```
**********************************************************
   77783 Wed Aug 27 14:22:07 2008
new/usr/src/cmd/cmd-inet/usr.sbin/ipsecutils/ikeadm.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <unistd.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <sys/sysconf.h>
#include <string.h>
#include <strings.h>
#include <libintl.h>
#include <locale.h>
#include <ctype.h>
#include <time.h>
#include <sys/sysmacros.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>
#include <assert.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <door.h>
#include <setjmp.h>

#include <ipsec_util.h>
#include <ikedoor.h>

static int      doorfd = -1;

/*
 * These are additional return values for the command line parsing
 * function (parsecmd()).  They are specific to this utility, but
 * need to share the same space as the IKE_SVC_* defs, without conflicts.
```



```
 * So they're defined relative to the end of that range.
 */
#define IKEADM_HELP_GENERAL     IKE_SVC_MAX + 1
#define IKEADM_HELP_GET         IKE_SVC_MAX + 2
#define IKEADM_HELP_SET         IKE_SVC_MAX + 3
#define IKEADM_HELP_ADD         IKE_SVC_MAX + 4
#define IKEADM_HELP_DEL         IKE_SVC_MAX + 5
#define IKEADM_HELP_DUMP        IKE_SVC_MAX + 6
#define IKEADM_HELP_FLUSH       IKE_SVC_MAX + 7
#define IKEADM_HELP_READ        IKE_SVC_MAX + 8
#define IKEADM_HELP_WRITE       IKE_SVC_MAX + 9
#define IKEADM_HELP_HELP        IKE_SVC_MAX + 10
#define IKEADM_EXIT             IKE_SVC_MAX + 11

static void
command_complete(int s)
{
        if (interactive) {
                longjmp(env, 1);
        } else {
                exit(s);
        }
}
```

unchanged_portion_omitted_

```
/*
 * Parsing functions
 */

/* stolen from ipseckey.c, with a second tier added */
static int
parsecmd(char *cmdstr, char *objstr)
{
#define MAXOBJS         10
        struct objtbl {
                char    *obj;
                int     token;
        };
        static struct cmdtbl {
                char            *cmd;
                int             null_obj_token;
                struct objtbl   objt[MAXOBJS];
        } table[] = {
                {"get", IKE_SVC_ERROR, {
                                {"debug",       IKE_SVC_GET_DBG},
                                {"priv",        IKE_SVC_GET_PRIV},
                                {"stats",       IKE_SVC_GET_STATS},
                                {"p1",          IKE_SVC_GET_P1},
                                {"rule",        IKE_SVC_GET_RULE},
                                {"preshared",   IKE_SVC_GET_PS},
                                {"defaults",    IKE_SVC_GET_DEFS},
                                {NULL,          IKE_SVC_ERROR}
                                {NULL,          IKE_SVC_ERROR},
                        }
                },
                {"set", IKE_SVC_ERROR, {
                                {"debug",       IKE_SVC_SET_DBG},
                                {"priv",        IKE_SVC_SET_PRIV},
                                {NULL,          IKE_SVC_ERROR}
                                {NULL,          IKE_SVC_ERROR},
                        }
                },
                {"add", IKE_SVC_ERROR, {
                                {"rule",        IKE_SVC_NEW_RULE},
                                {"preshared",   IKE_SVC_NEW_PS},
                                {NULL,          IKE_SVC_ERROR}
```

```
391                             {NULL,          IKE_SVC_ERROR},
390                     }
391             },
392             {"del", IKE_SVC_ERROR, {
393                             {"p1",          IKE_SVC_DEL_P1},
394                             {"rule",        IKE_SVC_DEL_RULE},
395                             {"preshared",   IKE_SVC_DEL_PS},
396                             {NULL,          IKE_SVC_ERROR}
398                             {NULL,          IKE_SVC_ERROR},
397                     }
398             },
399             {"dump", IKE_SVC_ERROR, {
400                             {"p1",          IKE_SVC_DUMP_P1S},
401                             {"rule",        IKE_SVC_DUMP_RULES},
402                             {"preshared",   IKE_SVC_DUMP_PS},
403                             {NULL,          IKE_SVC_ERROR}
405                             {NULL,          IKE_SVC_ERROR},
404                     }
405             },
406             {"flush", IKE_SVC_ERROR, {
407                             {"p1",          IKE_SVC_FLUSH_P1S},
408                             {NULL,          IKE_SVC_ERROR}
410                             {NULL,          IKE_SVC_ERROR},
409                     }
410             },
411             {"read", IKE_SVC_ERROR, {
412                             {"rule",        IKE_SVC_READ_RULES},
413                             {"preshared",   IKE_SVC_READ_PS},
414                             {NULL,          IKE_SVC_ERROR}
416                             {NULL,          IKE_SVC_ERROR},
415                     }
416             },
417             {"write", IKE_SVC_ERROR, {
418                             {"rule",        IKE_SVC_WRITE_RULES},
419                             {"preshared",   IKE_SVC_WRITE_PS},
420                             {NULL,          IKE_SVC_ERROR}
422                             {NULL,          IKE_SVC_ERROR},
421                     }
422             },
423             {"help", IKEADM_HELP_GENERAL, {
424                             {"get",         IKEADM_HELP_GET},
425                             {"set",         IKEADM_HELP_SET},
426                             {"add",         IKEADM_HELP_ADD},
427                             {"del",         IKEADM_HELP_DEL},
428                             {"dump",        IKEADM_HELP_DUMP},
429                             {"flush",       IKEADM_HELP_FLUSH},
430                             {"read",        IKEADM_HELP_READ},
431                             {"write",       IKEADM_HELP_WRITE},
432                             {"help",        IKEADM_HELP_HELP},
433                             {NULL,          IKE_SVC_ERROR}
435                             {NULL,          IKE_SVC_ERROR},
434                     }
435             },
436             {"exit", IKEADM_EXIT, {
437                             {NULL,          IKE_SVC_ERROR}
439                             {NULL,          IKE_SVC_ERROR},
438                     }
439             },
440             {"quit", IKEADM_EXIT, {
441                             {NULL,          IKE_SVC_ERROR}
443                             {NULL,          IKE_SVC_ERROR},
442                     }
443             },
444             {"dbg", IKE_SVC_ERROR, {
445                             {"rbdump",      IKE_SVC_DBG_RBDUMP},
446                             {NULL,          IKE_SVC_ERROR}
```



```
448                             {NULL,          IKE_SVC_ERROR},
447                     }
448             },
449             {NULL,  IKE_SVC_ERROR, {
450                             {NULL,          IKE_SVC_ERROR}
452                             {NULL,          IKE_SVC_ERROR},
451                     }
452             }
454             },
453     };
454     struct cmdtbl   *ct = table;
455     struct objtbl   *ot;

457     if (cmdstr == NULL) {
458             return (IKE_SVC_ERROR);
459     }

461     while (ct->cmd != NULL && strcmp(ct->cmd, cmdstr) != 0)
462             ct++;
463     ot = ct->objt;

465     if (ct->cmd == NULL) {
466             message(gettext("Unrecognized command '%s'"), cmdstr);
467             return (ot->token);
468     }

470     if (objstr == NULL) {
471             return (ct->null_obj_token);
472     }

474     while (ot->obj != NULL && strcmp(ot->obj, objstr) != 0)
475             ot++;

477     if (ot->obj == NULL)
478             message(gettext("Unrecognized object '%s'"), objstr);

480     return (ot->token);
481 }
```

unchanged_portion_omitted_

```
1141 /* stolen from libdhcputil (dhcp_inittab.c) */
1142 static uint64_t
1143 ike_ntohll(uint64_t nll)
1144 {
1145 #ifdef  _LITTLE_ENDIAN
1146         return ((uint64_t)ntohl(nll & 0xffffffff) << 32 | ntohl(nll >> 32));
1147 #else
1148         return (nll);
1149 #endif
1150 }

1139 /*
1140  * Printing functions
1141  *
1142  * A potential point of confusion here is that the ikeadm-specific string-
1143  * producing functions do not match the ipsec_util.c versions in style: the
1144  * ikeadm-specific functions return a string (and are named foostr), while
1145  * the ipsec_util.c functions actually print the string to the file named
1146  * in the second arg to the function (and are named dump_foo).
1147  *
1148  * Localization for ikeadm seems more straightforward when complete
1149  * phrases are translated rather than: a part of a phrase, a call to
1150  * dump_foo(), and more of the phrase.  It could also accommodate
1151  * non-English grammar more easily.
1161  * The reason for this is that in the context of the ikeadm output, it
1162  * seemed like the localization of the text would be more straightforward
```

```
 * (and could more easily accomodate non-english grammar!) if more complete
 * phrases were being translated, rather than a bit of a phrase followed by
 * a call to dump_foo() followed by more of the phrase.
 */

static char *
errstr(int err)
{
        static char     rtn[MAXLINESIZE];

        switch (err) {
        case IKE_ERR_NO_OBJ:
                return (gettext("No data returned"));
        case IKE_ERR_NO_DESC:
                return (gettext("No destination provided"));
        case IKE_ERR_ID_INVALID:
                return (gettext("Id info invalid"));
        case IKE_ERR_LOC_INVALID:
                return (gettext("Destination invalid"));
        case IKE_ERR_CMD_INVALID:
                return (gettext("Command invalid"));
        case IKE_ERR_DATA_INVALID:
                return (gettext("Supplied data invalid"));
        case IKE_ERR_CMD_NOTSUP:
                return (gettext("Unknown command"));
        case IKE_ERR_REQ_INVALID:
                return (gettext("Request invalid"));
        case IKE_ERR_NO_PRIV:
                return (gettext("Not allowed at current privilege level"));
        case IKE_ERR_SYS_ERR:
                return (gettext("System error"));
        case IKE_ERR_DUP_IGNORED:
                return (gettext("One or more duplicate entries ignored"));
        default:
                (void) snprintf(rtn, MAXLINESIZE,
                    gettext("<unknown error %d>"), err);
                return (rtn);
        }
}
______unchanged_portion_omitted_

static void
print_hdr(char *prefix, ike_p1_hdr_t *hdrp)
{
        (void) printf(
            gettext("%s Cookies: Initiator 0x%llx  Responder 0x%llx\n"),
            prefix, ntohll(hdrp->p1hdr_cookies.cky_i),
            ntohll(hdrp->p1hdr_cookies.cky_r));
            prefix, ike_ntohll(hdrp->p1hdr_cookies.cky_i),
            ike_ntohll(hdrp->p1hdr_cookies.cky_r));
        (void) printf(gettext("%s The local host is the %s.\n"), prefix,
            hdrp->p1hdr_isinit ? gettext("initiator") : gettext("responder"));
        (void) printf(gettext("%s ISAKMP version %d.%d; %s exchange\n"), prefix,
            hdrp->p1hdr_major, hdrp->p1hdr_minor, xchgstr(hdrp->p1hdr_xchg));
        (void) printf(gettext("%s Current state is %s"), prefix,
            statestr(hdrp->p1hdr_state));
        (void) printf("\n");
}
______unchanged_portion_omitted_
```

```
**********************************************************
   14282 Wed Aug 27 14:22:12 2008
new/usr/src/cmd/iscsi/iscsitgtd/t10_spc.h
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _T10_SPC_H
#define _T10_SPC_H

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

#ifdef __cplusplus
extern "C" {
#endif

/*
 * []------------------------------------------------------------------[]
 * | SPC-3                                                            |
 * []------------------------------------------------------------------[]
 */

/*
 * FIXED_SENSE_ADDL_INFO_LEN is the length of INFORMATION field
 * in fixed format sense data
 */
#define FIXED_SENSE_ADDL_INFO_LEN 0xFFFFFFFF
#define INFORMATION_SENSE_DESCR sizeof (struct scsi_information_sense_descr)

#include <sys/types.h>
#include <netinet/in.h>
#include <sys/scsi/generic/inquiry.h>
#include <sys/scsi/generic/mode.h>

/*
 * SPC Command Functions
 */
void spc_tur(struct t10_cmd *cmd, uint8_t *cdb, size_t cdb_len);
void spc_request_sense(struct t10_cmd *cmd, uint8_t *cdb, size_t cdb_len);
void spc_unsupported(struct t10_cmd *cmd, uint8_t *cdb, size_t cdb_len);
```



```
void spc_inquiry(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);
void spc_mselect(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);
void spc_mselect_data(t10_cmd_t *cmd, emul_handle_t id, size_t offset,
    char *data, size_t data_len);
void spc_report_luns(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);
void spc_report_tpgs(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);
void spc_msense(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);
void spc_startstop(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);
void spc_send_diag(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);

/*
 * SPC Support Functions
 */
void spc_cmd_offline(t10_cmd_t *cmd, uint8_t *cdb, size_t cdb_len);
void spc_sense_create(struct t10_cmd *cmd, int sense_key, int addl_sense_len);
void spc_sense_ascq(struct t10_cmd *cmd, int asc, int ascq);
void spc_sense_info(t10_cmd_t *cmd, uint64_t info);
void spc_sense_flags(t10_cmd_t *cmd, int flags);
void spc_sense_raw(t10_cmd_t *cmd, uchar_t *sense_buf, size_t sense_len);
Boolean_t spc_decode_lu_addr(uint8_t *buf, int len, uint32_t *val);
Boolean_t spc_encode_lu_addr(uint8_t *buf, int select_field, uint32_t lun);

/*
 * SPC flags to use when setting various sense code flags
 */
#define SPC_SENSE_EOM   0x01
#define SPC_SENSE_FM    0x02
#define SPC_SENSE_ILI   0x04

#ifdef _BIG_ENDIAN
#define htonll(x)   (x)
#define ntohll(x)   (x)
#else
#define htonll(x)   ((((unsigned long long)htonl(x & 0xffffffff)) << 32) + \
                    htonl(x >> 32))
#define ntohll(x)   ((((unsigned long long)ntohl(x)) << 32) + ntohl(x >> 32))
#endif

/*
 * []------------------------------------------------------------------[]
 * | SPC-3, revision 21c -- ASC/ASCQ values                           |
 * | The full tables can be found in Appendix D (numerical order) or  |
 * | section 4.5.6 (alphabetical order). There are close to fifteen   |
 * | pages of values which will not be included here. Only those used |
 * | by the code.                                                     |
 * []------------------------------------------------------------------[]
 */
#define SPC_ASC_FM_DETECTED     0x00 /* file-mark detected */
#define SPC_ASCQ_FM_DETECTED    0x01

#define SPC_ASC_EOP             0x00 /* end-of-partition/medium detected */
#define SPC_ASCQ_EOP            0x02

#define SPC_ASC_IN_PROG         0x04
#define SPC_ASCQ_IN_PROG        0x07

#define SPC_ASC_WRITE_ERROR     0x0c
#define SPC_ASCQ_WRITE_ERROR    0x00

#define SPC_ASC_PARAM_LIST_LEN  0x1a /* Parameter List Length Error */
#define SPC_ASCQ_PARAM_LIST_LEN 0x00

#define SPC_ASC_MISCOMPARE      0x1d
#define SPC_ASCQ_MISCOMPARE     0x00

#define SPC_ASC_INVALID_LU      0x20
```

```
#define SPC_ASCQ_INVALID_LU     0x09

#define SPC_ASC_BLOCK_RANGE     0x21
#define SPC_ASCQ_BLOCK_RANGE    0x00

#define SPC_ASC_INVALID_FIELD_IN_PARAMETER_LIST         0x26
#define SPC_ASCQ_INVALID_FIELD_IN_PARAMETER_LIST        0x00

#define SPC_ASC_INVALID_CDB     0x24
#define SPC_ASCQ_INVALID_CDB    0x00

#define SPC_ASC_PARAMETERS_CHANGED      0x2a
#define SPC_ASCQ_RES_PREEMPTED  0x03
#define SPC_ASCQ_RES_RELEASED   0x04

#define SPC_ASC_PWR_RESET       0x29
#define SPC_ASCQ_PWR_RESET      0x00

#define SPC_ASC_PWR_ON          0x29
#define SPC_ASCQ_PWR_ON         0x01

#define SPC_ASC_BUS_RESET       0x29
#define SPC_ASCQ_BUS_RESET      0x02

#define SPC_ASC_CAP_CHANGE      0x2a
#define SPC_ASCQ_CAP_CHANGE     0x09

#define SPC_ASC_DATA_PATH       0x41
#define SPC_ASCQ_DATA_PATH      0x00

#define SPC_ASC_MEMORY_OUT_OF   0x55 /* Auxillary Memory Out Of Space */
#define SPC_ASCQ_MEMORY_OUT_OF  0x00
#define SPC_ASCQ_RESERVATION_FAIL 0x02


/*
 * []------------------------------------------------------------------[]
 * | SAM-3, revision 14, section 5.2 - Command descriptor block (CDB)   |
 * |                                                                    |
 * | "All CDBs shall contain a CONTROL byte (see table 21). The         |
 * | location of the CONTROL byte within a CDB depends on the CDB       |
 * | format (see SPC-3)."                                               |
 * |                                                                    |
 * | bits       meaning                                                 |
 * | 6-7        vendor specific (we don't use so must be zero)          |
 * | 3-5        reserved must be zero                                   |
 * | 2          NACA (currently we don't support so must be zero)       |
 * | 1          Obsolete                                                |
 * | 0          Link (currently we don't support so must be zero)       |
 * |                                                                    |
 * | So, this means the control byte must be zero and therefore if      |
 * | this macro returns a non-zero value the emulation code should      |
 * | return a CHECK CONDITION with status set to ILLEGAL REQUEST        |
 * | and the additional sense code set to INVALID FIELD IN CDB.         |
 * |                                                                    |
 * | In the future this will likely change with support routines        |
 * | added for dealing with NACA and Linked commands.                   |
 * []------------------------------------------------------------------[]
 */
#define SAM_CONTROL_BYTE_RESERVED(byte) (byte)

/* ---- Disable Block Descriptors ---- */
#define SPC_MODE_SENSE_DBD      0x8

#define SPC_GROUP4_SERVICE_ACTION_MASK  0x1f
```



```
#define SPC_SEND_DIAG_SELFTEST  0x04

/*
 * []------------------------------------------------------------------[]
 * | SPC-3 revision 21c, section 6.4 -- INQUIRY                         |
 * | Various defines. The structure for the inquiry command can be      |
 * | found in /usr/include/sys/scsi/generic/inquiry.h                   |
 * []------------------------------------------------------------------[]
 */
#define SPC_INQUIRY_CODE_SET_BINARY     1
#define SPC_INQUIRY_CODE_SET_ASCII      2
#define SPC_INQUIRY_CODE_SET_UTF8       3

/* ---- Table 82: Inquiry Version ----  */
#define SPC_INQ_VERS_NONE       0x00
#define SPC_INQ_VERS_OBSOLETE   0x02
#define SPC_INQ_VERS_SPC_1      0x03
#define SPC_INQ_VERS_SPC_2      0x04
#define SPC_INQ_VERS_SPC_3      0x05

/* ---- INQUIRY Response Data Format field ---- */
#define SPC_INQ_RDF             0x02    /* all other values are OBSOLETE */

/*
 * Table 85 -- Version descriptor values
 * There are many, many different values available, so we'll only include
 * those that we actually use.
