));
1621         else
1622                 (void) fprintf(stderr, "%s: uadmin unexpectedly returned 0\n",
1623                     cmdname);

1625         do {
1626                 r = remove(resetting);
1627         } while (r != 0 && errno == EINTR);

1629         if (r != 0 && errno != ENOENT)
1630                 (void) fprintf(stderr, gettext("%s: could not remove %s.\n"),
1631                     cmdname, resetting);

1633         if (direct_init(PCRUN) == -1) {
1634                 /*
1635                  * TRANSLATION_NOTE
1636                  * Don't translate the word "init"
1637                  */
1638                 (void) fprintf(stderr,
1639                     gettext("%s: can't resume init\n"), cmdname);
1640         }

1642         continue_restarters();
```

```
1644         if (get_initpid() != -1)
1645                 /* tell init to restate current level */
1646                 (void) kill(get_initpid(), SIGHUP);

1648 fail:
1649         if (fcn == AD_BOOT)
1650                 (void) audit_reboot_fail();
1651         else
1652                 (void) audit_halt_fail();

1654         if (fast_reboot == 1) {
1655                 if (bename) {
1656                         (void) halt_exec(BEADM_PROG, "umount", bename, NULL);

1658                 } else if (strlen(fastboot_mounted) != 0) {
1659                         (void) umount(fastboot_mounted);
1660 #if defined(__i386)
1661                 } else if (fbarg_used != NULL) {
1662                         grub_cleanup_boot_args(fbarg_used);
1663 #endif  /* __i386 */
1664                 }
1665         }

1667         return (1);
1668 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2824 Fri Jul 26 18:54:21 2013
new/usr/src/man/man1m/halt.1m
3913 there is no dialup, only zuul
**********************************************************
'\" te
.\" Copyright (c) 2004 Sun Microsystems, Inc. All Rights Reserved.
.\" Copyright (c) 2013, Joyent, Inc. All rights reserved.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH HALT 1M "Jul 26, 2013"
.TH HALT 1M "Nov 2, 2004"
.SH NAME
halt, poweroff \- stop the processor
.SH SYNOPSIS
.LP
.nf
\fB/usr/sbin/halt\fR [\fB-dlnqy\fR]
.fi

.LP
.nf
\fB/usr/sbin/poweroff\fR [\fB-dlnqy\fR]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBhalt\fR and \fBpoweroff\fR utilities write any pending information to
the disks and then stop the processor. The \fBpoweroff\fR utility has the
machine remove power, if possible.
.sp
.LP
The \fBhalt\fR and \fBpoweroff\fR utilities normally log the system shutdown to
the system log daemon, \fBsyslogd\fR(1M), and place a shutdown record in the
login accounting file \fB/var/adm/wtmpx\fR. These actions are inhibited if the
\fB-n\fR or \fB-q\fR options are present.
.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-d\fR\fR
.ad
.RS 6n
Force a system crash dump before rebooting. See \fBdumpadm\fR(1M) for
information on configuring system crash dumps.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR\fR
.ad
.RS 6n
Suppress sending a message to the system log daemon, \fBsyslogd\fR(1M), about
who executed \fBhalt\fR.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR\fR
```



```
.ad
.RS 6n
Prevent the \fBsync\fR(1M) before stopping.
.RE

.sp
.ne 2
.na
\fB\fB-q\fR\fR
.ad
.RS 6n
Quick halt. No graceful shutdown is attempted.
.RE

.sp
.ne 2
.na
\fB\fB-y\fR\fR
.ad
.RS 6n
This option is ignored for backwards compatibility.
Halt the system, even from a dialup terminal.
.RE

.SH FILES
.sp
.ne 2
.na
\fB\fB/var/adm/wtmpx\fR\fR
.ad
.RS 18n
History of user access and administration information.
.RE

.SH SEE ALSO
.sp
.LP
\fBdumpadm\fR(1M), \fBinit\fR(1M), \fBreboot\fR(1M), \fBshutdown\fR(1M),
\fBsync\fR(1M), \fBsyslogd\fR(1M), \fBinittab\fR(4), \fBattributes\fR(5),
\fBsmf\fR(5)
.SH NOTES
.sp
.LP
The \fBhalt\fR and \fBpoweroff\fR utilities do not cleanly shutdown
\fBsmf\fR(5) services. Execute the scripts in \fB/etc/rcnum.d\fR or execute
shutdown actions in \fBinittab\fR(4). To ensure a complete shutdown of system
services, use \fBshutdown\fR(1M) or \fBinit\fR(1M) to reboot a Solaris system.
```