 */
#define SPC_INQ_VD_SAM3         0x0076
#define SPC_INQ_VD_SPC3         0x0307
#define SPC_INQ_VD_SBC2         0x0322
#define SPC_INQ_VD_SSC3         0x0400
#define SPC_INQ_VD_OSD          0x0355

/* --- Version Descriptor length details --- */
#define SPC_INQ_VD_IDX          0x3A
#define SPC_INQ_VD_LEN          0x10

#define SPC_INQ_PAGE0           0x00
#define SPC_INQ_PAGE80          0x80
#define SPC_INQ_PAGE83          0x83
#define SPC_INQ_PAGE86          0x86

/* ---- REPORT LUNS select report has valid values of 0, 1, or 2 ---- */
#define SPC_RPT_LUNS_SELECT_MASK        0x03

/* ---- Table 293: IDENTIFIER TYPE field ---- */
#define SPC_INQUIRY_ID_TYPE_T10ID       1       /* ref 7.6.4.3 */
#define SPC_INQUIRY_ID_TYPE_EUI         2       /* ref 7.6.4.4 */
#define SPC_INQUIRY_ID_TYPE_NAA         3       /* ref 7.6.4.5 */
#define SPC_INQUIRY_ID_TYPE_RELATIVE    4       /* ref 7.6.4.6 */
#define SPC_INQUIRY_ID_TYPE_TARG_PORT   5       /* ref 7.6.4.7 */
#define SPC_INQUIRY_ID_TYPE_LUN         6       /* ref 7.6.4.8 */
#define SPC_INQUIRY_ID_TYPE_MD5         7       /* ref 7.6.4.9 */
#define SPC_INQUIRY_ID_TYPE_SCSI        8       /* ref 7.6.4.10 */

/* ---- Table 292: ASSOCIATION field ----  */
#define SPC_INQUIRY_ASSOC_LUN           0
#define SPC_INQUIRY_ASSOC_TARGPORT      1
#define SPC_INQUIRY_ASSOC_TARG          2

/* ---- Table 80: Peripheral qualifier ---- */
#define SPC_INQUIRY_PERIPH_CONN         0
#define SPC_INQUIRY_PERIPH_DISCONN      1
#define SPC_INQUIRY_PERIPH_INVALID      3
```

```
247 /* ---- Table 256: PROTOCOL IDENTIFIER values ---- */
248 #define SPC_INQUIRY_PROTOCOL_FC         0
249 #define SPC_INQUIRY_PROTOCOL_PSCSI      1
250 #define SPC_INQUIRY_PROTOCOL_SSA        2
251 #define SPC_INQUIRY_PROTOCOL_IEEE1394   3
252 #define SPC_INQUIRY_PROTOCOL_SCSIRDMA   4
253 #define SPC_INQUIRY_PROTOCOL_ISCSI      5
254 #define SPC_INQUIRY_PROTOCOL_SAS        6
255 #define SPC_INQUIRY_PROTOCOL_ADT        7
256 #define SPC_INQUIRY_PROTOCOL_ATA        8

258 #define SPC_DEFAULT_TPG 1

260 /*
261  * SPC-3, revision 21c, section 7.6.5
262  * Extended INQUIRY Data VPD page
263  */
264 typedef struct extended_inq_data {
265         struct vpd_hdr  ei_hdr;
266 #if defined(_BIT_FIELDS_LTOH)
267         uchar_t         ei_ref_chk      : 1,
268                         ei_app_chk      : 1,
269                         ei_grd_chk      : 1,
270                         ei_rto          : 1,
271                         ei_rsvd1        : 4;
272         uchar_t         ei_simpsup      : 1,
273                         ei_ordsup       : 1,
274                         ei_headsup      : 1,
275                         ei_prior_sup    : 1,
276                         ei_group_sup    : 1,
277                         ei_rsvd2        : 3;
278         uchar_t         ei_v_sup        : 1,
279                         ei_nv_sup       : 1,
280                         ei_rsvd3        : 6;
281 #elif defined(_BIT_FIELDS_HTOL)
282         uchar_t         ei_ref_rsvd1    : 4,
283                         ei_rto          : 1,
284                         ei_grd_chk      : 1,
285                         ei_app_chk      : 1,
286                         ei_ref_chk      : 1;
287         uchar_t         ei_rsvd2        : 2,
288                         ei_group_sup    : 1,
289                         ei_prior_sup    : 1,
290                         ei_headsup      : 1,
291                         ei_ordsup       : 1,
292                         ei_simpsup      : 1;
293         uchar_t         ei_rsvd3        : 6,
294                         ei_nv_sup       : 1,
295                         ei_v_sup        : 1;
296 #else
297 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
298 #endif
299         uchar_t         ei_rsv4[57];
300 } extended_inq_data_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
   61009 Wed Aug 27 14:22:17 2008
new/usr/src/common/crypto/aes/aes_impl.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/ddi.h>
#include <sys/sysmacros.h>
#include <sys/strsun.h>
#include <netinet/in.h>
#include <sys/crypto/spi.h>
#include <modes/modes.h>
#include "aes_impl.h"
#ifndef _KERNEL
#include <strings.h>
#include <stdlib.h>
#endif  /* !_KERNEL */


/*
 * This file is derived from the file  rijndael-alg-fst.c  taken from the
 * "optimized C code v3.0" on the "rijndael home page"
 * http://www.iaik.tu-graz.ac.at/research/krypto/AES/old/~rijmen/rijndael/
 * pointed by the NIST web-site http://csrc.nist.gov/archive/aes/
 *
 * The following note is from the original file:
 */

/*
 * rijndael-alg-fst.c
 *
 * @version 3.0 (December 2000)
 *
 * Optimised ANSI C code for the Rijndael cipher (now AES)
 *
 * @author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>
```



```
 * @author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>
 * @author Paulo Barreto <paulo.barreto@terra.com.br>
 *
 * This code is hereby placed in the public domain.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
 * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/* EXPORT DELETE START */

#if defined(sun4u) || defined(__amd64)
/* External assembly functions: */
extern void aes_encrypt_impl(const uint32_t rk[], int Nr, const uint32_t pt[4],
        uint32_t ct[4]);
extern void aes_decrypt_impl(const uint32_t rk[], int Nr, const uint32_t ct[4],
        uint32_t pt[4]);
#define AES_ENCRYPT_IMPL                aes_encrypt_impl
#define AES_DECRYPT_IMPL                aes_decrypt_impl

#ifdef  __amd64
extern int rijndael_key_setup_enc(uint32_t rk[], const uint32_t cipherKey[],
        int keyBits);
extern int rijndael_key_setup_dec(uint32_t rk[], const uint32_t cipherKey[],
        int keyBits);
#endif

#else
#define AES_ENCRYPT_IMPL                rijndael_encrypt
#define AES_DECRYPT_IMPL                rijndael_decrypt
#define rijndael_key_setup_enc_raw      rijndael_key_setup_enc
#endif  /* sun4u || __amd64 */

#if defined(_LITTLE_ENDIAN) && !defined(__amd64)
#define AES_BYTE_SWAP
#endif

#ifndef __amd64
/*
 *  Constant tables
 */

/*
 * Te0[x] = S [x].[02, 01, 01, 03];
 * Te1[x] = S [x].[03, 02, 01, 01];
 * Te2[x] = S [x].[01, 03, 02, 01];
 * Te3[x] = S [x].[01, 01, 03, 02];
 * Te4[x] = S [x].[01, 01, 01, 01];
 *
 * Td0[x] = Si[x].[0e, 09, 0d, 0b];
 * Td1[x] = Si[x].[0b, 0e, 09, 0d];
 * Td2[x] = Si[x].[0d, 0b, 0e, 09];
 * Td3[x] = Si[x].[09, 0d, 0b, 0e];
 * Td4[x] = Si[x].[01, 01, 01, 01];
 */

/* Encrypt Sbox constants (for the substitute bytes operation) */
```

```
 125 static const uint32_t Te0[256] =
 126 {
 127         0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,
 128         0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,
 129         0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,
 130         0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,
 131         0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,
 132         0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfbf0f00bU,
 133         0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afafeaU,
 134         0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,
 135         0x75b7b7c2U, 0xe1fdfd1cU, 0x3d9393aeU, 0x4c26266aU,
 136         0x6c36365aU, 0x7e3f3f41U, 0xf5f7f702U, 0x83cccc4fU,
 137         0x6834345cU, 0x51a5a5f4U, 0xd1e5e534U, 0xf9f1f108U,
 138         0xe2717193U, 0xabd8d873U, 0x62313153U, 0x2a15153fU,
 139         0x0804040cU, 0x95c7c752U, 0x46232365U, 0x9dc3c35eU,
 140         0x30181828U, 0x379696a1U, 0x0a05050fU, 0x2f9a9ab5U,
 141         0x0e070709U, 0x24121236U, 0x1b80809bU, 0xdfe2e23dU,
 142         0xcdebeb26U, 0x4e272769U, 0x7fb2b2cdU, 0xea75759fU,
 143         0x1209091bU, 0x1d83839eU, 0x582c2c74U, 0x341a1a2eU,
 144         0x361b1b2dU, 0xdc6e6eb2U, 0xb45a5aeeU, 0x5ba0a0fbU,
 145         0xa45252f6U, 0x763b3b4dU, 0xb7d6d661U, 0x7db3b3ceU,
 146         0x5229297bU, 0xdde3e33eU, 0x5e2f2f71U, 0x13848497U,
 147         0xa65353f5U, 0xb9d1d168U, 0x00000000U, 0xc1eded2cU,
 148         0x40202060U, 0xe3fcfc1fU, 0x79b1b1c8U, 0xb65b5bedU,
 149         0xd46a6abeU, 0x8dcbcb46U, 0x67bebed9U, 0x7239394bU,
 150         0x944a4adeU, 0x984c4cd4U, 0xb05858e8U, 0x85cfcf4aU,
 151         0xbbd0d06bU, 0xc5efef2aU, 0x4faaaae5U, 0xedfbfb16U,
 152         0x864343c5U, 0x9a4d4dd7U, 0x66333355U, 0x11858594U,
 153         0x8a4545cfU, 0xe9f9f910U, 0x04020206U, 0xfe7f7f81U,
 154         0xa05050f0U, 0x783c3c44U, 0x259f9fbaU, 0x4ba8a8e3U,
 155         0xa25151f3U, 0x5da3a3feU, 0x804040c0U, 0x058f8f8aU,
 156         0x3f9292adU, 0x219d9dbcU, 0x70383848U, 0xf1f5f504U,
 157         0x63bcbcdfU, 0x77b6b6c1U, 0xafdada75U, 0x42212163U,
 158         0x20101030U, 0xe5ffff1aU, 0xfdf3f30eU, 0xbfd2d26dU,
 159         0x81cdcd4cU, 0x180c0c14U, 0x26131335U, 0xc3ecec2fU,
 160         0xbe5f5fe1U, 0x359797a2U, 0x884444ccU, 0x2e171739U,
 161         0x93c4c457U, 0x55a7a7f2U, 0xfc7e7e82U, 0x7a3d3d47U,
 162         0xc86464acU, 0xba5d5de7U, 0x3219192bU, 0xe6737395U,
 163         0xc06060a0U, 0x19818198U, 0x9e4f4fd1U, 0xa3dcdc7fU,
 164         0x44222266U, 0x542a2a7eU, 0x3b9090abU, 0x0b888883U,
 165         0x8c4646caU, 0xc7eeee29U, 0x6bb8b8d3U, 0x2814143cU,
 166         0xa7dede79U, 0xbc5e5ee2U, 0x160b0b1dU, 0xaddbdb76U,
 167         0xdbe0e03bU, 0x64323256U, 0x743a3a4eU, 0x140a0a1eU,
 168         0x924949dbU, 0x0c06060aU, 0x4824246cU, 0xb85c5ce4U,
 169         0x9fc2c25dU, 0xbdd3d36eU, 0x43acacefU, 0xc46262a6U,
 170         0x399191a8U, 0x319595a4U, 0xd3e4e437U, 0xf279798bU,
 171         0xd5e7e732U, 0x8bc8c843U, 0x6e373759U, 0xda6d6db7U,
 172         0x018d8d8cU, 0xb1d5d564U, 0x9c4e4ed2U, 0x49a9a9e0U,
 173         0xd86c6cb4U, 0xac5656faU, 0xf3f4f407U, 0xcfeaea25U,
 174         0xca6565afU, 0xf47a7a8eU, 0x47aeaee9U, 0x10080818U,
 175         0x6fbabad5U, 0xf0787888U, 0x4a25256fU, 0x5c2e2e72U,
 176         0x381c1c24U, 0x57a6a6f1U, 0x73b4b4c7U, 0x97c6c651U,
 177         0xcbe8e823U, 0xa1dddd7cU, 0xe874749cU, 0x3e1f1f21U,
 178         0x964b4bddU, 0x61bdbddcU, 0x0d8b8b86U, 0x0f8a8a85U,
 179         0xe0707090U, 0x7c3e3e42U, 0x71b5b5c4U, 0xcc6666aaU,
 180         0x904848d8U, 0x06030305U, 0xf7f6f601U, 0x1c0e0e12U,
 181         0xc26161a3U, 0x6a35355fU, 0xae5757f9U, 0x69b9b9d0U,
 182         0x17868691U, 0x99c1c158U, 0x3a1d1d27U, 0x279e9eb9U,
 183         0xd9e1e138U, 0xebf8f813U, 0x2b9898b3U, 0x22111133U,
 184         0xd26969bbU, 0xa9d9d970U, 0x078e8e89U, 0x339494a7U,
 185         0x2d9b9bb6U, 0x3c1e1e22U, 0x15878792U, 0xc9e9e920U,
 186         0x87cece49U, 0xaa5555ffU, 0x50282878U, 0xa5dfdf7aU,
 187         0x038c8c8fU, 0x59a1a1f8U, 0x09898980U, 0x1a0d0d17U,
 188         0x65bfbfdaU, 0xd7e6e631U, 0x844242c6U, 0xd06868b8U,
 189         0x824141c3U, 0x299999b0U, 0x5a2d2d77U, 0x1e0f0f11U,
```



```
 190         0x7bb0b0cbU, 0xa85454fcU, 0x6dbbbbd6U, 0x2c16163aU
 191 };
_____unchanged_portion_omitted_
1427 #endif  /* sun4u */
1428 /* EXPORT DELETE END */


1431 /*
1432  * Initialize key schedules for AES
1433  *
1434  * Parameters:
1435  * cipherKey    User key
1436  * keyBits      AES key size (128, 192, or 256 bits)
1437  * keysched     AES key schedule to be initialized, of type aes_key_t.
1438  *              Allocated by aes_alloc_keysched().
1439  */
1440 void
1441 aes_init_keysched(const uint8_t *cipherKey, uint_t keyBits, void *keysched)
1442 {
1443 /* EXPORT DELETE START */
1444         aes_key_t       *newbie = keysched;
1445         uint_t          keysize, i, j;
1446         union {
1447                 uint64_t        ka64[4];
1448                 uint32_t        ka32[8];
1449                 } keyarr;

1451         switch (keyBits) {
1452         case 128:
1453                 newbie->nr = 10;
1454                 break;

1456         case 192:
1457                 newbie->nr = 12;
1458                 break;

1460         case 256:
1461                 newbie->nr = 14;
1462                 break;

1464         default:
1465                 /* should never get here */
1466                 return;
1467         }
1468         keysize = keyBits >> 3;

1470         /*
1471          * For _LITTLE_ENDIAN machines (except AMD64), reverse every
1472          * 4 bytes in the key.  On _BIG_ENDIAN and AMD64, copy the key
1473          * without reversing bytes.
1474          * For AMD64, do not byte swap for aes_setupkeys().
1475          *
1476          * SPARCv8/v9 uses a key schedule array with 64-bit elements.
1477          * X86/AMD64  uses a key schedule array with 32-bit elements.
1478          */
1479 #ifndef AES_BYTE_SWAP
1480         if (IS_P2ALIGNED(cipherKey, sizeof (uint64_t))) {
1481                 for (i = 0, j = 0; j < keysize; i++, j += 8) {
1482                         /* LINTED: pointer alignment */
1483                         keyarr.ka64[i] = *((uint64_t *)&cipherKey[j]);
1484                 }
1485         } else {
1486                 bcopy(cipherKey, keyarr.ka32, keysize);
1487         }

1489 #else   /* byte swap */
```

```
1490         for (i = 0, j = 0; j < keysize; i++, j += 4) {
1491                 keyarr.ka32[i] = htonl(*(uint32_t *)&cipherKey[j]);
1493                 keyarr.ka32[i] = (((uint32_t)cipherKey[j] << 24) |
1494                     ((uint32_t)cipherKey[j + 1] << 16) |
1495                     ((uint32_t)cipherKey[j + 2] << 8) |
1496                     (uint32_t)cipherKey[j + 3]);
1492         }
1493 #endif

1495         aes_setupkeys(newbie, keyarr.ka32, keyBits);
1496 /* EXPORT DELETE END */
1497 }

1499 /*
1500  * Encrypt one block using AES.
1501  * Align if needed and (for x86 32-bit only) byte-swap.
1502  *
1503  * Parameters:
1504  * ks   Key schedule, of type aes_key_t
1505  * pt   Input block (plain text)
1506  * ct   Output block (crypto text).  Can overlap with pt
1507  */
1508 int
1509 aes_encrypt_block(const void *ks, const uint8_t *pt, uint8_t *ct)
1510 {
1511 /* EXPORT DELETE START */
1512         aes_key_t       *ksch = (aes_key_t *)ks;

1514 #ifndef AES_BYTE_SWAP
1515         if (IS_P2ALIGNED2(pt, ct, sizeof (uint32_t))) {
1516                 AES_ENCRYPT_IMPL(&ksch->encr_ks.ks32[0], ksch->nr,
1517                     /* LINTED:  pointer alignment */
1518                     (uint32_t *)pt, (uint32_t *)ct);
1519         } else {
1520 #endif
1521                 uint32_t buffer[AES_BLOCK_LEN / sizeof (uint32_t)];

1523                 /* Copy input block into buffer */
1524 #ifndef AES_BYTE_SWAP
1525                 bcopy(pt, &buffer, AES_BLOCK_LEN);

1527 #else   /* byte swap */
1528                 buffer[0] = htonl(*(uint32_t *)&pt[0]);
1529                 buffer[1] = htonl(*(uint32_t *)&pt[4]);
1530                 buffer[2] = htonl(*(uint32_t *)&pt[8]);
1531                 buffer[3] = htonl(*(uint32_t *)&pt[12]);
1533                 buffer[0] = (((uint32_t)pt[0] << 24) | ((uint32_t)pt[1] << 16) |
1534                     ((uint32_t)pt[2] << 8) | (uint32_t)pt[3]);
1535                 buffer[1] = (((uint32_t)pt[4] << 24) | ((uint32_t)pt[5] << 16) |
1536                     ((uint32_t)pt[6] << 8) | (uint32_t)pt[7]);
1537                 buffer[2] = (((uint32_t)pt[8] << 24) | ((uint32_t)pt[9] << 16) |
1538                     ((uint32_t)pt[10] << 8) | (uint32_t)pt[11]);
1539                 buffer[3] = (((uint32_t)pt[12] << 24) |
1540                     ((uint32_t)pt[13] << 16) | ((uint32_t)pt[14] << 8) |
1541                     (uint32_t)pt[15]);
1532 #endif

1534                 AES_ENCRYPT_IMPL(&ksch->encr_ks.ks32[0], ksch->nr,
1535                     buffer, buffer);

1537                 /* Copy result from buffer to output block */
1538 #ifndef AES_BYTE_SWAP
1539                 bcopy(&buffer, ct, AES_BLOCK_LEN);
1540         }

1542 #else   /* byte swap */
```



```
1543                 *(uint32_t *)&ct[0] = htonl(buffer[0]);
1544                 *(uint32_t *)&ct[4] = htonl(buffer[1]);
1545                 *(uint32_t *)&ct[8] = htonl(buffer[2]);
1546                 *(uint32_t *)&ct[12] = htonl(buffer[3]);
1553                 ct[0] = buffer[0] >> 24;
1554                 ct[1] = buffer[0] >> 16;
1555                 ct[2] = buffer[0] >> 8;
1556                 ct[3] = (uint8_t)buffer[0];
1557                 ct[4] = buffer[1] >> 24;
1558                 ct[5] = buffer[1] >> 16;
1559                 ct[6] = buffer[1] >> 8;
1560                 ct[7] = (uint8_t)buffer[1];
1561                 ct[8] = buffer[2] >> 24;
1562                 ct[9] = buffer[2] >> 16;
1563                 ct[10] = buffer[2] >> 8;
1564                 ct[11] = (uint8_t)buffer[2];
1565                 ct[12] = buffer[3] >> 24;
1566                 ct[13] = buffer[3] >> 16;
1567                 ct[14] = buffer[3] >> 8;
1568                 ct[15] = (uint8_t)buffer[3];
1547 #endif
1548 /* EXPORT DELETE END */
1549         return (CRYPTO_SUCCESS);
1550 }

1552 /*
1553  * Decrypt one block using AES.
1554  * Align and byte-swap if needed.
1555  *
1556  * Parameters:
1557  * ks   Key schedule, of type aes_key_t
1558  * ct   Input block (crypto text)
1559  * pt   Output block (plain text). Can overlap with pt
1560  */
1561 int
1562 aes_decrypt_block(const void *ks, const uint8_t *ct, uint8_t *pt)
1563 {
1564 /* EXPORT DELETE START */
1565         aes_key_t       *ksch = (aes_key_t *)ks;

1567 #ifndef AES_BYTE_SWAP
1568         if (IS_P2ALIGNED2(ct, pt, sizeof (uint32_t))) {
1569                 AES_DECRYPT_IMPL(&ksch->decr_ks.ks32[0], ksch->nr,
1570                     /* LINTED:  pointer alignment */
1571                     (uint32_t *)ct, (uint32_t *)pt);
1572         } else {
1573 #endif
1574                 uint32_t buffer[AES_BLOCK_LEN / sizeof (uint32_t)];

1576                 /* Copy input block into buffer */
1577 #ifndef AES_BYTE_SWAP
1578                 bcopy(ct, &buffer, AES_BLOCK_LEN);

1580 #else   /* byte swap */
1581                 buffer[0] = htonl(*(uint32_t *)&ct[0]);
1582                 buffer[1] = htonl(*(uint32_t *)&ct[4]);
1583                 buffer[2] = htonl(*(uint32_t *)&ct[8]);
1584                 buffer[3] = htonl(*(uint32_t *)&ct[12]);
1603                 buffer[0] = (((uint32_t)ct[0] << 24) | ((uint32_t)ct[1] << 16) |
1604                     ((uint32_t)ct[2] << 8) | (uint32_t)ct[3]);

1606                 buffer[1] = (((uint32_t)ct[4] << 24) | ((uint32_t)ct[5] << 16) |
1607                     ((uint32_t)ct[6] << 8) | (uint32_t)ct[7]);

1609                 buffer[2] = (((uint32_t)ct[8] << 24) | ((uint32_t)ct[9] << 16) |
1610                     ((uint32_t)ct[10] << 8) | (uint32_t)ct[11]);
```

```
1612                 buffer[3] = (((uint32_t)ct[12] << 24) |
1613                     ((uint32_t)ct[13] << 16) | ((uint32_t)ct[14] << 8) |
1614                     (uint32_t)ct[15]);
1585 #endif

1587                 AES_DECRYPT_IMPL(&ksch->decr_ks.ks32[0], ksch->nr,
1588                     buffer, buffer);

1590                 /* Copy result from buffer to output block */
1591 #ifndef AES_BYTE_SWAP
1592                 bcopy(&buffer, pt, AES_BLOCK_LEN);
1593         }

1595 #else   /* byte swap */
1596         *(uint32_t *)&pt[0] = htonl(buffer[0]);
1597         *(uint32_t *)&pt[4] = htonl(buffer[1]);
1598         *(uint32_t *)&pt[8] = htonl(buffer[2]);
1599         *(uint32_t *)&pt[12] = htonl(buffer[3]);
1626                 pt[0] = buffer[0] >> 24;
1627                 pt[1] = buffer[0] >> 16;
1628                 pt[2] = buffer[0] >> 8;
1629                 pt[3] = (uint8_t)buffer[0];
1630                 pt[4] = buffer[1] >> 24;
1631                 pt[5] = buffer[1] >> 16;
1632                 pt[6] = buffer[1] >> 8;
1633                 pt[7] = (uint8_t)buffer[1];
1634                 pt[8] = buffer[2] >> 24;
1635                 pt[9] = buffer[2] >> 16;
1636                 pt[10] = buffer[2] >> 8;
1637                 pt[11] = (uint8_t)buffer[2];
1638                 pt[12] = buffer[3] >> 24;
1639                 pt[13] = buffer[3] >> 16;
1640                 pt[14] = buffer[3] >> 8;
1641                 pt[15] = (uint8_t)buffer[3];
1600 #endif

1602 /* EXPORT DELETE END */
1603         return (CRYPTO_SUCCESS);
1604 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   24462 Wed Aug 27 14:22:22 2008
new/usr/src/common/crypto/aes/amd64/aesopt.h
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * ---------------------------------------------------------------------------
 * Copyright (c) 1998-2007, Brian Gladman, Worcester, UK. All rights reserved.
 *
 * LICENSE TERMS
 *
 * The free distribution and use of this software is allowed (with or without
 * changes) provided that:
 *
 *  1. source code distributions include the above copyright notice, this
 *      list of conditions and the following disclaimer;
 *
 *  2. binary distributions include the above copyright notice, this list
 *      of conditions and the following disclaimer in their documentation;
 *
 *  3. the name of the copyright holder is not used to endorse products
 *      built using this software without specific written permission.
 *
 * DISCLAIMER
 *
 * This software is provided 'as is' with no explicit or implied warranties
 * in respect of its properties, including, but not limited to, correctness
 * and/or fitness for purpose.
 * ---------------------------------------------------------------------------
 * Issue Date: 20/12/2007
 *
 * This file contains the compilation options for AES (Rijndael) and code
 * that is common across encryption, key scheduling and table generation.
 *
 * OPERATION
 *
 * These source code files implement the AES algorithm Rijndael designed by
 * Joan Daemen and Vincent Rijmen. This version is designed for the standard
 * block size of 16 bytes and for key sizes of 128, 192 and 256 bits (16, 24
 * and 32 bytes).
 *
 * This version is designed for flexibility and speed using operations on
 * 32-bit words rather than operations on bytes.  It can be compiled with
 * either big or little endian internal byte order but is faster when the
 * native byte order for the processor is used.
 *
 * THE CIPHER INTERFACE
 *
 * The cipher interface is implemented as an array of bytes in which lower
 * AES bit sequence indexes map to higher numeric significance within bytes.
 */

/*
 * OpenSolaris changes
 * 1. Added __cplusplus and _AESTAB_H header guards
 * 2. Added header files sys/types.h and aes_impl.h
 * 3. Added defines for AES_ENCRYPT, AES_DECRYPT, AES_REV_DKS, and ASM_AMD64_C
 * 4. Moved defines for IS_BIG_ENDIAN, IS_LITTLE_ENDIAN, PLATFORM_BYTE_ORDER
 *    from brg_endian.h
 * 5. Undefined VIA_ACE_POSSIBLE and ASSUME_VIA_ACE_PRESENT
 * 6. Changed uint_8t and uint_32t to uint8_t and uint32_t
 * 7. Defined aes_sw32 as htonl() for byte swapping
 * 8. Cstyled and hdrchk code
 * 7. cstyled and hdrchk code
```



```
 *
 */

#ifndef _AESOPT_H
#define _AESOPT_H

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

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <sys/byteorder.h>
#include <aes_impl.h>

/*  SUPPORT FEATURES */
#define AES_ENCRYPT /* if support for encryption is needed */
#define AES_DECRYPT /* if support for decryption is needed */

/*  PLATFORM-SPECIFIC FEATURES */
#define IS_BIG_ENDIAN           4321 /* byte 0 is most significant (mc68k) */
#define IS_LITTLE_ENDIAN        1234 /* byte 0 is least significant (i386) */
#define PLATFORM_BYTE_ORDER     IS_LITTLE_ENDIAN
#define AES_REV_DKS /* define to reverse decryption key schedule */


/*
 *  CONFIGURATION - THE USE OF DEFINES
 *      Later in this section there are a number of defines that control the
 *      operation of the code.  In each section, the purpose of each define is
 *      explained so that the relevant form can be included or excluded by
 *      setting either 1's or 0's respectively on the branches of the related
 *      #if clauses.  The following local defines should not be changed.
 */

#define ENCRYPTION_IN_C 1
#define DECRYPTION_IN_C 2
#define ENC_KEYING_IN_C 4
#define DEC_KEYING_IN_C 8

#define NO_TABLES       0
#define ONE_TABLE       1
#define FOUR_TABLES     4
#define NONE            0
#define PARTIAL         1
#define FULL            2

/*  --- START OF USER CONFIGURED OPTIONS --- */

/*
 *  1. BYTE ORDER WITHIN 32 BIT WORDS
 *
 *      The fundamental data processing units in Rijndael are 8-bit bytes. The
 *      input, output and key input are all enumerated arrays of bytes in which
 *      bytes are numbered starting at zero and increasing to one less than the
 *      number of bytes in the array in question. This enumeration is only used
 *      for naming bytes and does not imply any adjacency or order relationship
 *      from one byte to another. When these inputs and outputs are considered
 *      as bit sequences, bits 8*n to 8*n+7 of the bit sequence are mapped to
 *      byte[n] with bit 8n+i in the sequence mapped to bit 7-i within the byte.
 *      In this implementation bits are numbered from 0 to 7 starting at the
 *      numerically least significant end of each byte.  Bit n represents 2^n.
 *
 *      However, Rijndael can be implemented more efficiently using 32-bit
 *      words by packing bytes into words so that bytes 4*n to 4*n+3 are placed
```

```
 *      into word[n]. While in principle these bytes can be assembled into words
 *      in any positions, this implementation only supports the two formats in
 *      which bytes in adjacent positions within words also have adjacent byte
 *      numbers. This order is called big-endian if the lowest numbered bytes
 *      in words have the highest numeric significance and little-endian if the
 *      opposite applies.
 *
 *      This code can work in either order irrespective of the order used by the
 *      machine on which it runs. Normally the internal byte order will be set
 *      to the order of the processor on which the code is to be run but this
 *      define  can be used to reverse this in special situations
 *
 *      WARNING: Assembler code versions rely on PLATFORM_BYTE_ORDER being set.
 *      This define will hence be redefined later (in section 4) if necessary
 */

#if 1
#define ALGORITHM_BYTE_ORDER PLATFORM_BYTE_ORDER
#elif 0
#define ALGORITHM_BYTE_ORDER IS_LITTLE_ENDIAN
#elif 0
#define ALGORITHM_BYTE_ORDER IS_BIG_ENDIAN
#else
#error The algorithm byte order is not defined
#endif

/*  2. VIA ACE SUPPORT */

#if defined(__GNUC__) && defined(__i386__) || \
        defined(_WIN32) && defined(_M_IX86) && \
        !(defined(_WIN64) || defined(_WIN32_WCE) || \
        defined(_MSC_VER) && (_MSC_VER <= 800))
#define VIA_ACE_POSSIBLE
#endif

/*
 *  Define this option if support for the VIA ACE is required. This uses
 *  inline assembler instructions and is only implemented for the Microsoft,
 *  Intel and GCC compilers.  If VIA ACE is known to be present, then defining
 *  ASSUME_VIA_ACE_PRESENT will remove the ordinary encryption/decryption
 *  code.  If USE_VIA_ACE_IF_PRESENT is defined then VIA ACE will be used if
 *  it is detected (both present and enabled) but the normal AES code will
 *  also be present.
 *
 *  When VIA ACE is to be used, all AES encryption contexts MUST be 16 byte
 *  aligned; other input/output buffers do not need to be 16 byte aligned
 *  but there are very large performance gains if this can be arranged.
 *  VIA ACE also requires the decryption key schedule to be in reverse
 *  order (which later checks below ensure).
 */

/*  VIA ACE is not used here for OpenSolaris: */
#undef  VIA_ACE_POSSIBLE
#undef  ASSUME_VIA_ACE_PRESENT

#if 0 && defined(VIA_ACE_POSSIBLE) && !defined(USE_VIA_ACE_IF_PRESENT)
#define USE_VIA_ACE_IF_PRESENT
#endif

#if 0 && defined(VIA_ACE_POSSIBLE) && !defined(ASSUME_VIA_ACE_PRESENT)
#define ASSUME_VIA_ACE_PRESENT
#endif


/*
 *  3. ASSEMBLER SUPPORT
```



```
 *
 *      This define (which can be on the command line) enables the use of the
 *      assembler code routines for encryption, decryption and key scheduling
 *      as follows:
 *
 *      ASM_X86_V1C uses the assembler (aes_x86_v1.asm) with large tables for
 *              encryption and decryption and but with key scheduling in C
 *      ASM_X86_V2  uses assembler (aes_x86_v2.asm) with compressed tables for
 *              encryption, decryption and key scheduling
 *      ASM_X86_V2C uses assembler (aes_x86_v2.asm) with compressed tables for
 *              encryption and decryption and but with key scheduling in C
 *      ASM_AMD64_C uses assembler (aes_amd64.asm) with compressed tables for
 *              encryption and decryption and but with key scheduling in C
 *
 *      Change one 'if 0' below to 'if 1' to select the version or define
 *      as a compilation option.
 */

#if 0 && !defined(ASM_X86_V1C)
#define ASM_X86_V1C
#elif 0 && !defined(ASM_X86_V2)
#define ASM_X86_V2
#elif 0 && !defined(ASM_X86_V2C)
#define ASM_X86_V2C
#elif 1 && !defined(ASM_AMD64_C)
#define ASM_AMD64_C
#endif

#if (defined(ASM_X86_V1C) || defined(ASM_X86_V2) || defined(ASM_X86_V2C)) && \
        !defined(_M_IX86) || defined(ASM_AMD64_C) && !defined(_M_X64) && \
        !defined(__amd64)
#error Assembler code is only available for x86 and AMD64 systems
#endif

/*
 *  4. FAST INPUT/OUTPUT OPERATIONS.
 *
 *      On some machines it is possible to improve speed by transferring the
 *      bytes in the input and output arrays to and from the internal 32-bit
 *      variables by addressing these arrays as if they are arrays of 32-bit
 *      words.  On some machines this will always be possible but there may
 *      be a large performance penalty if the byte arrays are not aligned on
 *      the normal word boundaries. On other machines this technique will
 *      lead to memory access errors when such 32-bit word accesses are not
 *      properly aligned. The option SAFE_IO avoids such problems but will
 *      often be slower on those machines that support misaligned access
 *      (especially so if care is taken to align the input  and output byte
 *      arrays on 32-bit word boundaries). If SAFE_IO is not defined it is
 *      assumed that access to byte arrays as if they are arrays of 32-bit
 *      words will not cause problems when such accesses are misaligned.
 */
#if 1 && !defined(_MSC_VER)
#define SAFE_IO
#endif

/*
 *  5. LOOP UNROLLING
 *
 *      The code for encryption and decryption cycles through a number of rounds
 *      that can be implemented either in a loop or by expanding the code into a
 *      long sequence of instructions, the latter producing a larger program but
 *      one that will often be much faster. The latter is called loop unrolling.
 *      There are also potential speed advantages in expanding two iterations in
 *      a loop with half the number of iterations, which is called partial loop
 *      unrolling.  The following options allow partial or full loop unrolling
 *      to be set independently for encryption and decryption
```

```
 */
#if 1
#define ENC_UNROLL  FULL
#elif 0
#define ENC_UNROLL  PARTIAL
#else
#define ENC_UNROLL  NONE
#endif

#if 1
#define DEC_UNROLL  FULL
#elif 0
#define DEC_UNROLL  PARTIAL
#else
#define DEC_UNROLL  NONE
#endif

#if 1
#define ENC_KS_UNROLL
#endif

#if 1
#define DEC_KS_UNROLL
#endif

/*
 *  6. FAST FINITE FIELD OPERATIONS
 *
 *      If this section is included, tables are used to provide faster finite
 *      field arithmetic.  This has no effect if FIXED_TABLES is defined.
 */
#if 1
#define FF_TABLES
#endif

/*
 *  7. INTERNAL STATE VARIABLE FORMAT
 *
 *      The internal state of Rijndael is stored in a number of local 32-bit
 *      word variables which can be defined either as an array or as individual
 *      names variables. Include this section if you want to store these local
 *      variables in arrays. Otherwise individual local variables will be used.
 */
#if 1
#define ARRAYS
#endif

/*
 *  8. FIXED OR DYNAMIC TABLES
 *
 *      When this section is included the tables used by the code are compiled
 *      statically into the binary file.  Otherwise the subroutine aes_init()
 *      must be called to compute them before the code is first used.
 */
#if 1 && !(defined(_MSC_VER) && (_MSC_VER <= 800))
#define FIXED_TABLES
#endif

/*
 *  9. MASKING OR CASTING FROM LONGER VALUES TO BYTES
 *
 *      In some systems it is better to mask longer values to extract bytes
 *      rather than using a cast. This option allows this choice.
 */
#if 0
#define to_byte(x)  ((uint8_t)(x))
```



```
#else
#define to_byte(x)  ((x) & 0xff)
#endif

/*
 *  10. TABLE ALIGNMENT
 *
 *      On some systems speed will be improved by aligning the AES large lookup
 *      tables on particular boundaries. This define should be set to a power of
 *      two giving the desired alignment. It can be left undefined if alignment
 *      is not needed.  This option is specific to the Micrsoft VC++ compiler -
 *      it seems to sometimes cause trouble for the VC++ version 6 compiler.
 */

#if 1 && defined(_MSC_VER) && (_MSC_VER >= 1300)
#define TABLE_ALIGN 32
#endif

/*
 *  11.  REDUCE CODE AND TABLE SIZE
 *
 *      This replaces some expanded macros with function calls if AES_ASM_V2 or
 *      AES_ASM_V2C are defined
 */

#if 1 && (defined(ASM_X86_V2) || defined(ASM_X86_V2C))
#define REDUCE_CODE_SIZE
#endif

/*
 *  12. TABLE OPTIONS
 *
 *      This cipher proceeds by repeating in a number of cycles known as rounds
 *      which are implemented by a round function which is optionally be speeded
 *      up using tables.  The basic tables are 256 32-bit words, with either
 *      one or four tables being required for each round function depending on
 *      how much speed is required. Encryption and decryption round functions
 *      are different and the last encryption and decryption round functions are
 *      different again making four different round functions in all.
 *
 *      This means that:
 *      1. Normal encryption and decryption rounds can each use either 0, 1
 *              or 4 tables and table spaces of 0, 1024 or 4096 bytes each.
 *      2. The last encryption and decryption rounds can also use either 0, 1
 *              or 4 tables and table spaces of 0, 1024 or 4096 bytes each.
 *
 *      Include or exclude the appropriate definitions below to set the number
 *      of tables used by this implementation.
 */

#if 1   /* set tables for the normal encryption round */
#define ENC_ROUND   FOUR_TABLES
#elif 0
#define ENC_ROUND   ONE_TABLE
#else
#define ENC_ROUND   NO_TABLES
#endif

#if 1   /* set tables for the last encryption round */
#define LAST_ENC_ROUND  FOUR_TABLES
#elif 0
#define LAST_ENC_ROUND  ONE_TABLE
#else
#define LAST_ENC_ROUND  NO_TABLES
#endif
```

```
#if 1   /* set tables for the normal decryption round */
#define DEC_ROUND   FOUR_TABLES
#elif 0
#define DEC_ROUND   ONE_TABLE
#else
#define DEC_ROUND   NO_TABLES
#endif

#if 1   /* set tables for the last decryption round */
#define LAST_DEC_ROUND  FOUR_TABLES
#elif 0
#define LAST_DEC_ROUND  ONE_TABLE
#else
#define LAST_DEC_ROUND  NO_TABLES
#endif

/*
 *  The decryption key schedule can be speeded up with tables in the same
 *      way that the round functions can.  Include or exclude the following
 *      defines to set this requirement.
 */
#if 1
#define KEY_SCHED   FOUR_TABLES
#elif 0
#define KEY_SCHED   ONE_TABLE
#else
#define KEY_SCHED   NO_TABLES
#endif

/*  ---- END OF USER CONFIGURED OPTIONS ---- */

/* VIA ACE support is only available for VC++ and GCC */

#if !defined(_MSC_VER) && !defined(__GNUC__)
#if defined(ASSUME_VIA_ACE_PRESENT)
#undef ASSUME_VIA_ACE_PRESENT
#endif
#if defined(USE_VIA_ACE_IF_PRESENT)
#undef USE_VIA_ACE_IF_PRESENT
#endif
#endif

#if defined(ASSUME_VIA_ACE_PRESENT) && !defined(USE_VIA_ACE_IF_PRESENT)
#define USE_VIA_ACE_IF_PRESENT
#endif

#if defined(USE_VIA_ACE_IF_PRESENT) && !defined(AES_REV_DKS)
#define AES_REV_DKS
#endif

/* Assembler support requires the use of platform byte order */

#if (defined(ASM_X86_V1C) || defined(ASM_X86_V2C) || defined(ASM_AMD64_C)) && \
        (ALGORITHM_BYTE_ORDER != PLATFORM_BYTE_ORDER)
#undef  ALGORITHM_BYTE_ORDER
#define ALGORITHM_BYTE_ORDER PLATFORM_BYTE_ORDER
#endif

/*
 * In this implementation the columns of the state array are each held in
 *      32-bit words. The state array can be held in various ways: in an array
 *      of words, in a number of individual word variables or in a number of
 *      processor registers. The following define maps a variable name x and
 *      a column number c to the way the state array variable is to be held.
 *      The first define below maps the state into an array x[c] whereas the
 *      second form maps the state into a number of individual variables x0,
```



```
 *      x1, etc.  Another form could map individual state columns to machine
 *      register names.
 */

#if defined(ARRAYS)
#define s(x, c) x[c]
#else
#define s(x, c) x##c
#endif

/*
 *  This implementation provides subroutines for encryption, decryption
 *      and for setting the three key lengths (separately) for encryption
 *      and decryption. Since not all functions are needed, masks are set
 *      up here to determine which will be implemented in C
 */

#if !defined(AES_ENCRYPT)
#define EFUNCS_IN_C   0
#elif defined(ASSUME_VIA_ACE_PRESENT) || defined(ASM_X86_V1C) || \
        defined(ASM_X86_V2C) || defined(ASM_AMD64_C)
#define EFUNCS_IN_C   ENC_KEYING_IN_C
#elif !defined(ASM_X86_V2)
#define EFUNCS_IN_C   (ENCRYPTION_IN_C | ENC_KEYING_IN_C)
#else
#define EFUNCS_IN_C   0
#endif

#if !defined(AES_DECRYPT)
#define DFUNCS_IN_C   0
#elif defined(ASSUME_VIA_ACE_PRESENT) || defined(ASM_X86_V1C) || \
        defined(ASM_X86_V2C) || defined(ASM_AMD64_C)
#define DFUNCS_IN_C   DEC_KEYING_IN_C
#elif !defined(ASM_X86_V2)
#define DFUNCS_IN_C   (DECRYPTION_IN_C | DEC_KEYING_IN_C)
#else
#define DFUNCS_IN_C   0
#endif

#define FUNCS_IN_C  (EFUNCS_IN_C | DFUNCS_IN_C)

/* END OF CONFIGURATION OPTIONS */

/* Disable or report errors on some combinations of options */

#if ENC_ROUND == NO_TABLES && LAST_ENC_ROUND != NO_TABLES
#undef  LAST_ENC_ROUND
#define LAST_ENC_ROUND  NO_TABLES
#elif ENC_ROUND == ONE_TABLE && LAST_ENC_ROUND == FOUR_TABLES
#undef  LAST_ENC_ROUND
#define LAST_ENC_ROUND  ONE_TABLE
#endif

#if ENC_ROUND == NO_TABLES && ENC_UNROLL != NONE
#undef  ENC_UNROLL
#define ENC_UNROLL  NONE
#endif

#if DEC_ROUND == NO_TABLES && LAST_DEC_ROUND != NO_TABLES
#undef  LAST_DEC_ROUND
#define LAST_DEC_ROUND  NO_TABLES
#elif DEC_ROUND == ONE_TABLE && LAST_DEC_ROUND == FOUR_TABLES
#undef  LAST_DEC_ROUND
#define LAST_DEC_ROUND  ONE_TABLE
#endif
```

```
519 #if DEC_ROUND == NO_TABLES && DEC_UNROLL != NONE
520 #undef  DEC_UNROLL
521 #define DEC_UNROLL  NONE
522 #endif

524 #if (ALGORITHM_BYTE_ORDER == IS_LITTLE_ENDIAN)
525 #define aes_sw32        htonl
526 #elif defined(bswap32)
524 #if defined(bswap32)
527 #define aes_sw32        bswap32
528 #elif defined(bswap_32)
529 #define aes_sw32        bswap_32
530 #else
531 #define brot(x, n)  (((uint32_t)(x) << (n)) | ((uint32_t)(x) >> (32 - (n))))
529 #define brot(x, n)   (((uint32_t)(x) <<  n) | ((uint32_t)(x) >> (32 - n)))
532 #define aes_sw32(x) ((brot((x), 8) & 0x00ff00ff) | (brot((x), 24) & 0xff00ff00))
533 #endif


536 /*
537  *      upr(x, n):  rotates bytes within words by n positions, moving bytes to
538  *              higher index positions with wrap around into low positions
539  *      ups(x, n):  moves bytes by n positions to higher index positions in
540  *              words but without wrap around
541  *      bval(x, n): extracts a byte from a word
542  *
543  *      WARNING:   The definitions given here are intended only for use with
544  *              unsigned variables and with shift counts that are compile
545  *              time constants
546  */

548 #if (ALGORITHM_BYTE_ORDER == IS_LITTLE_ENDIAN)
549 #define upr(x, n)       (((uint32_t)(x) << (8 * (n))) | \
550                         ((uint32_t)(x) >> (32 - 8 * (n))))
551 #define ups(x, n)       ((uint32_t)(x) << (8 * (n)))
552 #define bval(x, n)      to_byte((x) >> (8 * (n)))
553 #define bytes2word(b0, b1, b2, b3)  \
554                 (((uint32_t)(b3) << 24) | ((uint32_t)(b2) << 16) | \
555                 ((uint32_t)(b1) << 8) | (b0))
556 #endif

558 #if (ALGORITHM_BYTE_ORDER == IS_BIG_ENDIAN)
559 #define upr(x, n)       (((uint32_t)(x) >> (8 * (n))) | \
560                         ((uint32_t)(x) << (32 - 8 * (n))))
561 #define ups(x, n)       ((uint32_t)(x) >> (8 * (n)))
562 #define bval(x, n)      to_byte((x) >> (24 - 8 * (n)))
563 #define bytes2word(b0, b1, b2, b3)  \
564                 (((uint32_t)(b0) << 24) | ((uint32_t)(b1) << 16) | \
565                 ((uint32_t)(b2) << 8) | (b3))
566 #endif

568 #if defined(SAFE_IO)
569 #define word_in(x, c)   bytes2word(((const uint8_t *)(x) + 4 * c)[0], \
570                                 ((const uint8_t *)(x) + 4 * c)[1], \
571                                 ((const uint8_t *)(x) + 4 * c)[2], \
572                                 ((const uint8_t *)(x) + 4 * c)[3])
573 #define word_out(x, c, v) { ((uint8_t *)(x) + 4 * c)[0] = bval(v, 0); \
574                         ((uint8_t *)(x) + 4 * c)[1] = bval(v, 1); \
575                         ((uint8_t *)(x) + 4 * c)[2] = bval(v, 2); \
576                         ((uint8_t *)(x) + 4 * c)[3] = bval(v, 3); }
577 #elif (ALGORITHM_BYTE_ORDER == PLATFORM_BYTE_ORDER)
578 #define word_in(x, c)   (*((uint32_t *)(x) + (c)))
579 #define word_out(x, c, v) (*((uint32_t *)(x) + (c)) = (v))
580 #else
581 #define word_in(x, c)   aes_sw32(*((uint32_t *)(x) + (c)))
582 #define word_out(x, c, v) (*((uint32_t *)(x) + (c)) = aes_sw32(v))
```



```
583 #endif

585 /* the finite field modular polynomial and elements */

587 #define WPOLY   0x011b
588 #define BPOLY   0x1b

590 /* multiply four bytes in GF(2^8) by 'x' {02} in parallel */

592 #define m1  0x80808080
593 #define m2  0x7f7f7f7f
594 #define gf_mulx(x)  ((((x) & m2) << 1) ^ ((((x) & m1) >> 7) * BPOLY))

596 /*
597  * The following defines provide alternative definitions of gf_mulx that might
598  * give improved performance if a fast 32-bit multiply is not available. Note
599  * that a temporary variable u needs to be defined where gf_mulx is used.
600  *
601  * #define      gf_mulx(x) (u = (x) & m1, u |= (u >> 1), ((x) & m2) << 1) ^ \
602  *                         ((u >> 3) | (u >> 6))
603  * #define      m4  (0x01010101 * BPOLY)
604  * #define      gf_mulx(x) (u = (x) & m1, ((x) & m2) << 1) ^ ((u - (u >> 7)) \
605  *                         & m4)
606  */

608 /* Work out which tables are needed for the different options   */

610 #if defined(ASM_X86_V1C)
611 #if defined(ENC_ROUND)
612 #undef  ENC_ROUND
613 #endif
614 #define ENC_ROUND   FOUR_TABLES
615 #if defined(LAST_ENC_ROUND)
616 #undef  LAST_ENC_ROUND
617 #endif
618 #define LAST_ENC_ROUND  FOUR_TABLES
619 #if defined(DEC_ROUND)
620 #undef  DEC_ROUND
621 #endif
622 #define DEC_ROUND   FOUR_TABLES
623 #if defined(LAST_DEC_ROUND)
624 #undef  LAST_DEC_ROUND
625 #endif
626 #define LAST_DEC_ROUND  FOUR_TABLES
627 #if defined(KEY_SCHED)
628 #undef  KEY_SCHED
629 #define KEY_SCHED   FOUR_TABLES
630 #endif
631 #endif

633 #if (FUNCS_IN_C & ENCRYPTION_IN_C) || defined(ASM_X86_V1C)
634 #if ENC_ROUND == ONE_TABLE
635 #define FT1_SET
636 #elif ENC_ROUND == FOUR_TABLES
637 #define FT4_SET
638 #else
639 #define SBX_SET
640 #endif
641 #if LAST_ENC_ROUND == ONE_TABLE
642 #define FL1_SET
643 #elif LAST_ENC_ROUND == FOUR_TABLES
644 #define FL4_SET
645 #elif !defined(SBX_SET)
646 #define SBX_SET
647 #endif
648 #endif
```

```
650 #if (FUNCS_IN_C & DECRYPTION_IN_C) || defined(ASM_X86_V1C)
651 #if DEC_ROUND == ONE_TABLE
652 #define IT1_SET
653 #elif DEC_ROUND == FOUR_TABLES
654 #define IT4_SET
655 #else
656 #define ISB_SET
657 #endif
658 #if LAST_DEC_ROUND == ONE_TABLE
659 #define IL1_SET
660 #elif LAST_DEC_ROUND == FOUR_TABLES
661 #define IL4_SET
662 #elif !defined(ISB_SET)
663 #define ISB_SET
664 #endif
665 #endif


668 #if !(defined(REDUCE_CODE_SIZE) && (defined(ASM_X86_V2) || \
669         defined(ASM_X86_V2C)))
670 #if ((FUNCS_IN_C & ENC_KEYING_IN_C) || (FUNCS_IN_C & DEC_KEYING_IN_C))
671 #if KEY_SCHED == ONE_TABLE
672 #if !defined(FL1_SET) && !defined(FL4_SET)
673 #define LS1_SET
674 #endif
675 #elif KEY_SCHED == FOUR_TABLES
676 #if !defined(FL4_SET)
677 #define LS4_SET
678 #endif
679 #elif !defined(SBX_SET)
680 #define SBX_SET
681 #endif
682 #endif
683 #if (FUNCS_IN_C & DEC_KEYING_IN_C)
684 #if KEY_SCHED == ONE_TABLE
685 #define IM1_SET
686 #elif KEY_SCHED == FOUR_TABLES
687 #define IM4_SET
688 #elif !defined(SBX_SET)
689 #define SBX_SET
690 #endif
691 #endif
692 #endif

694 /* generic definitions of Rijndael macros that use tables */

696 #define no_table(x, box, vf, rf, c) bytes2word(\
697         box[bval(vf(x, 0, c), rf(0, c))], \
698         box[bval(vf(x, 1, c), rf(1, c))], \
699         box[bval(vf(x, 2, c), rf(2, c))], \
700         box[bval(vf(x, 3, c), rf(3, c))])

702 #define one_table(x, op, tab, vf, rf, c) \
703         (tab[bval(vf(x, 0, c), rf(0, c))] \
704         ^ op(tab[bval(vf(x, 1, c), rf(1, c))], 1) \
705         ^ op(tab[bval(vf(x, 2, c), rf(2, c))], 2) \
706         ^ op(tab[bval(vf(x, 3, c), rf(3, c))], 3))

708 #define four_tables(x, tab, vf, rf, c) \
709         (tab[0][bval(vf(x, 0, c), rf(0, c))] \
710         ^ tab[1][bval(vf(x, 1, c), rf(1, c))] \
711         ^ tab[2][bval(vf(x, 2, c), rf(2, c))] \
712         ^ tab[3][bval(vf(x, 3, c), rf(3, c))])

714 #define vf1(x, r, c)    (x)
```



```
715 #define rf1(r, c)       (r)
716 #define rf2(r, c)       ((8+r-c)&3)

718 /*
719  * Perform forward and inverse column mix operation on four bytes in long word
720  * x in parallel. NOTE: x must be a simple variable, NOT an expression in
721  * these macros.
722  */

724 #if !(defined(REDUCE_CODE_SIZE) && (defined(ASM_X86_V2) || \
725         defined(ASM_X86_V2C)))

727 #if defined(FM4_SET)    /* not currently used */
728 #define fwd_mcol(x)     four_tables(x, t_use(f, m), vf1, rf1, 0)
729 #elif defined(FM1_SET)  /* not currently used */
730 #define fwd_mcol(x)     one_table(x, upr, t_use(f, m), vf1, rf1, 0)
731 #else
732 #define dec_fmvars      uint32_t g2
733 #define fwd_mcol(x)     (g2 = gf_mulx(x), g2 ^ upr((x) ^ g2, 3) ^ \
734                                 upr((x), 2) ^ upr((x), 1))
735 #endif

737 #if defined(IM4_SET)
738 #define inv_mcol(x)     four_tables(x, t_use(i, m), vf1, rf1, 0)
739 #elif defined(IM1_SET)
740 #define inv_mcol(x)     one_table(x, upr, t_use(i, m), vf1, rf1, 0)
741 #else
742 #define dec_imvars      uint32_t g2, g4, g9
743 #define inv_mcol(x)     (g2 = gf_mulx(x), g4 = gf_mulx(g2), g9 = \
744                                 (x) ^ gf_mulx(g4), g4 ^= g9, \
745                                 (x) ^ g2 ^ g4 ^ upr(g2 ^ g9, 3) ^ \
746                                 upr(g4, 2) ^ upr(g9, 1))
747 #endif

749 #if defined(FL4_SET)
750 #define ls_box(x, c)    four_tables(x, t_use(f, l), vf1, rf2, c)
751 #elif defined(LS4_SET)
752 #define ls_box(x, c)    four_tables(x, t_use(l, s), vf1, rf2, c)
753 #elif defined(FL1_SET)
754 #define ls_box(x, c)    one_table(x, upr, t_use(f, l), vf1, rf2, c)
755 #elif defined(LS1_SET)
756 #define ls_box(x, c)    one_table(x, upr, t_use(l, s), vf1, rf2, c)
757 #else
758 #define ls_box(x, c)    no_table(x, t_use(s, box), vf1, rf2, c)
759 #endif

761 #endif

763 #if defined(ASM_X86_V1C) && defined(AES_DECRYPT) && !defined(ISB_SET)
764 #define ISB_SET
765 #endif

767 #ifdef  __cplusplus
768 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   26741 Wed Aug 27 14:22:27 2008
new/usr/src/common/crypto/blowfish/blowfish_impl.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * Blowfish encryption/decryption and keyschedule code.
 */

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/ddi.h>
#include <sys/sysmacros.h>
#include <sys/strsun.h>
#include <sys/note.h>
#include <sys/byteorder.h>
#include <sys/crypto/spi.h>
#include <modes/modes.h>
#include <sys/crypto/common.h>
#include "blowfish_impl.h"

#ifdef _KERNEL

#define BLOWFISH_ASSERT(x)      ASSERT(x)

#else /* !_KERNEL */

#include <strings.h>
#include <stdlib.h>
#define BLOWFISH_ASSERT(x)

#endif /* _KERNEL */

#if defined(__i386) || defined(__amd64)
#include <sys/byteorder.h>
#define UNALIGNED_POINTERS_PERMITTED
#endif
```



```
/* EXPORT DELETE START */

/*
 * Blowfish initial P box and S boxes, derived from the hex digits of PI.
 *
 * NOTE:  S boxes are placed into one large array.
 */
static const uint32_t init_P[] = {
        0x243f6a88U, 0x85a308d3U, 0x13198a2eU,
        0x03707344U, 0xa4093822U, 0x299f31d0U,
        0x082efa98U, 0xec4e6c89U, 0x452821e6U,
        0x38d01377U, 0xbe5466cfU, 0x34e90c6cU,
        0xc0ac29b7U, 0xc97c50ddU, 0x3f84d5b5U,
        0xb5470917U, 0x9216d5d9U, 0x8979fb1bU
};
_____unchanged_portion_omitted_

/*
 * Since ROUND() is a macro, make sure that the things inside can be
 * evaluated more than once.  Especially when calling F().
 * Assume the presence of local variables:
 *
 *      uint32_t *P;
 *      uint32_t *S;
 *      uint32_t tmp;
 *
 *
 * And to Microsoft interview survivors out there, perhaps I should do the
 * XOR swap trick, or at least #ifdef (__i386) the tmp = ... = tmp; stuff.
 */

#define F(word) \
        (((S[(word >> 24) & 0xff] + S[256 + ((word >> 16) & 0xff)]) ^ \
                S[512 + ((word >> 8) & 0xff)]) + S[768 + (word & 0xff)])

#define ROUND(left, right, i) \
        (left) ^= P[i]; \
        (right) ^= F((left)); \
        tmp = (left); \
        (left) = (right); \
        (right) = tmp;

/* EXPORT DELETE END */

/*
 * Encrypt a block of data.  Because of addition operations, convert blocks
 * to their big-endian representation, even on Intel boxen.
 */
/* ARGSUSED */
int
blowfish_encrypt_block(const void *cookie, const uint8_t *block,
    uint8_t *out_block)
{
/* EXPORT DELETE START */
        keysched_t *ksch = (keysched_t *)cookie;

        uint32_t left, right, tmp;
        uint32_t *P = ksch->ksch_P;
        uint32_t *S = ksch->ksch_S;
#ifdef _BIG_ENDIAN
        uint32_t *b32;

        if (IS_P2ALIGNED(block, sizeof (uint32_t))) {
                /* LINTED:  pointer alignment */
                b32 = (uint32_t *)block;
```

```
393                 left = b32[0];
394                 right = b32[1];
395         } else
393         } else {
396 #endif
397         {
398         /*
399          * Read input block and place in left/right in big-endian order.
400          */
401 #ifdef UNALIGNED_POINTERS_PERMITTED
402         left = htonl(*(uint32_t *)&block[0]);
403         right = htonl(*(uint32_t *)&block[4]);
404 #else
405         left = ((uint32_t)block[0] << 24)
406             | ((uint32_t)block[1] << 16)
407             | ((uint32_t)block[2] << 8)
408             | (uint32_t)block[3];
409         right = ((uint32_t)block[4] << 24)
410             | ((uint32_t)block[5] << 16)
411             | ((uint32_t)block[6] << 8)
412             | (uint32_t)block[7];
413 #endif  /* UNALIGNED_POINTERS_PERMITTED */
406 #ifdef _BIG_ENDIAN
414         }
408 #endif

416         ROUND(left, right, 0);
417         ROUND(left, right, 1);
418         ROUND(left, right, 2);
419         ROUND(left, right, 3);
420         ROUND(left, right, 4);
421         ROUND(left, right, 5);
422         ROUND(left, right, 6);
423         ROUND(left, right, 7);
424         ROUND(left, right, 8);
425         ROUND(left, right, 9);
426         ROUND(left, right, 10);
427         ROUND(left, right, 11);
428         ROUND(left, right, 12);
429         ROUND(left, right, 13);
430         ROUND(left, right, 14);
431         ROUND(left, right, 15);

433         tmp = left;
434         left = right;
435         right = tmp;
436         right ^= P[16];
437         left ^= P[17];

439 #ifdef _BIG_ENDIAN
440         if (IS_P2ALIGNED(out_block, sizeof (uint32_t))) {
441                 /* LINTED:  pointer alignment */
442                 b32 = (uint32_t *)out_block;
443                 b32[0] = left;
444                 b32[1] = right;
445         } else
439         } else {
446 #endif
447         {
448                 /* Put the block back into the user's block with final swap */
449 #ifdef UNALIGNED_POINTERS_PERMITTED
450                 *(uint32_t *)&out_block[0] = htonl(left);
451                 *(uint32_t *)&out_block[4] = htonl(right);
452 #else
453                 out_block[0] = left >> 24;
454                 out_block[1] = left >> 16;
```



```
455                 out_block[2] = left >> 8;
456                 out_block[3] = left;
457                 out_block[4] = right >> 24;
458                 out_block[5] = right >> 16;
459                 out_block[6] = right >> 8;
460                 out_block[7] = right;
461 #endif  /* UNALIGNED_POINTERS_PERMITTED */
450 #ifdef _BIG_ENDIAN
462         }
452 #endif
463 /* EXPORT DELETE END */
464         return (CRYPTO_SUCCESS);
465 }

467 /*
468  * Decrypt a block of data.  Because of addition operations, convert blocks
469  * to their big-endian representation, even on Intel boxen.
470  * It should look like the blowfish_encrypt_block() operation
471  * except for the order in which the S/P boxes are accessed.
472  */
473 /* ARGSUSED */
474 int
475 blowfish_decrypt_block(const void *cookie, const uint8_t *block,
476     uint8_t *out_block)
477 {
478 /* EXPORT DELETE START */
479         keysched_t *ksch = (keysched_t *)cookie;

481         uint32_t left, right, tmp;
482         uint32_t *P = ksch->ksch_P;
483         uint32_t *S = ksch->ksch_S;
484 #ifdef _BIG_ENDIAN
485         uint32_t *b32;

487         if (IS_P2ALIGNED(block, sizeof (uint32_t))) {
488                 /* LINTED:  pointer alignment */
489                 b32 = (uint32_t *)block;
490                 left = b32[0];
491                 right = b32[1];
492         } else
482         } else {
493 #endif
494         {
495         /*
496          * Read input block and place in left/right in big-endian order.
497          */
498 #ifdef UNALIGNED_POINTERS_PERMITTED
499         left = htonl(*(uint32_t *)&block[0]);
500         right = htonl(*(uint32_t *)&block[4]);
501 #else
502         left = ((uint32_t)block[0] << 24)
503             | ((uint32_t)block[1] << 16)
504             | ((uint32_t)block[2] << 8)
505             | (uint32_t)block[3];
506         right = ((uint32_t)block[4] << 24)
507             | ((uint32_t)block[5] << 16)
508             | ((uint32_t)block[6] << 8)
509             | (uint32_t)block[7];
510 #endif  /* UNALIGNED_POINTERS_PERMITTED */
495 #ifdef _BIG_ENDIAN
511         }
497 #endif

513         ROUND(left, right, 17);
514         ROUND(left, right, 16);
515         ROUND(left, right, 15);
```

```
        ROUND(left, right, 14);
        ROUND(left, right, 13);
        ROUND(left, right, 12);
        ROUND(left, right, 11);
        ROUND(left, right, 10);
        ROUND(left, right, 9);
        ROUND(left, right, 8);
        ROUND(left, right, 7);
        ROUND(left, right, 6);
        ROUND(left, right, 5);
        ROUND(left, right, 4);
        ROUND(left, right, 3);
        ROUND(left, right, 2);

        tmp = left;
        left = right;
        right = tmp;
        right ^= P[1];
        left ^= P[0];

#ifdef _BIG_ENDIAN
        if (IS_P2ALIGNED(out_block, sizeof (uint32_t))) {
                /* LINTED:  pointer alignment */
                b32 = (uint32_t *)out_block;
                b32[0] = left;
                b32[1] = right;
        } else
        } else {
#endif
        {
        /* Put the block back into the user's block with final swap */
#ifdef UNALIGNED_POINTERS_PERMITTED
                *(uint32_t *)&out_block[0] = htonl(left);
                *(uint32_t *)&out_block[4] = htonl(right);
#else
                out_block[0] = left >> 24;
                out_block[1] = left >> 16;
                out_block[2] = left >> 8;
                out_block[3] = left;
                out_block[4] = right >> 24;
                out_block[5] = right >> 16;
                out_block[6] = right >> 8;
                out_block[7] = right;
#endif  /* UNALIGNED_POINTERS_PERMITTED */
#ifdef _BIG_ENDIAN
        }
#endif
/* EXPORT DELETE END */
        return (CRYPTO_SUCCESS);
}
```

unchanged_portion_omitted_

```
**********************************************************
   47242 Wed Aug 27 14:22:32 2008
new/usr/src/common/crypto/des/des_impl.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/ddi.h>
#include <sys/sysmacros.h>
#include <sys/strsun.h>
#include <sys/crypto/spi.h>
#include <modes/modes.h>
#include <sys/crypto/common.h>
#include "des_impl.h"
#ifndef _KERNEL
#include <strings.h>
#include <stdlib.h>
#endif  /* !_KERNEL */

#if defined(__i386) || defined(__amd64)
#include <sys/byteorder.h>
#define UNALIGNED_POINTERS_PERMITTED
#endif

/* EXPORT DELETE START */

typedef struct keysched_s {
        uint64_t ksch_encrypt[16];
        uint64_t ksch_decrypt[16];
} keysched_t;
```

*unchanged_portion_omitted_*

```
#endif /* !sun4u */

/* EXPORT DELETE END */

int
des3_crunch_block(const void *cookie, const uint8_t block[DES_BLOCK_LEN],
```



```
    uint8_t out_block[DES_BLOCK_LEN], boolean_t decrypt)
{
/* EXPORT DELETE START */
        keysched3_t *ksch = (keysched3_t *)cookie;

        /*
         * The code below, that is always executed on LITTLE_ENDIAN machines,
         * reverses bytes in the block.  On BIG_ENDIAN, the same code
         * copies the block without reversing bytes.
         */
#ifdef _BIG_ENDIAN
        if (IS_P2ALIGNED(block, sizeof (uint64_t)) &&
            IS_P2ALIGNED(out_block, sizeof (uint64_t))) {
                if (decrypt == B_TRUE)
                        /* LINTED */
                        *(uint64_t *)out_block = des_crypt_impl(
                            ksch->ksch_decrypt, /* LINTED */
                            ksch->ksch_decrypt,
                            /* LINTED */
                            *(uint64_t *)block, 3);
                else
                        /* LINTED */
                        *(uint64_t *)out_block = des_crypt_impl(
                            ksch->ksch_encrypt, /* LINTED */
                            ksch->ksch_encrypt,
                            /* LINTED */
                            *(uint64_t *)block, 3);
        } else
#endif  /* _BIG_ENDIAN */
        {
        } else {
#endif
                uint64_t tmp;

#ifdef UNALIGNED_POINTERS_PERMITTED
                tmp = htonll(*(uint64_t *)&block[0]);
#else
                tmp = (((uint64_t)block[0] << 56) | ((uint64_t)block[1] << 48) |
                    ((uint64_t)block[2] << 40) | ((uint64_t)block[3] << 32) |
                    ((uint64_t)block[4] << 24) | ((uint64_t)block[5] << 16) |
                    ((uint64_t)block[6] << 8) | (uint64_t)block[7]);
#endif  /* UNALIGNED_POINTERS_PERMITTED */

                if (decrypt == B_TRUE)
                        tmp = des_crypt_impl(ksch->ksch_decrypt, tmp, 3);
                else
                        tmp = des_crypt_impl(ksch->ksch_encrypt, tmp, 3);

#ifdef UNALIGNED_POINTERS_PERMITTED
                *(uint64_t *)&out_block[0] = htonll(tmp);
#else
                out_block[0] = tmp >> 56;
                out_block[1] = tmp >> 48;
                out_block[2] = tmp >> 40;
                out_block[3] = tmp >> 32;
                out_block[4] = tmp >> 24;
                out_block[5] = tmp >> 16;
                out_block[6] = tmp >> 8;
                out_block[7] = (uint8_t)tmp;
#endif  /* UNALIGNED_POINTERS_PERMITTED */
#ifdef _BIG_ENDIAN
        }
#endif
/* EXPORT DELETE END */
        return (CRYPTO_SUCCESS);
}
```

```
int
des_crunch_block(const void *cookie, const uint8_t block[DES_BLOCK_LEN],
    uint8_t out_block[DES_BLOCK_LEN], boolean_t decrypt)
{
/* EXPORT DELETE START */
        keysched_t *ksch = (keysched_t *)cookie;

        /*
         * The code below, that is always executed on LITTLE_ENDIAN machines,
         * reverses bytes in the block.  On BIG_ENDIAN, the same code
         * copies the block without reversing bytes.
         */
#ifdef _BIG_ENDIAN
        if (IS_P2ALIGNED(block, sizeof (uint64_t)) &&
            IS_P2ALIGNED(out_block, sizeof (uint64_t))) {
                if (decrypt == B_TRUE)
                        /* LINTED */
                        *(uint64_t *)out_block = des_crypt_impl(
                            ksch->ksch_decrypt, /* LINTED */
                            ksch->ksch_decrypt,
                            /* LINTED */
                            *(uint64_t *)block, 1);
                else
                        /* LINTED */
                        *(uint64_t *)out_block = des_crypt_impl(
                            ksch->ksch_encrypt, /* LINTED */
                            ksch->ksch_encrypt,
                            /* LINTED */
                            *(uint64_t *)block, 1);

        } else
#endif  /* _BIG_ENDIAN */
        {
        } else {
#endif
                uint64_t tmp;

#ifdef UNALIGNED_POINTERS_PERMITTED
                tmp = htonll(*(uint64_t *)&block[0]);
#else
                tmp = (((uint64_t)block[0] << 56) | ((uint64_t)block[1] << 48) |
                    ((uint64_t)block[2] << 40) | ((uint64_t)block[3] << 32) |
                    ((uint64_t)block[4] << 24) | ((uint64_t)block[5] << 16) |
                    ((uint64_t)block[6] << 8) | (uint64_t)block[7]);
#endif  /* UNALIGNED_POINTERS_PERMITTED */


                if (decrypt == B_TRUE)
                        tmp = des_crypt_impl(ksch->ksch_decrypt, tmp, 1);
                else
                        tmp = des_crypt_impl(ksch->ksch_encrypt, tmp, 1);

#ifdef UNALIGNED_POINTERS_PERMITTED
                *(uint64_t *)&out_block[0] = htonll(tmp);
#else
                out_block[0] = tmp >> 56;
                out_block[1] = tmp >> 48;
                out_block[2] = tmp >> 40;
                out_block[3] = tmp >> 32;
                out_block[4] = tmp >> 24;
                out_block[5] = tmp >> 16;
                out_block[6] = tmp >> 8;
                out_block[7] = (uint8_t)tmp;
#endif  /* UNALIGNED_POINTERS_PERMITTED */
#ifdef _BIG_ENDIAN
```



```
        }
#endif
/* EXPORT DELETE END */
        return (CRYPTO_SUCCESS);
}

static boolean_t
keycheck(uint8_t *key, uint8_t *corrected_key)
{
/* EXPORT DELETE START */
        uint64_t key_so_far;
        uint_t i;
        /*
         * Table of weak and semi-weak keys.  Fortunately, weak keys are
         * endian-independent, and some semi-weak keys can be paired up in
         * endian-opposite order.  Since keys are stored as uint64_t's,
         * use the ifdef _LITTLE_ENDIAN where appropriate.
         */
        static uint64_t des_weak_keys[] = {
                /* Really weak keys.  Byte-order independent values. */
                0x0101010101010101ULL,
                0x1f1f1f1f0e0e0e0eULL,
                0xe0e0e0e0f1f1f1f1ULL,
                0xfefefefefefefefeULL,

                /* Semi-weak (and a few possibly-weak) keys. */

                /* Byte-order independent semi-weak keys. */
                0x01fe01fe01fe01feULL,  0xfe01fe01fe01fe01ULL,

                /* Byte-order dependent semi-weak keys. */
#ifdef _LITTLE_ENDIAN
                0xf10ef10ee01fe01fULL,  0x0ef10ef11fe01fe0ULL,
                0x01f101f101e001e0ULL,  0xf101f101e001e001ULL,
                0x0efe0efe1ffe1ffeULL,  0xfe0efe0efe1ffe1fULL,
                0x010e010e011f011fULL,  0x0e010e011f011f01ULL,
                0xf1fef1fee0fee0feULL,  0xfef1fef1fee0fee0ULL,
#else   /* Big endian */
                0x1fe01fe00ef10ef1ULL,  0xe01fe01ff10ef10eULL,
                0x01e001e001f101f1ULL,  0xe001e001f101f101ULL,
                0x1ffe1ffe0efe0efeULL,  0xfe1ffe1ffe0efe0eULL,
                0x011f011f010e010eULL,  0x1f011f010e010e01ULL,
                0xe0fee0fef1fef1feULL,  0xfee0fee0fef1fef1ULL,
#endif  /* _LITTLE_ENDIAN */
#endif

                /* We'll save the other possibly-weak keys for the future. */
        };

        if (key == NULL)
                return (B_FALSE);

#ifdef UNALIGNED_POINTERS_PERMITTED
        key_so_far = htonll(*(uint64_t *)&key[0]);
#else
        /*
         * The code below reverses the bytes on LITTLE_ENDIAN machines.
         * On BIG_ENDIAN, the same code copies without reversing
         * the bytes.
         */
        key_so_far = (((uint64_t)key[0] << 56) | ((uint64_t)key[1] << 48) |
            ((uint64_t)key[2] << 40) | ((uint64_t)key[3] << 32) |
            ((uint64_t)key[4] << 24) | ((uint64_t)key[5] << 16) |
            ((uint64_t)key[6] << 8) | (uint64_t)key[7]);
#endif  /* UNALIGNED_POINTERS_PERMITTED */
```

```
689         /*
690          * Fix parity.
691          */
692         fix_des_parity(&key_so_far);
693
694         /* Do weak key check itself. */
695         for (i = 0; i < (sizeof (des_weak_keys) / sizeof (uint64_t)); i++)
696                 if (key_so_far == des_weak_keys[i]) {
697                         return (B_FALSE);
698                 }
699
700         if (corrected_key != NULL) {
701 #ifdef UNALIGNED_POINTERS_PERMITTED
702                 *(uint64_t *)&corrected_key[0] = htonll(key_so_far);
703 #else
704                 /*
705                  * The code below reverses the bytes on LITTLE_ENDIAN machines.
706                  * On BIG_ENDIAN, the same code copies without reversing
707                  * the bytes.
708                  */
709                 corrected_key[0] = key_so_far >> 56;
710                 corrected_key[1] = key_so_far >> 48;
711                 corrected_key[2] = key_so_far >> 40;
712                 corrected_key[3] = key_so_far >> 32;
713                 corrected_key[4] = key_so_far >> 24;
714                 corrected_key[5] = key_so_far >> 16;
715                 corrected_key[6] = key_so_far >> 8;
716                 corrected_key[7] = (uint8_t)key_so_far;
717 #endif  /* UNALIGNED_POINTERS_PERMITTED */
718         }
719 /* EXPORT DELETE END */
720         return (B_TRUE);
721 }
722
723 static boolean_t
724 des3_keycheck(uint8_t *key, uint8_t *corrected_key)
725 {
726 /* EXPORT DELETE START */
727         uint64_t aligned_key[DES3_KEYSIZE / sizeof (uint64_t)];
728         uint64_t key_so_far, scratch, *currentkey;
729         uint_t j, num_weakkeys = 0;
730
731         if (key == NULL) {
732                 return (B_FALSE);
733         }
734
735         if (!IS_P2ALIGNED(key, sizeof (uint64_t))) {
736                 bcopy(key, aligned_key, DES3_KEYSIZE);
737                 currentkey = (uint64_t *)aligned_key;
738         } else {
739                 /* LINTED */
740                 currentkey = (uint64_t *)key;
741         }
742
743         for (j = 0; j < 3; j++) {
744                 key_so_far = currentkey[j];
745
746                 if (!keycheck((uint8_t *)&key_so_far, (uint8_t *)&scratch)) {
747                         if (++num_weakkeys > 1) {
748                                 return (B_FALSE);
749                         }
750                         /*
751                          * We found a weak key, but since
752                          * we've only found one weak key,
753                          * we can not reject the whole 3DES
754                          * set of keys as weak.
```



```
755                          *
756                          * Break from the weak key loop
757                          * (since this DES key is weak) and
758                          * continue on.
759                          */
760                 }
761
762                 currentkey[j] = scratch;
763         }
764
765         /*
766          * Perform key equivalence checks, now that parity is properly set.
767          * 1st and 2nd keys must be unique, the 3rd key can be the same as
768          * the 1st key for the 2 key variant of 3DES.
746          * the 1st key for the 2 key varient of 3DES.
769          */
770         if (currentkey[0] == currentkey[1] || currentkey[1] == currentkey[2])
771                 return (B_FALSE);
772
773         if (corrected_key != NULL) {
774                 bcopy(currentkey, corrected_key, DES3_KEYSIZE);
775         }
776
777 /* EXPORT DELETE END */
778         return (B_TRUE);
779 }
_____unchanged_portion_omitted_
792
793 void
794 des_parity_fix(uint8_t *key, des_strength_t strength, uint8_t *corrected_key)
795 {
796 /* EXPORT DELETE START */
797         uint64_t aligned_key[DES3_KEYSIZE / sizeof (uint64_t)];
798         uint8_t *paritied_key;
799         uint64_t key_so_far;
800         int i = 0, offset = 0;
801
802         if (strength == DES)
803                 bcopy(key, aligned_key, DES_KEYSIZE);
804         else
805                 bcopy(key, aligned_key, DES3_KEYSIZE);
806
807         paritied_key = (uint8_t *)aligned_key;
808         while (strength > i) {
809                 offset = 8 * i;
810 #ifdef UNALIGNED_POINTERS_PERMITTED
811                 key_so_far = htonll(*(uint64_t *)&paritied_key[offset]);
812 #else
813                 key_so_far = (((uint64_t)paritied_key[offset + 0] << 56) |
814                     ((uint64_t)paritied_key[offset + 1] << 48) |
815                     ((uint64_t)paritied_key[offset + 2] << 40) |
816                     ((uint64_t)paritied_key[offset + 3] << 32) |
817                     ((uint64_t)paritied_key[offset + 4] << 24) |
818                     ((uint64_t)paritied_key[offset + 5] << 16) |
819                     ((uint64_t)paritied_key[offset + 6] << 8) |
820                     (uint64_t)paritied_key[offset + 7]);
821 #endif  /* UNALIGNED_POINTERS_PERMITTED */
822
823                 fix_des_parity(&key_so_far);
824
825 #ifdef UNALIGNED_POINTERS_PERMITTED
826                 *(uint64_t *)&paritied_key[offset] = htonll(key_so_far);
827 #else
828                 paritied_key[offset + 0] = key_so_far >> 56;
829                 paritied_key[offset + 1] = key_so_far >> 48;
830                 paritied_key[offset + 2] = key_so_far >> 40;
```

```
831                 paritied_key[offset + 3] = key_so_far >> 32;
832                 paritied_key[offset + 4] = key_so_far >> 24;
833                 paritied_key[offset + 5] = key_so_far >> 16;
834                 paritied_key[offset + 6] = key_so_far >> 8;
835                 paritied_key[offset + 7] = (uint8_t)key_so_far;
836 #endif  /* UNALIGNED_POINTERS_PERMITTED */
837
838                 i++;
839         }
840
841         bcopy(paritied_key, corrected_key, DES_KEYSIZE * strength);
842 /* EXPORT DELETE END */
843 }
844
845
846 /*
847  * Initialize key schedule for DES, DES2, and DES3
848  */
849 void
850 des_init_keysched(uint8_t *cipherKey, des_strength_t strength, void *ks)
851 {
852 /* EXPORT DELETE START */
853         uint64_t *encryption_ks;
854         uint64_t *decryption_ks;
855         uint64_t keysched[48];
856         uint64_t key_uint64[3];
857         uint64_t tmp;
858         uint_t keysize, i, j;
859
860         switch (strength) {
861         case DES:
862                 keysize = DES_KEYSIZE;
863                 encryption_ks = ((keysched_t *)ks)->ksch_encrypt;
864                 decryption_ks = ((keysched_t *)ks)->ksch_decrypt;
865                 break;
866         case DES2:
867                 keysize = DES2_KEYSIZE;
868                 encryption_ks = ((keysched3_t *)ks)->ksch_encrypt;
869                 decryption_ks = ((keysched3_t *)ks)->ksch_decrypt;
870                 break;
871         case DES3:
872                 keysize = DES3_KEYSIZE;
873                 encryption_ks = ((keysched3_t *)ks)->ksch_encrypt;
874                 decryption_ks = ((keysched3_t *)ks)->ksch_decrypt;
875         }
876
877         /*
878          * The code below, that is always executed on LITTLE_ENDIAN machines,
879          * reverses every 8 bytes in the key.  On BIG_ENDIAN, the same code
880          * copies the key without reversing bytes.
881          */
882 #ifdef _BIG_ENDIAN
883         if (IS_P2ALIGNED(cipherKey, sizeof (uint64_t))) {
884                 for (i = 0, j = 0; j < keysize; i++, j += 8) {
885                         /* LINTED: pointer alignment */
886                         key_uint64[i] = *((uint64_t *)&cipherKey[j]);
887                 }
888         } else
889 #endif  /* _BIG_ENDIAN */
858         } else {
859 #endif
890         {
891                 for (i = 0, j = 0; j < keysize; i++, j += 8) {
892 #ifdef UNALIGNED_POINTERS_PERMITTED
893                         key_uint64[i] = htonll(*(uint64_t *)&cipherKey[j]);
894 #else
```



```
895                         key_uint64[i] = (((uint64_t)cipherKey[j] << 56) |
896                             ((uint64_t)cipherKey[j + 1] << 48) |
897                             ((uint64_t)cipherKey[j + 2] << 40) |
898                             ((uint64_t)cipherKey[j + 3] << 32) |
899                             ((uint64_t)cipherKey[j + 4] << 24) |
900                             ((uint64_t)cipherKey[j + 5] << 16) |
901                             ((uint64_t)cipherKey[j + 6] << 8) |
902                             (uint64_t)cipherKey[j + 7]);
903 #endif  /* UNALIGNED_POINTERS_PERMITTED */
904                 }
905         }
872 #ifdef _BIG_ENDIAN
873         }
874 #endif
875
907         switch (strength) {
908         case DES:
909                 des_ks(keysched, key_uint64[0]);
910                 break;
911
912         case DES2:
913                 /* DES2 is just DES3 with the first and third keys the same */
914                 bcopy(key_uint64, key_uint64 + 2, DES_KEYSIZE);
915                 /* FALLTHRU */
916         case DES3:
917                 des_ks(keysched, key_uint64[0]);
918                 des_ks(keysched + 16, key_uint64[1]);
919                 for (i = 0; i < 8; i++) {
920                         tmp = keysched[16+i];
921                         keysched[16+i] = keysched[31-i];
922                         keysched[31-i] = tmp;
923                 }
924                 des_ks(keysched+32, key_uint64[2]);
925                 keysize = DES3_KEYSIZE;
926         }
927
928         /* save the encryption keyschedule */
929         bcopy(keysched, encryption_ks, keysize * 16);
930
931         /* reverse the key schedule */
932         for (i = 0; i < keysize; i++) {
933                 tmp = keysched[i];
934                 keysched[i] = keysched[2 * keysize - 1 - i];
935                 keysched[2 * keysize -1 -i] = tmp;
936         }
937
938         /* save the decryption keyschedule */
939         bcopy(keysched, decryption_ks, keysize * 16);
940 /* EXPORT DELETE END */
941 }
______unchanged_portion_omitted_
942
982 /*
983  * Replace the LSB of each byte by the xor of the other
984  * 7 bits.  The tricky thing is that the original contents of the LSBs
985  * are nullified by including them twice in the xor computation.
954  * are nullifed by including them twice in the xor computation.
986  */
987 static void
988 fix_des_parity(uint64_t *keyp)
989 {
990 /* EXPORT DELETE START */
991         uint64_t k = *keyp;
992         k ^= k >> 1;
993         k ^= k >> 2;
994         k ^= k >> 4;
```

```
 995         *keyp ^= (k & 0x0101010101010101ULL);
 996         *keyp ^= 0x0101010101010101ULL;
 997 /* EXPORT DELETE END */
 998 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8340 Wed Aug 27 14:22:37 2008
new/usr/src/common/crypto/md4/md4.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * MD4C.C - RSA Data Security, Inc., MD4 message-digest algorithm
 */

/*
 * Copyright (C) 1990-2, RSA Data Security, Inc. All rights reserved.
 *
 * License to copy and use this software is granted provided that it
 * is identified as the "RSA Data Security, Inc. MD4 Message-Digest
 * Algorithm" in all material mentioning or referencing this software
 * or this function.
 *
 * License is also granted to make and use derivative works provided
 * that such works are identified as "derived from the RSA Data
 * Security, Inc. MD4 Message-Digest Algorithm" in all material
 * mentioning or referencing the derived work.
 *
 * RSA Data Security, Inc. makes no representations concerning either
 * the merchantability of this software or the suitability of this
 * software for any particular purpose. It is provided "as is"
 * without express or implied warranty of any kind.
 *
 * These notices must be retained in any copies of any part of this
 * documentation and/or software.
 */

#include <sys/types.h>
#ifdef _KERNEL
#include <sys/sunddi.h>
#else
#include <strings.h>
#endif /* _KERNEL */

#if defined(__i386) || defined(__amd64)
#define UNALIGNED_POINTERS_PERMITTED
#endif

#include <sys/md4.h>

/*
 * Constants for MD4Transform routine.
 */
#define S11 3
#define S12 7
#define S13 11
#define S14 19
#define S21 3
#define S22 5
#define S23 9
#define S24 13
#define S31 3
```



```
#define S32 9
#define S33 11
#define S34 15

static void MD4Transform(uint32_t [4], unsigned char [64]);
static void Encode(unsigned char *, uint32_t *, unsigned int);
static void Decode(uint32_t *, unsigned char *, unsigned int);

static unsigned char PADDING[64] = {
        0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};
_____unchanged_portion_omitted_

/*
 * Encodes input (uint32_t) into output (unsigned char). Assumes len is
 * a multiple of 4.
 */
static void
Encode(unsigned char *output, uint32_t *input, unsigned int len)
Encode(output, input, len)
        unsigned char *output;
        uint32_t *input;
        unsigned int len;
{
        unsigned int i, j;

        for (i = 0, j = 0; j < len; i++, j += 4) {
#if defined(_LITTLE_ENDIAN) && defined(UNALIGNED_POINTERS_PERMITTED)
                *(uint32_t *)&output[j] = input[i];
#else
                /* endian-independent code */
                output[j] = (unsigned char)(input[i] & 0xff);
                output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
                output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
                output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
#endif  /* _LITTLE_ENDIAN && UNALIGNED_POINTERS_PERMITTED */
        }
}

/*
 * Decodes input (unsigned char) into output (uint32_t). Assumes len is
 * a multiple of 4.
 */
static void
Decode(uint32_t *output, unsigned char *input, unsigned int len)
Decode(output, input, len)
        uint32_t *output;
        unsigned char *input;
        unsigned int len;
{
        unsigned int i, j;

        for (i = 0, j = 0; j < len; i++, j += 4) {
#if defined(_LITTLE_ENDIAN) && defined(UNALIGNED_POINTERS_PERMITTED)
                output[i] = *(uint32_t *)&input[j];
#else
                /* endian-independent code */
        for (i = 0, j = 0; j < len; i++, j += 4)
                output[i] = ((uint32_t)input[j]) |
                    (((uint32_t)input[j+1]) << 8) |
                    (((uint32_t)input[j+2]) << 16) |
                    (((uint32_t)input[j+3]) << 24);
#endif  /* _LITTLE_ENDIAN && UNALIGNED_POINTERS_PERMITTED */
        }
```

```
 305 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    7387 Wed Aug 27 14:22:42 2008
new/usr/src/common/crypto/md5/md5_byteswap.h
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _MD5_BYTESWAP_H
#define _MD5_BYTESWAP_H

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

/*
 * definitions for inline functions for little-endian loads.
 *
 * This file has special definitions for UltraSPARC architectures,
 * which have a special address space identifier for loading 32 and 16 bit
 * integers in little-endian byte order.
 *
 * This file and common/crypto/md5/sparc/sun4[uv]/byteswap.il implement the
 * same thing and must be changed together.
 */

#include <sys/types.h>
#if defined(__sparc)
#include <v9/sys/asi.h>
#elif defined(_LITTLE_ENDIAN)
#include <sys/byteorder.h>
#endif

#ifdef  __cplusplus
extern "C" {
#endif

#if defined(_LITTLE_ENDIAN)

/*
 * Little-endian optimization:  I don't need to do any weirdness.   On
 * some little-endian boxen, I'll have to do alignment checks, but I can do
```



```
 * that below.
 */

#if !defined(__i386) && !defined(__amd64)
/*
 * i386 and amd64 don't require aligned 4-byte loads.  The symbol
 * _MD5_CHECK_ALIGNMENT indicates below whether the MD5Transform function
 * requires alignment checking.
 */
#define _MD5_CHECK_ALIGNMENT
#endif /* !__i386 && !__amd64 */

#define LOAD_LITTLE_32(addr)    (*(uint32_t *)(addr))

#else   /* !_LITTLE_ENDIAN */

/*
 * sparc v9/v8plus optimization:
 *
 * on the sparc v9/v8plus, we can load data little endian.  however, since
 * the compiler doesn't have direct support for little endian, we
 * link to an assembly-language routine 'load_little_32' to do
 * the magic.  note that special care must be taken to ensure the
 * address is 32-bit aligned -- in the interest of speed, we don't
 * check to make sure, since careful programming can guarantee this
 * for us.
 */
#if defined(sun4u)

/* Define alignment check because we can 4-byte load as little endian. */
#define _MD5_CHECK_ALIGNMENT
#define LOAD_LITTLE_32(addr)    load_little_32((uint32_t *)(addr))

#if !defined(__lint) && defined(__GNUC__)

static __inline__ uint32_t
load_little_32(uint32_t *addr)
{
        uint32_t value;

        __asm__(
            "lduwa      [%1] %2, %0\n\t"
            : "=r" (value)
            : "r" (addr), "i" (ASI_PL));

        return (value);
}

static __inline__ uint16_t
load_little_16(uint16_t *addr)
{
        uint16_t value;

        __asm__(
            "lduha      [%1] %2, %0\n\t"
        : "=r" (value)
        : "r" (addr), "i" (ASI_PL));

        return (value);
}

#endif  /* !__lint && __GNUC__ */

#if !defined(__GNUC__)
extern  uint32_t load_little_32(uint32_t *);
#endif  /* !__GNUC__ */
```

```
110 /* Placate lint */
111 #if defined(__lint)
112 uint32_t
113 load_little_32(uint32_t *addr)
114 {
115         return (*addr);
116 }
117 #endif  /* __lint */

119 #elif defined(_LITTLE_ENDIAN)
120 #define LOAD_LITTLE_32(addr)    htonl(addr)
132 #else   /* !sun4u */

122 #else
123 /* big endian -- will work on little endian, but slowly */
124 /* Since we do byte operations, we don't have to check for alignment. */
125 #define LOAD_LITTLE_32(addr)    \
126         ((addr)[0] | ((addr)[1] << 8) | ((addr)[2] << 16) | ((addr)[3] << 24))

127 #endif  /* sun4u */

129 #if defined(sun4v)

131 /*
132  * For N1 want to minimize number of arithmetic operations. This is best
133  * achieved by using the %asi register to specify ASI for the lduwa operations.
134  * Also, have a separate inline template for each word, so can utilize the
135  * immediate offset in lduwa, without relying on the compiler to do the right
136  * thing.
137  *
138  * Moving to 64-bit loads might also be beneficial.
139  */
140 #define LOAD_LITTLE_32_0(addr)  load_little_32_0((uint32_t *)(addr))
141 #define LOAD_LITTLE_32_1(addr)  load_little_32_1((uint32_t *)(addr))
142 #define LOAD_LITTLE_32_2(addr)  load_little_32_2((uint32_t *)(addr))
143 #define LOAD_LITTLE_32_3(addr)  load_little_32_3((uint32_t *)(addr))
144 #define LOAD_LITTLE_32_4(addr)  load_little_32_4((uint32_t *)(addr))
145 #define LOAD_LITTLE_32_5(addr)  load_little_32_5((uint32_t *)(addr))
146 #define LOAD_LITTLE_32_6(addr)  load_little_32_6((uint32_t *)(addr))
147 #define LOAD_LITTLE_32_7(addr)  load_little_32_7((uint32_t *)(addr))
148 #define LOAD_LITTLE_32_8(addr)  load_little_32_8((uint32_t *)(addr))
149 #define LOAD_LITTLE_32_9(addr)  load_little_32_9((uint32_t *)(addr))
150 #define LOAD_LITTLE_32_a(addr)  load_little_32_a((uint32_t *)(addr))
151 #define LOAD_LITTLE_32_b(addr)  load_little_32_b((uint32_t *)(addr))
152 #define LOAD_LITTLE_32_c(addr)  load_little_32_c((uint32_t *)(addr))
153 #define LOAD_LITTLE_32_d(addr)  load_little_32_d((uint32_t *)(addr))
154 #define LOAD_LITTLE_32_e(addr)  load_little_32_e((uint32_t *)(addr))
155 #define LOAD_LITTLE_32_f(addr)  load_little_32_f((uint32_t *)(addr))

157 #if !defined(__lint) && defined(__GNUC__)

159 /*
160  * This actually sets the ASI register, not necessarily to ASI_PL.
161  */
162 static __inline__ void
163 set_little(uint8_t asi)
164 {
165         __asm__ __volatile__(
166             "wr %%g0, %0, %%asi\n\t"
167             : /* Nothing */
168             : "r" (asi));
169 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   25319 Wed Aug 27 14:22:47 2008
new/usr/src/common/crypto/modes/ccm.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _KERNEL
#include <strings.h>
#include <limits.h>
#include <assert.h>
#include <security/cryptoki.h>
#endif

#include <sys/types.h>
#include <sys/kmem.h>
#include <modes/modes.h>
#include <sys/crypto/common.h>
#include <sys/crypto/impl.h>

#if defined(__i386) || defined(__amd64)
#include <sys/byteorder.h>
#define UNALIGNED_POINTERS_PERMITTED
#endif

/*
 * Encrypt multiple blocks of data in CCM mode.  Decrypt for CCM mode
 * is done in another function.
 */
int
ccm_mode_encrypt_contiguous_blocks(ccm_ctx_t *ctx, char *data, size_t length,
    crypto_data_t *out, size_t block_size,
    int (*encrypt_block)(const void *, const uint8_t *, uint8_t *),
    void (*copy_block)(uint8_t *, uint8_t *),
    void (*xor_block)(uint8_t *, uint8_t *))
{
        size_t remainder = length;
        size_t need;
        uint8_t *datap = (uint8_t *)data;
        uint8_t *blockp;
        uint8_t *lastp;
```



```
        void *iov_or_mp;
        offset_t offset;
        uint8_t *out_data_1;
        uint8_t *out_data_2;
        size_t out_data_1_len;
        uint64_t counter;
        uint8_t *mac_buf;
#ifdef _LITTLE_ENDIAN
        uint8_t *p;
#endif

        if (length + ctx->ccm_remainder_len < block_size) {
                /* accumulate bytes here and return */
                bcopy(datap,
                    (uint8_t *)ctx->ccm_remainder + ctx->ccm_remainder_len,
                    length);
                ctx->ccm_remainder_len += length;
                ctx->ccm_copy_to = datap;
                return (CRYPTO_SUCCESS);
        }

        lastp = (uint8_t *)ctx->ccm_cb;
        if (out != NULL)
                crypto_init_ptrs(out, &iov_or_mp, &offset);

        mac_buf = (uint8_t *)ctx->ccm_mac_buf;

        do {
                /* Unprocessed data from last call. */
                if (ctx->ccm_remainder_len > 0) {
                        need = block_size - ctx->ccm_remainder_len;

                        if (need > remainder)
                                return (CRYPTO_DATA_LEN_RANGE);

                        bcopy(datap, &((uint8_t *)ctx->ccm_remainder)
                            [ctx->ccm_remainder_len], need);

                        blockp = (uint8_t *)ctx->ccm_remainder;
                } else {
                        blockp = datap;
                }

                /*
                 * do CBC MAC
                 *
                 * XOR the previous cipher block current clear block.
                 * mac_buf always contain previous cipher block.
                 */
                xor_block(blockp, mac_buf);
                encrypt_block(ctx->ccm_keysched, mac_buf, mac_buf);

                /* ccm_cb is the counter block */
                encrypt_block(ctx->ccm_keysched, (uint8_t *)ctx->ccm_cb,
                    (uint8_t *)ctx->ccm_tmp);

                lastp = (uint8_t *)ctx->ccm_tmp;

                /*
                 * Increment counter. Counter bits are confined
                 * to the bottom 64 bits of the counter block.
                 */
#ifdef _LITTLE_ENDIAN
                counter = ntohll(ctx->ccm_cb[1] & ctx->ccm_counter_mask);
                counter = htonll(counter + 1);
#else
```

```
123                 counter = ctx->ccm_cb[1] & ctx->ccm_counter_mask;
118 #ifdef _LITTLE_ENDIAN
119                 p = (uint8_t *)&counter;
120                 counter = (((uint64_t)p[0] << 56) |
121                     ((uint64_t)p[1] << 48) |
122                     ((uint64_t)p[2] << 40) |
123                     ((uint64_t)p[3] << 32) |
124                     ((uint64_t)p[4] << 24) |
125                     ((uint64_t)p[5] << 16) |
126                     ((uint64_t)p[6] << 8) |
127                     (uint64_t)p[7]);
128 #endif
124                 counter++;
125 #endif  /* _LITTLE_ENDIAN */
130 #ifdef _LITTLE_ENDIAN
131                 counter = (((uint64_t)p[0] << 56) |
132                     ((uint64_t)p[1] << 48) |
133                     ((uint64_t)p[2] << 40) |
134                     ((uint64_t)p[3] << 32) |
135                     ((uint64_t)p[4] << 24) |
136                     ((uint64_t)p[5] << 16) |
137                     ((uint64_t)p[6] << 8) |
138                     (uint64_t)p[7]);
139 #endif
126                 counter &= ctx->ccm_counter_mask;
127                 ctx->ccm_cb[1] =
128                     (ctx->ccm_cb[1] & ~(ctx->ccm_counter_mask)) | counter;

130                 /*
131                  * XOR encrypted counter block with the current clear block.
132                  */
133                 xor_block(blockp, lastp);

135                 ctx->ccm_processed_data_len += block_size;

137                 if (out == NULL) {
138                         if (ctx->ccm_remainder_len > 0) {
139                                 bcopy(blockp, ctx->ccm_copy_to,
140                                     ctx->ccm_remainder_len);
141                                 bcopy(blockp + ctx->ccm_remainder_len, datap,
142                                     need);
143                         }
144                 } else {
145                         crypto_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
146                             &out_data_1_len, &out_data_2, block_size);

148                         /* copy block to where it belongs */
149                         if (out_data_1_len == block_size) {
150                                 copy_block(lastp, out_data_1);
151                         } else {
152                                 bcopy(lastp, out_data_1, out_data_1_len);
153                                 if (out_data_2 != NULL) {
154                                         bcopy(lastp + out_data_1_len,
155                                             out_data_2,
156                                             block_size - out_data_1_len);
157                                 }
158                         }
159                         /* update offset */
160                         out->cd_offset += block_size;
161                 }

163                 /* Update pointer to next block of data to be processed. */
164                 if (ctx->ccm_remainder_len != 0) {
165                         datap += need;
166                         ctx->ccm_remainder_len = 0;
167                 } else {
```



```
168                         datap += block_size;
169                 }

171                 remainder = (size_t)&data[length] - (size_t)datap;

173                 /* Incomplete last block. */
174                 if (remainder > 0 && remainder < block_size) {
175                         bcopy(datap, ctx->ccm_remainder, remainder);
176                         ctx->ccm_remainder_len = remainder;
177                         ctx->ccm_copy_to = datap;
178                         goto out;
179                 }
180                 ctx->ccm_copy_to = NULL;

182         } while (remainder > 0);

184 out:
185         return (CRYPTO_SUCCESS);
186 }
_____unchanged_portion_omitted_

211 /* ARGSUSED */
212 int
213 ccm_encrypt_final(ccm_ctx_t *ctx, crypto_data_t *out, size_t block_size,
214     int (*encrypt_block)(const void *, const uint8_t *, uint8_t *),
215     void (*xor_block)(uint8_t *, uint8_t *))
216 {
217         uint8_t *lastp, *mac_buf, *ccm_mac_p, *macp;
218         void *iov_or_mp;
219         offset_t offset;
220         uint8_t *out_data_1;
221         uint8_t *out_data_2;
222         size_t out_data_1_len;
223         int i;

225         if (out->cd_length < (ctx->ccm_remainder_len + ctx->ccm_mac_len)) {
226                 return (CRYPTO_DATA_LEN_RANGE);
227         }

229         /*
230          * When we get here, the number of bytes of payload processed
231          * plus whatever data remains, if any,
232          * should be the same as the number of bytes that's being
233          * passed in the argument during init time.
234          */
235         if ((ctx->ccm_processed_data_len + ctx->ccm_remainder_len)
236             != (ctx->ccm_data_len)) {
237                 return (CRYPTO_DATA_LEN_RANGE);
238         }

240         mac_buf = (uint8_t *)ctx->ccm_mac_buf;

242         if (ctx->ccm_remainder_len > 0) {

244                 /* ccm_mac_input_buf is not used for encryption */
245                 macp = (uint8_t *)ctx->ccm_mac_input_buf;
246                 bzero(macp, block_size);

248                 /* copy remainder to temporary buffer */
249                 bcopy(ctx->ccm_remainder, macp, ctx->ccm_remainder_len);

251                 /* calculate the CBC MAC */
252                 xor_block(macp, mac_buf);
253                 encrypt_block(ctx->ccm_keysched, mac_buf, mac_buf);

255                 /* calculate the counter mode */
```

```
256                 lastp = (uint8_t *)ctx->ccm_tmp;
257                 encrypt_block(ctx->ccm_keysched, (uint8_t *)ctx->ccm_cb, lastp);

259                 /* XOR with counter block */
260                 for (i = 0; i < ctx->ccm_remainder_len; i++) {
261                         macp[i] ^= lastp[i];
262                 }
263                 ctx->ccm_processed_data_len += ctx->ccm_remainder_len;
264         }

266         /* Calculate the CCM MAC */
267         ccm_mac_p = (uint8_t *)ctx->ccm_tmp;
268         calculate_ccm_mac(ctx, ccm_mac_p, encrypt_block);

270         crypto_init_ptrs(out, &iov_or_mp, &offset);
271         crypto_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
272             &out_data_1_len, &out_data_2,
273             ctx->ccm_remainder_len + ctx->ccm_mac_len);

275         if (ctx->ccm_remainder_len > 0) {

277                 /* copy temporary block to where it belongs */
278                 if (out_data_2 == NULL) {
279                         /* everything will fit in out_data_1 */
280                         bcopy(macp, out_data_1, ctx->ccm_remainder_len);
281                         bcopy(ccm_mac_p, out_data_1 + ctx->ccm_remainder_len,
282                             ctx->ccm_mac_len);
283                 } else {

285                         if (out_data_1_len < ctx->ccm_remainder_len) {

287                                 size_t data_2_len_used;

289                                 bcopy(macp, out_data_1, out_data_1_len);

291                                 data_2_len_used = ctx->ccm_remainder_len
292                                     - out_data_1_len;

294                                 bcopy((uint8_t *)macp + out_data_1_len,
295                                     out_data_2, data_2_len_used);
296                                 bcopy(ccm_mac_p, out_data_2 + data_2_len_used,
297                                     ctx->ccm_mac_len);
298                         } else {
299                                 bcopy(macp, out_data_1, out_data_1_len);
300                                 if (out_data_1_len == ctx->ccm_remainder_len) {
301                                         /* mac will be in out_data_2 */
302                                         bcopy(ccm_mac_p, out_data_2,
303                                             ctx->ccm_mac_len);
304                                 } else {
305                                         size_t len_not_used = out_data_1_len -
319                                         size_t len_not_used
320                                             = out_data_1_len -
306                                             ctx->ccm_remainder_len;
307                                         /*
308                                          * part of mac in will be in
309                                          * out_data_1, part of the mac will be
310                                          * in out_data_2
311                                          */
312                                         bcopy(ccm_mac_p,
313                                             out_data_1 + ctx->ccm_remainder_len,
314                                             len_not_used);
315                                         bcopy(ccm_mac_p + len_not_used,
316                                             out_data_2,
317                                             ctx->ccm_mac_len - len_not_used);

319                                 }
```



```
320                         }
321                 }
322         } else {
323                 /* copy block to where it belongs */
324                 bcopy(ccm_mac_p, out_data_1, out_data_1_len);
325                 if (out_data_2 != NULL) {
326                         bcopy(ccm_mac_p + out_data_1_len, out_data_2,
327                             block_size - out_data_1_len);
328                 }
329         }
330         out->cd_offset += ctx->ccm_remainder_len + ctx->ccm_mac_len;
331         ctx->ccm_remainder_len = 0;
332         return (CRYPTO_SUCCESS);
333 }
______unchanged_portion_omitted_

358 /*
359  * This will decrypt the cipher text.  However, the plaintext won't be
360  * returned to the caller.  It will be returned when decrypt_final() is
361  * called if the MAC matches
362  */
363 /* ARGSUSED */
364 int
365 ccm_mode_decrypt_contiguous_blocks(ccm_ctx_t *ctx, char *data, size_t length,
366     crypto_data_t *out, size_t block_size,
367     int (*encrypt_block)(const void *, const uint8_t *, uint8_t *),
368     void (*copy_block)(uint8_t *, uint8_t *),
369     void (*xor_block)(uint8_t *, uint8_t *))
370 {
371         size_t remainder = length;
372         size_t need;
373         uint8_t *datap = (uint8_t *)data;
374         uint8_t *blockp;
375         uint8_t *cbp;
376         uint64_t counter;
377         size_t pt_len, total_decrypted_len, mac_len, pm_len, pd_len;
378         uint8_t *resultp;
379 #ifdef _LITTLE_ENDIAN
380         uint8_t *p;
381 #endif  /* _LITTLE_ENDIAN */


384         pm_len = ctx->ccm_processed_mac_len;

386         if (pm_len > 0) {
387                 uint8_t *tmp;
388                 /*
389                  * all ciphertext has been processed, just waiting for
390                  * part of the value of the mac
391                  */
392                 if ((pm_len + length) > ctx->ccm_mac_len) {
393                         return (CRYPTO_ENCRYPTED_DATA_LEN_RANGE);
394                 }
395                 tmp = (uint8_t *)ctx->ccm_mac_input_buf;

397                 bcopy(datap, tmp + pm_len, length);

399                 ctx->ccm_processed_mac_len += length;
400                 return (CRYPTO_SUCCESS);
401         }

403         /*
404          * If we decrypt the given data, what total amount of data would
405          * have been decrypted?
406          */
407         pd_len = ctx->ccm_processed_data_len;
```

```
408         total_decrypted_len = pd_len + length + ctx->ccm_remainder_len;
410         if (total_decrypted_len >
411             (ctx->ccm_data_len + ctx->ccm_mac_len)) {
412                 return (CRYPTO_ENCRYPTED_DATA_LEN_RANGE);
413         }
415         pt_len = ctx->ccm_data_len;
417         if (total_decrypted_len > pt_len) {
418                 /*
419                  * part of the input will be the MAC, need to isolate that
420                  * to be dealt with later.  The left-over data in
421                  * ccm_remainder_len from last time will not be part of the
422                  * MAC.  Otherwise, it would have already been taken out
423                  * when this call is made last time.
424                  */
425                 size_t pt_part = pt_len - pd_len - ctx->ccm_remainder_len;
427                 mac_len = length - pt_part;
429                 ctx->ccm_processed_mac_len = mac_len;
430                 bcopy(data + pt_part, ctx->ccm_mac_input_buf, mac_len);
432                 if (pt_part + ctx->ccm_remainder_len < block_size) {
433                         /*
434                          * since this is last of the ciphertext, will
435                          * just decrypt with it here
436                          */
437                         bcopy(datap, &((uint8_t *)ctx->ccm_remainder)
438                             [ctx->ccm_remainder_len], pt_part);
439                         ctx->ccm_remainder_len += pt_part;
440                         ccm_decrypt_incomplete_block(ctx, encrypt_block);
441                         ctx->ccm_remainder_len = 0;
442                         ctx->ccm_processed_data_len += pt_part;
443                         return (CRYPTO_SUCCESS);
444                 } else {
445                         /* let rest of the code handle this */
446                         length = pt_part;
447                 }
448         } else if (length + ctx->ccm_remainder_len < block_size) {
449                         /* accumulate bytes here and return */
450                 bcopy(datap,
451                     (uint8_t *)ctx->ccm_remainder + ctx->ccm_remainder_len,
452                     length);
453                 ctx->ccm_remainder_len += length;
454                 ctx->ccm_copy_to = datap;
455                 return (CRYPTO_SUCCESS);
456         }
458         do {
459                 /* Unprocessed data from last call. */
460                 if (ctx->ccm_remainder_len > 0) {
461                         need = block_size - ctx->ccm_remainder_len;
463                         if (need > remainder)
464                                 return (CRYPTO_ENCRYPTED_DATA_LEN_RANGE);
466                         bcopy(datap, &((uint8_t *)ctx->ccm_remainder)
467                             [ctx->ccm_remainder_len], need);
469                         blockp = (uint8_t *)ctx->ccm_remainder;
470                 } else {
471                         blockp = datap;
472                 }
```



```
474                 /* Calculate the counter mode, ccm_cb is the counter block */
475                 cbp = (uint8_t *)ctx->ccm_tmp;
476                 encrypt_block(ctx->ccm_keysched, (uint8_t *)ctx->ccm_cb, cbp);
478                 /*
479                  * Increment counter.
480                  * Counter bits are confined to the bottom 64 bits
481                  */
482 #ifdef _LITTLE_ENDIAN
483                 counter = ntohll(ctx->ccm_cb[1] & ctx->ccm_counter_mask);
484                 counter = htonll(counter + 1);
485 #else
486                 counter = ctx->ccm_cb[1] & ctx->ccm_counter_mask;
498 #ifdef _LITTLE_ENDIAN
499                 p = (uint8_t *)&counter;
500                 counter = (((uint64_t)p[0] << 56) |
501                     ((uint64_t)p[1] << 48) |
502                     ((uint64_t)p[2] << 40) |
503                     ((uint64_t)p[3] << 32) |
504                     ((uint64_t)p[4] << 24) |
505                     ((uint64_t)p[5] << 16) |
506                     ((uint64_t)p[6] << 8) |
507                     (uint64_t)p[7]);
508 #endif
487                 counter++;
488 #endif  /* _LITTLE_ENDIAN */
510 #ifdef _LITTLE_ENDIAN
511                 counter = (((uint64_t)p[0] << 56) |
512                     ((uint64_t)p[1] << 48) |
513                     ((uint64_t)p[2] << 40) |
514                     ((uint64_t)p[3] << 32) |
515                     ((uint64_t)p[4] << 24) |
516                     ((uint64_t)p[5] << 16) |
517                     ((uint64_t)p[6] << 8) |
518                     (uint64_t)p[7]);
519 #endif
489                 counter &= ctx->ccm_counter_mask;
490                 ctx->ccm_cb[1] =
491                     (ctx->ccm_cb[1] & ~(ctx->ccm_counter_mask)) | counter;
493                 /* XOR with the ciphertext */
494                 xor_block(blockp, cbp);
496                 /* Copy the plaintext to the "holding buffer" */
497                 resultp = (uint8_t *)ctx->ccm_pt_buf +
498                     ctx->ccm_processed_data_len;
499                 copy_block(cbp, resultp);
501                 ctx->ccm_processed_data_len += block_size;
503                 ctx->ccm_lastp = blockp;
505                 /* Update pointer to next block of data to be processed. */
506                 if (ctx->ccm_remainder_len != 0) {
507                         datap += need;
508                         ctx->ccm_remainder_len = 0;
509                 } else {
510                         datap += block_size;
511                 }
513                 remainder = (size_t)&data[length] - (size_t)datap;
515                 /* Incomplete last block */
516                 if (remainder > 0 && remainder < block_size) {
517                         bcopy(datap, ctx->ccm_remainder, remainder);
518                         ctx->ccm_remainder_len = remainder;
```

```
519                         ctx->ccm_copy_to = datap;
520                         if (ctx->ccm_processed_mac_len > 0) {
521                                 /*
522                                  * not expecting anymore ciphertext, just
523                                  * compute plaintext for the remaining input
524                                  */
525                                 ccm_decrypt_incomplete_block(ctx,
526                                     encrypt_block);
527                                 ctx->ccm_processed_data_len += remainder;
528                                 ctx->ccm_remainder_len = 0;
529                         }
530                         goto out;
531                 }
532                 ctx->ccm_copy_to = NULL;

534         } while (remainder > 0);

536 out:
537         return (CRYPTO_SUCCESS);
538 }
```

unchanged_portion_omitted_

```
655 /*
656  * Format the first block used in CBC-MAC (B0) and the initial counter
657  * block based on formatting functions and counter generation functions
658  * specified in RFC 3610 and NIST publication 800-38C, appendix A
659  *
660  * b0 is the first block used in CBC-MAC
661  * cb0 is the first counter block
662  *
663  * It's assumed that the arguments b0 and cb0 are preallocated AES blocks
664  *
665  */
666 static void
667 ccm_format_initial_blocks(uchar_t *nonce, ulong_t nonceSize,
668     ulong_t authDataSize, uint8_t *b0, ccm_ctx_t *aes_ctx)
669 {
670         uint64_t payloadSize;
671         uint8_t t, q, have_adata = 0;
672         size_t limit;
673         int i, j, k;
674         uint64_t mask = 0;
675         uint8_t *cb;
707 #ifdef _LITTLE_ENDIAN
708         uint8_t *p8;
709 #endif  /* _LITTLE_ENDIAN */

677         q = (uint8_t)((15 - nonceSize) & 0xFF);
678         t = (uint8_t)((aes_ctx->ccm_mac_len) & 0xFF);

680         /* Construct the first octet of b0 */
681         if (authDataSize > 0) {
682                 have_adata = 1;
683         }
684         b0[0] = (have_adata << 6) | (((t - 2)  / 2) << 3) | (q - 1);

686         /* copy the nonce value into b0 */
687         bcopy(nonce, &(b0[1]), nonceSize);

689         /* store the length of the payload into b0 */
690         bzero(&(b0[1+nonceSize]), q);

692         payloadSize = aes_ctx->ccm_data_len;
693         limit = 8 < q ? 8 : q;

695         for (i = 0, j = 0, k = 15; i < limit; i++, j += 8, k--) {
```



```
696                 b0[k] = (uint8_t)((payloadSize >> j) & 0xFF);
697         }

699         /* format the counter block */

701         cb = (uint8_t *)aes_ctx->ccm_cb;

703         cb[0] = 0x07 & (q-1); /* first byte */

705         /* copy the nonce value into the counter block */
706         bcopy(nonce, &(cb[1]), nonceSize);

708         bzero(&(cb[1+nonceSize]), q);

710         /* Create the mask for the counter field based on the size of nonce */
711         q <<= 3;
712         while (q-- > 0) {
713                 mask |= (1ULL << q);
714         }

716 #ifdef _LITTLE_ENDIAN
717         mask = htonll(mask);
751         p8 = (uint8_t *)&mask;
752         mask = (((uint64_t)p8[0] << 56) |
753             ((uint64_t)p8[1] << 48) |
754             ((uint64_t)p8[2] << 40) |
755             ((uint64_t)p8[3] << 32) |
756             ((uint64_t)p8[4] << 24) |
757             ((uint64_t)p8[5] << 16) |
758             ((uint64_t)p8[6] << 8) |
759             (uint64_t)p8[7]);
718 #endif
719         aes_ctx->ccm_counter_mask = mask;

721         /*
722          * During calculation, we start using counter block 1, we will
723          * set it up right here.
724          * We can just set the last byte to have the value 1, because
725          * even with the biggest nonce of 13, the last byte of the
726          * counter block will be used for the counter value.
727          */
728         cb[15] = 0x01;
729 }

731 /*
732  * Encode the length of the associated data as
733  * specified in RFC 3610 and NIST publication 800-38C, appendix A
734  */
735 static void
736 encode_adata_len(ulong_t auth_data_len, uint8_t *encoded, size_t *encoded_len)
737 {
738 #ifdef UNALIGNED_POINTERS_PERMITTED
739         uint32_t        *lencoded_ptr;
740 #ifdef _LP64
741         uint64_t        *llencoded_ptr;
742 #endif
743 #endif  /* UNALIGNED_POINTERS_PERMITTED */

745         if (auth_data_len < ((1ULL<<16) - (1ULL<<8))) {
746                 /* 0 < a < (2^16-2^8) */
747                 *encoded_len = 2;
748                 encoded[0] = (auth_data_len & 0xff00) >> 8;
749                 encoded[1] = auth_data_len & 0xff;

751         } else if ((auth_data_len >= ((1ULL<<16) - (1ULL<<8))) &&
752             (auth_data_len < (1ULL << 31))) {
```

```
753                 /* (2^16-2^8) <= a < 2^32 */
754                 *encoded_len = 6;
755                 encoded[0] = 0xff;
756                 encoded[1] = 0xfe;
757 #ifdef UNALIGNED_POINTERS_PERMITTED
758                 lencoded_ptr = (uint32_t *)&encoded[2];
759                 *lencoded_ptr = htonl(auth_data_len);
760 #else
761                 encoded[2] = (auth_data_len & 0xff000000) >> 24;
762                 encoded[3] = (auth_data_len & 0xff0000) >> 16;
763                 encoded[4] = (auth_data_len & 0xff00) >> 8;
764                 encoded[5] = auth_data_len & 0xff;
765 #endif  /* UNALIGNED_POINTERS_PERMITTED */

767 #ifdef _LP64
768         } else {
769                 /* 2^32 <= a < 2^64 */
770                 *encoded_len = 10;
771                 encoded[0] = 0xff;
772                 encoded[1] = 0xff;
773 #ifdef UNALIGNED_POINTERS_PERMITTED
774                 llencoded_ptr = (uint64_t *)&encoded[2];
775                 *llencoded_ptr = htonl(auth_data_len);
776 #else
777                 encoded[2] = (auth_data_len & 0xff00000000000000) >> 56;
778                 encoded[3] = (auth_data_len & 0xff000000000000) >> 48;
779                 encoded[4] = (auth_data_len & 0xff0000000000) >> 40;
780                 encoded[5] = (auth_data_len & 0xff00000000) >> 32;
781                 encoded[6] = (auth_data_len & 0xff000000) >> 24;
782                 encoded[7] = (auth_data_len & 0xff0000) >> 16;
783                 encoded[8] = (auth_data_len & 0xff00) >> 8;
784                 encoded[9] = auth_data_len & 0xff;
785 #endif  /* UNALIGNED_POINTERS_PERMITTED */
786 #endif  /* _LP64 */
787         }
788 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    6176 Wed Aug 27 14:22:52 2008
new/usr/src/common/crypto/modes/ctr.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#ifndef _KERNEL
#include <strings.h>
#include <limits.h>
#include <assert.h>
#include <security/cryptoki.h>
#endif

#include <sys/types.h>
#include <modes/modes.h>
#include <sys/crypto/common.h>
#include <sys/crypto/impl.h>

#ifdef _LITTLE_ENDIAN
#include <sys/byteorder.h>
#endif

/*
 * Encrypt and decrypt multiple blocks of data in counter mode.
 */
int
ctr_mode_contiguous_blocks(ctr_ctx_t *ctx, char *data, size_t length,
    crypto_data_t *out, size_t block_size,
    int (*cipher)(const void *ks, const uint8_t *pt, uint8_t *ct),
    void (*xor_block)(uint8_t *, uint8_t *))
{
        size_t remainder = length;
        size_t need;
        uint8_t *datap = (uint8_t *)data;
        uint8_t *blockp;
        uint8_t *lastp;
        void *iov_or_mp;
        offset_t offset;
```



```
        uint8_t *out_data_1;
        uint8_t *out_data_2;
        size_t out_data_1_len;
        uint64_t counter;
#ifdef _LITTLE_ENDIAN
        uint8_t *p;
#endif

        if (length + ctx->ctr_remainder_len < block_size) {
                /* accumulate bytes here and return */
                bcopy(datap,
                    (uint8_t *)ctx->ctr_remainder + ctx->ctr_remainder_len,
                    length);
                ctx->ctr_remainder_len += length;
                ctx->ctr_copy_to = datap;
                return (CRYPTO_SUCCESS);
        }

        lastp = (uint8_t *)ctx->ctr_cb;
        if (out != NULL)
                crypto_init_ptrs(out, &iov_or_mp, &offset);

        do {
                /* Unprocessed data from last call. */
                if (ctx->ctr_remainder_len > 0) {
                        need = block_size - ctx->ctr_remainder_len;

                        if (need > remainder)
                                return (CRYPTO_DATA_LEN_RANGE);

                        bcopy(datap, &((uint8_t *)ctx->ctr_remainder)
                            [ctx->ctr_remainder_len], need);

                        blockp = (uint8_t *)ctx->ctr_remainder;
                } else {
                        blockp = datap;
                }

                /* ctr_cb is the counter block */
                cipher(ctx->ctr_keysched, (uint8_t *)ctx->ctr_cb,
                    (uint8_t *)ctx->ctr_tmp);

                lastp = (uint8_t *)ctx->ctr_tmp;

                /*
                 * Increment counter. Counter bits are confined
                 * to the bottom 64 bits of the counter block.
                 */
#ifdef _LITTLE_ENDIAN
                counter = ntohll(ctx->ctr_cb[1] & ctx->ctr_counter_mask);
                counter = htonll(counter + 1);
#else
                counter = ctx->ctr_cb[1] & ctx->ctr_counter_mask;
#ifdef _LITTLE_ENDIAN
                p = (uint8_t *)&counter;
                counter = (((uint64_t)p[0] << 56) |
                    ((uint64_t)p[1] << 48) |
                    ((uint64_t)p[2] << 40) |
                    ((uint64_t)p[3] << 32) |
                    ((uint64_t)p[4] << 24) |
                    ((uint64_t)p[5] << 16) |
                    ((uint64_t)p[6] << 8) |
                    (uint64_t)p[7]);
#endif
                counter++;
#endif  /* _LITTLE_ENDIAN */
```

```
#ifdef _LITTLE_ENDIAN
                counter = (((uint64_t)p[0] << 56) |
                    ((uint64_t)p[1] << 48) |
                    ((uint64_t)p[2] << 40) |
                    ((uint64_t)p[3] << 32) |
                    ((uint64_t)p[4] << 24) |
                    ((uint64_t)p[5] << 16) |
                    ((uint64_t)p[6] << 8) |
                    (uint64_t)p[7]);
#endif
                counter &= ctx->ctr_counter_mask;
                ctx->ctr_cb[1] =
                    (ctx->ctr_cb[1] & ~(ctx->ctr_counter_mask)) | counter;

                /*
                 * XOR the previous cipher block or IV with the
                 * current clear block.
                 */
                xor_block(blockp, lastp);

                if (out == NULL) {
                        if (ctx->ctr_remainder_len > 0) {
                                bcopy(lastp, ctx->ctr_copy_to,
                                    ctx->ctr_remainder_len);
                                bcopy(lastp + ctx->ctr_remainder_len, datap,
                                    need);
                        }
                } else {
                        crypto_get_ptrs(out, &iov_or_mp, &offset, &out_data_1,
                            &out_data_1_len, &out_data_2, block_size);

                        /* copy block to where it belongs */
                        bcopy(lastp, out_data_1, out_data_1_len);
                        if (out_data_2 != NULL) {
                                bcopy(lastp + out_data_1_len, out_data_2,
                                    block_size - out_data_1_len);
                        }
                        /* update offset */
                        out->cd_offset += block_size;
                }

                /* Update pointer to next block of data to be processed. */
                if (ctx->ctr_remainder_len != 0) {
                        datap += need;
                        ctx->ctr_remainder_len = 0;
                } else {
                        datap += block_size;
                }

                remainder = (size_t)&data[length] - (size_t)datap;

                /* Incomplete last block. */
                if (remainder > 0 && remainder < block_size) {
                        bcopy(datap, ctx->ctr_remainder, remainder);
                        ctx->ctr_remainder_len = remainder;
                        ctx->ctr_copy_to = datap;
                        goto out;
                }
                ctx->ctr_copy_to = NULL;

        } while (remainder > 0);

out:
        return (CRYPTO_SUCCESS);
}
```

unchanged_portion_omitted_


```
int
ctr_init_ctx(ctr_ctx_t *ctr_ctx, ulong_t count, uint8_t *cb,
void (*copy_block)(uint8_t *, uint8_t *))
{
        uint64_t mask = 0;
#ifdef _LITTLE_ENDIAN
        uint8_t *p8;
#endif

        if (count == 0 || count > 64) {
                return (CRYPTO_MECHANISM_PARAM_INVALID);
        }
        while (count-- > 0)
                mask |= (1ULL << count);

#ifdef _LITTLE_ENDIAN
        mask = htonll(mask);
        p8 = (uint8_t *)&mask;
        mask = (((uint64_t)p8[0] << 56) |
            ((uint64_t)p8[1] << 48) |
            ((uint64_t)p8[2] << 40) |
            ((uint64_t)p8[3] << 32) |
            ((uint64_t)p8[4] << 24) |
            ((uint64_t)p8[5] << 16) |
            ((uint64_t)p8[6] << 8) |
            (uint64_t)p8[7]);
#endif
        ctr_ctx->ctr_counter_mask = mask;
        copy_block(cb, (uchar_t *)ctr_ctx->ctr_cb);
        ctr_ctx->ctr_lastp = (uint8_t *)&ctr_ctx->ctr_cb[0];
        ctr_ctx->ctr_flags |= CTR_MODE;
        return (CRYPTO_SUCCESS);
}
```

unchanged_portion_omitted_

```
**********************************************************
   31438 Wed Aug 27 14:22:56 2008
new/usr/src/common/crypto/sha1/sha1.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * The basic framework for this code came from the reference
 * implementation for MD5.  That implementation is Copyright (C)
 * 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
 *
 * License to copy and use this software is granted provided that it
 * is identified as the "RSA Data Security, Inc. MD5 Message-Digest
 * Algorithm" in all material mentioning or referencing this software
 * or this function.
 *
 * License is also granted to make and use derivative works provided
 * that such works are identified as "derived from the RSA Data
 * Security, Inc. MD5 Message-Digest Algorithm" in all material
 * mentioning or referencing the derived work.
 *
 * RSA Data Security, Inc. makes no representations concerning either
 * the merchantability of this software or the suitability of this
 * software for any particular purpose. It is provided "as is"
 * without express or implied warranty of any kind.
 *
 * These notices must be retained in any copies of any part of this
 * documentation and/or software.
 *
 * NOTE: Cleaned-up and optimized, version of SHA1, based on the FIPS 180-1
 * standard, available at http://www.itl.nist.gov/fipspubs/fip180-1.htm
 * standard, available at http://www.itl.nist.gov/div897/pubs/fip180-1.htm
 * Not as fast as one would like -- further optimizations are encouraged
 * and appreciated.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/sysmacros.h>
#include <sys/sha1.h>
#include <sys/sha1_consts.h>

#ifndef _KERNEL
#include <strings.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/systeminfo.h>
#endif  /* !_KERNEL */

#ifdef _LITTLE_ENDIAN
#include <sys/byteorder.h>
#define HAVE_HTONL
#endif

static void Encode(uint8_t *, const uint32_t *, size_t);

#if     defined(__sparc)
```



```
#define SHA1_TRANSFORM(ctx, in) \
        SHA1Transform((ctx)->state[0], (ctx)->state[1], (ctx)->state[2], \
                (ctx)->state[3], (ctx)->state[4], (ctx), (in))

static void SHA1Transform(uint32_t, uint32_t, uint32_t, uint32_t, uint32_t,
    SHA1_CTX *, const uint8_t *);

#elif   defined(__amd64)

#define SHA1_TRANSFORM(ctx, in) sha1_block_data_order((ctx), (in), 1)
#define SHA1_TRANSFORM_BLOCKS(ctx, in, num) sha1_block_data_order((ctx), \
                (in), (num))

void sha1_block_data_order(SHA1_CTX *ctx, const void *inpp, size_t num_blocks);

#else

#define SHA1_TRANSFORM(ctx, in) SHA1Transform((ctx), (in))

static void SHA1Transform(SHA1_CTX *, const uint8_t *);

#endif


static uint8_t PADDING[64] = { 0x80, /* all zeros */ };

/*
 * F, G, and H are the basic SHA1 functions.
 */
#define F(b, c, d)      (((b) & (c)) | ((~b) & (d)))
#define G(b, c, d)      ((b) ^ (c) ^ (d))
#define H(b, c, d)      (((b) & (c)) | (((b)|(c)) & (d)))

/*
 * ROTATE_LEFT rotates x left n bits.
 */

#if     defined(__GNUC__) && defined(_LP64)
static __inline__ uint64_t
ROTATE_LEFT(uint64_t value, uint32_t n)
{
        uint32_t t32;

        t32 = (uint32_t)value;
        return ((t32 << n) | (t32 >> (32 - n)));
}

#else

#define ROTATE_LEFT(x, n)       \
        (((x) << (n)) | ((x) >> ((sizeof (x) * NBBY)-(n))))

#endif

#if     defined(__GNUC__) && (defined(__i386) || defined(__amd64))

#define HAVE_BSWAP

extern __inline__ uint32_t bswap(uint32_t value)
{
        __asm__("bswap %0" : "+r" (value));
        return (value);
}

#endif
```

```
113 /*
114  * SHA1Init()
115  *
116  * purpose: initializes the sha1 context and begins and sha1 digest operation
117  *   input: SHA1_CTX *  : the context to initializes.
118  *  output: void
119  */

121 void
122 SHA1Init(SHA1_CTX *ctx)
123 {
124         ctx->count[0] = ctx->count[1] = 0;

126         /*
127          * load magic initialization constants. Tell lint
128          * that these constants are unsigned by using U.
129          */

131         ctx->state[0] = 0x67452301U;
132         ctx->state[1] = 0xefcdab89U;
133         ctx->state[2] = 0x98badcfeU;
134         ctx->state[3] = 0x10325476U;
135         ctx->state[4] = 0xc3d2e1f0U;
136 }

138 #ifdef VIS_SHA1
139 #ifdef _KERNEL

141 #include <sys/regset.h>
142 #include <sys/vis.h>
143 #include <sys/fpu/fpusystm.h>

145 /* the alignment for block stores to save fp registers */
146 #define VIS_ALIGN       (64)

148 extern int sha1_savefp(kfpu_t *, int);
149 extern void sha1_restorefp(kfpu_t *);

151 uint32_t        vis_sha1_svfp_threshold = 128;

153 #endif /* _KERNEL */

155 /*
156  * VIS SHA-1 consts.
157  */
158 static uint64_t VIS[] = {
159         0x8000000080000000ULL,
160         0x0002000200020002ULL,
161         0x5a8279996ed9eba1ULL,
162         0x8f1bbcdcca62c1d6ULL,
163         0x012389ab456789abULL};

165 extern void SHA1TransformVIS(uint64_t *, uint32_t *, uint32_t *, uint64_t *);


168 /*
169  * SHA1Update()
170  *
171  * purpose: continues an sha1 digest operation, using the message block
172  *          to update the context.
173  *   input: SHA1_CTX *  : the context to update
174  *          void *      : the message block
175  *          size_t    : the length of the message block in bytes
176  *  output: void
177  */
```



```
179 void
180 SHA1Update(SHA1_CTX *ctx, const void *inptr, size_t input_len)
181 {
182         uint32_t i, buf_index, buf_len;
183         uint64_t X0[40], input64[8];
184         const uint8_t *input = inptr;
185 #ifdef _KERNEL
186         int usevis = 0;
187 #else
188         int usevis = 1;
189 #endif /* _KERNEL */

191         /* check for noop */
192         if (input_len == 0)
193                 return;

195         /* compute number of bytes mod 64 */
196         buf_index = (ctx->count[1] >> 3) & 0x3F;

198         /* update number of bits */
199         if ((ctx->count[1] += (input_len << 3)) < (input_len << 3))
200                 ctx->count[0]++;

202         ctx->count[0] += (input_len >> 29);

204         buf_len = 64 - buf_index;

206         /* transform as many times as possible */
207         i = 0;
208         if (input_len >= buf_len) {
209 #ifdef _KERNEL
210                 kfpu_t *fpu;
211                 if (fpu_exists) {
212                         uint8_t fpua[sizeof (kfpu_t) + GSR_SIZE + VIS_ALIGN];
213                         uint32_t len = (input_len + buf_index) & ~0x3f;
214                         int svfp_ok;

216                         fpu = (kfpu_t *)P2ROUNDUP((uintptr_t)fpua, 64);
217                         svfp_ok = ((len >= vis_sha1_svfp_threshold) ? 1 : 0);
218                         usevis = fpu_exists && sha1_savefp(fpu, svfp_ok);
219                 } else {
220                         usevis = 0;
221                 }
222 #endif /* _KERNEL */

224                 /*
225                  * general optimization:
226                  *
227                  * only do initial bcopy() and SHA1Transform() if
228                  * buf_index != 0.  if buf_index == 0, we're just
229                  * wasting our time doing the bcopy() since there
230                  * wasn't any data left over from a previous call to
231                  * SHA1Update().
232                  */

234                 if (buf_index) {
235                         bcopy(input, &ctx->buf_un.buf8[buf_index], buf_len);
236                         if (usevis) {
237                                 SHA1TransformVIS(X0,
238                                     ctx->buf_un.buf32,
239                                     &ctx->state[0], VIS);
240                         } else {
241                                 SHA1_TRANSFORM(ctx, ctx->buf_un.buf8);
242                         }
243                         i = buf_len;
```

```
244                 }
246                 /*
247                  * VIS SHA-1: uses the VIS 1.0 instructions to accelerate
248                  * SHA-1 processing. This is achieved by "offloading" the
249                  * computation of the message schedule (MS) to the VIS units.
250                  * This allows the VIS computation of the message schedule
251                  * to be performed in parallel with the standard integer
252                  * processing of the remainder of the SHA-1 computation.
253                  * performance by up to around 1.37X, compared to an optimized
254                  * integer-only implementation.
255                  *
256                  * The VIS implementation of SHA1Transform has a different API
257                  * to the standard integer version:
258                  *
259                  * void SHA1TransformVIS(
260                  *       uint64_t *, // Pointer to MS for ith block
261                  *       uint32_t *, // Pointer to ith block of message data
262                  *       uint32_t *, // Pointer to SHA state i.e ctx->state
263                  *       uint64_t *, // Pointer to various VIS constants
264                  * )
265                  *
266                  * Note: the message data must by 4-byte aligned.
267                  *
268                  * Function requires VIS 1.0 support.
269                  *
270                  * Handling is provided to deal with arbitrary byte alingment
271                  * of the input data but the performance gains are reduced
272                  * for alignments other than 4-bytes.
273                  */
274                 if (usevis) {
275                         if (!IS_P2ALIGNED(&input[i], sizeof (uint32_t))) {
276                                 /*
277                                  * Main processing loop - input misaligned
278                                  */
279                                 for (; i + 63 < input_len; i += 64) {
280                                         bcopy(&input[i], input64, 64);
281                                         SHA1TransformVIS(X0,
282                                             (uint32_t *)input64,
283                                             &ctx->state[0], VIS);
284                                 }
285                         } else {
286                                 /*
287                                  * Main processing loop - input 8-byte aligned
288                                  */
289                                 for (; i + 63 < input_len; i += 64) {
290                                         SHA1TransformVIS(X0,
291                                             /* LINTED E_BAD_PTR_CAST_ALIGN */
292                                             (uint32_t *)&input[i], /* CSTYLED */
300                                             (uint32_t *)&input[i],
293                                             &ctx->state[0], VIS);
294                                 }
296                         }
297 #ifdef _KERNEL
298                         sha1_restorefp(fpu);
299 #endif /* _KERNEL */
300                 } else {
301                         for (; i + 63 < input_len; i += 64) {
302                                 SHA1_TRANSFORM(ctx, &input[i]);
303                         }
304                 }
306                 /*
307                  * general optimization:
308                  *
```



```
309                  * if i and input_len are the same, return now instead
310                  * of calling bcopy(), since the bcopy() in this case
311                  * will be an expensive nop.
312                  */
314                 if (input_len == i)
315                         return;
317                 buf_index = 0;
318         }
320         /* buffer remaining input */
321         bcopy(&input[i], &ctx->buf_un.buf8[buf_index], input_len - i);
322 }
_____unchanged_portion_omitted_
436 #if !defined(__amd64)
438 typedef uint32_t sha1word;
440 /*
441  * sparc optimization:
442  *
443  * on the sparc, we can load big endian 32-bit data easily.  note that
444  * special care must be taken to ensure the address is 32-bit aligned.
445  * in the interest of speed, we don't check to make sure, since
446  * careful programming can guarantee this for us.
447  */
449 #if     defined(_BIG_ENDIAN)
450 #define LOAD_BIG_32(addr)       (*(uint32_t *)(addr))
452 #elif   defined(HAVE_HTONL)
453 #define LOAD_BIG_32(addr) htonl(*((uint32_t *)(addr)))
461 #else   /* !defined(_BIG_ENDIAN) */
455 #else
463 #if     defined(HAVE_BSWAP)
465 #define LOAD_BIG_32(addr) bswap(*((uint32_t *)(addr)))
467 #else   /* !defined(HAVE_BSWAP) */
456 /* little endian -- will work on big endian, but slowly */
457 #define LOAD_BIG_32(addr)       \
458         (((addr)[0] << 24) | ((addr)[1] << 16) | ((addr)[2] << 8) | (addr)[3])
459 #endif  /* _BIG_ENDIAN */
473 #endif  /* !defined(HAVE_BSWAP) */
475 #endif  /* !defined(_BIG_ENDIAN) */
461 /*
462  * SHA1Transform()
463  */
464 #if     defined(W_ARRAY)
465 #define W(n) w[n]
466 #else   /* !defined(W_ARRAY) */
467 #define W(n) w_ ## n
468 #endif  /* !defined(W_ARRAY) */
471 #if     defined(__sparc)
```

```
473 /*
474  * sparc register window optimization:
475  *
476  * ‘a’, ‘b’, ‘c’, ‘d’, and ‘e’ are passed into SHA1Transform
477  * explicitly since it increases the number of registers available to
478  * the compiler.  under this scheme, these variables can be held in
479  * %i0 - %i4, which leaves more local and out registers available.
480  *
481  * purpose: sha1 transformation -- updates the digest based on ‘block’
482  *   input: uint32_t    : bytes  1 -  4 of the digest
483  *          uint32_t    : bytes  5 -  8 of the digest
484  *          uint32_t    : bytes  9 - 12 of the digest
485  *          uint32_t    : bytes 12 - 16 of the digest
486  *          uint32_t    : bytes 16 - 20 of the digest
487  *          SHA1_CTX *  : the context to update
488  *          uint8_t [64]: the block to use to update the digest
489  *  output: void
490  */

492 void
493 SHA1Transform(uint32_t a, uint32_t b, uint32_t c, uint32_t d, uint32_t e,
494     SHA1_CTX *ctx, const uint8_t blk[64])
495 {
496         /*
497          * sparc optimization:
498          *
499          * while it is somewhat counter-intuitive, on sparc, it is
500          * more efficient to place all the constants used in this
501          * function in an array and load the values out of the array
502          * than to manually load the constants.  this is because
503          * setting a register to a 32-bit value takes two ops in most
504          * cases: a ‘sethi’ and an ‘or’, but loading a 32-bit value
505          * from memory only takes one ‘ld’ (or ‘lduw’ on v9).  while
506          * this increases memory usage, the compiler can find enough
507          * other things to do while waiting to keep the pipeline does
508          * not stall.  additionally, it is likely that many of these
509          * constants are cached so that later accesses do not even go
510          * out to the bus.
511          *
512          * this array is declared ‘static’ to keep the compiler from
513          * having to bcopy() this array onto the stack frame of
514          * SHA1Transform() each time it is called -- which is
515          * unacceptably expensive.
516          *
517          * the ‘const’ is to ensure that callers are good citizens and
518          * do not try to munge the array.  since these routines are
519          * going to be called from inside multithreaded kernelland,
520          * this is a good safety check. -- ‘sha1_consts’ will end up in
521          * .rodata.
522          *
523          * unfortunately, loading from an array in this manner hurts
524          * performance under Intel.  So, there is a macro,
540          * performance under intel.  so, there is a macro,
525          * SHA1_CONST(), used in SHA1Transform(), that either expands to
526          * a reference to this array, or to the actual constant,
527          * depending on what platform this code is compiled for.
528          */

530         static const uint32_t sha1_consts[] = {
531                 SHA1_CONST_0, SHA1_CONST_1, SHA1_CONST_2, SHA1_CONST_3
547                 SHA1_CONST_0,   SHA1_CONST_1,   SHA1_CONST_2,   SHA1_CONST_3,
532         };

534         /*
535          * general optimization:
536          *
```



```
537          * use individual integers instead of using an array.  this is a
538          * win, although the amount it wins by seems to vary quite a bit.
539          */

541         uint32_t        w_0, w_1, w_2,  w_3,  w_4,  w_5,  w_6,  w_7;
542         uint32_t        w_8, w_9, w_10, w_11, w_12, w_13, w_14, w_15;

544         /*
545          * sparc optimization:
546          *
547          * if ‘block’ is already aligned on a 4-byte boundary, use
548          * LOAD_BIG_32() directly.  otherwise, bcopy() into a
549          * buffer that *is* aligned on a 4-byte boundary and then do
550          * the LOAD_BIG_32() on that buffer.  benchmarks have shown
551          * that using the bcopy() is better than loading the bytes
552          * individually and doing the endian-swap by hand.
553          *
554          * even though it’s quite tempting to assign to do:
555          *
556          * blk = bcopy(ctx->buf_un.buf32, blk, sizeof (ctx->buf_un.buf32));
557          *
558          * and only have one set of LOAD_BIG_32()’s, the compiler
559          * *does not* like that, so please resist the urge.
560          */

562         if ((uintptr_t)blk & 0x3) {             /* not 4-byte aligned? */
563                 bcopy(blk, ctx->buf_un.buf32,  sizeof (ctx->buf_un.buf32));
564                 w_15 = LOAD_BIG_32(ctx->buf_un.buf32 + 15);
565                 w_14 = LOAD_BIG_32(ctx->buf_un.buf32 + 14);
566                 w_13 = LOAD_BIG_32(ctx->buf_un.buf32 + 13);
567                 w_12 = LOAD_BIG_32(ctx->buf_un.buf32 + 12);
568                 w_11 = LOAD_BIG_32(ctx->buf_un.buf32 + 11);
569                 w_10 = LOAD_BIG_32(ctx->buf_un.buf32 + 10);
570                 w_9  = LOAD_BIG_32(ctx->buf_un.buf32 +  9);
571                 w_8  = LOAD_BIG_32(ctx->buf_un.buf32 +  8);
572                 w_7  = LOAD_BIG_32(ctx->buf_un.buf32 +  7);
573                 w_6  = LOAD_BIG_32(ctx->buf_un.buf32 +  6);
574                 w_5  = LOAD_BIG_32(ctx->buf_un.buf32 +  5);
575                 w_4  = LOAD_BIG_32(ctx->buf_un.buf32 +  4);
576                 w_3  = LOAD_BIG_32(ctx->buf_un.buf32 +  3);
577                 w_2  = LOAD_BIG_32(ctx->buf_un.buf32 +  2);
578                 w_1  = LOAD_BIG_32(ctx->buf_un.buf32 +  1);
579                 w_0  = LOAD_BIG_32(ctx->buf_un.buf32 +  0);
580         } else {
581                 /*LINTED*/
582                 w_15 = LOAD_BIG_32(blk + 60);
583                 /*LINTED*/
584                 w_14 = LOAD_BIG_32(blk + 56);
585                 /*LINTED*/
586                 w_13 = LOAD_BIG_32(blk + 52);
587                 /*LINTED*/
588                 w_12 = LOAD_BIG_32(blk + 48);
589                 /*LINTED*/
590                 w_11 = LOAD_BIG_32(blk + 44);
591                 /*LINTED*/
592                 w_10 = LOAD_BIG_32(blk + 40);
593                 /*LINTED*/
594                 w_9  = LOAD_BIG_32(blk + 36);
595                 /*LINTED*/
596                 w_8  = LOAD_BIG_32(blk + 32);
597                 /*LINTED*/
598                 w_7  = LOAD_BIG_32(blk + 28);
599                 /*LINTED*/
600                 w_6  = LOAD_BIG_32(blk + 24);
601                 /*LINTED*/
602                 w_5  = LOAD_BIG_32(blk + 20);
```

```
603                 /*LINTED*/
604                 w_4  = LOAD_BIG_32(blk + 16);
605                 /*LINTED*/
606                 w_3  = LOAD_BIG_32(blk + 12);
607                 /*LINTED*/
608                 w_2  = LOAD_BIG_32(blk +  8);
609                 /*LINTED*/
610                 w_1  = LOAD_BIG_32(blk +  4);
611                 /*LINTED*/
612                 w_0  = LOAD_BIG_32(blk +  0);
613         }
614 #else   /* !defined(__sparc) */

616 void /* CSTYLED */
632 void
617 SHA1Transform(SHA1_CTX *ctx, const uint8_t blk[64])
618 {
619         /* CSTYLED */
620         sha1word a = ctx->state[0];
621         sha1word b = ctx->state[1];
622         sha1word c = ctx->state[2];
623         sha1word d = ctx->state[3];
624         sha1word e = ctx->state[4];

626 #if     defined(W_ARRAY)
627         sha1word        w[16];
628 #else   /* !defined(W_ARRAY) */
629         sha1word        w_0, w_1, w_2,  w_3,  w_4,  w_5,  w_6,  w_7;
630         sha1word        w_8, w_9, w_10, w_11, w_12, w_13, w_14, w_15;
631 #endif  /* !defined(W_ARRAY) */

633         W(0)  = LOAD_BIG_32(blk +  0);
634         W(1)  = LOAD_BIG_32(blk +  4);
635         W(2)  = LOAD_BIG_32(blk +  8);
636         W(3)  = LOAD_BIG_32(blk + 12);
637         W(4)  = LOAD_BIG_32(blk + 16);
638         W(5)  = LOAD_BIG_32(blk + 20);
639         W(6)  = LOAD_BIG_32(blk + 24);
640         W(7)  = LOAD_BIG_32(blk + 28);
641         W(8)  = LOAD_BIG_32(blk + 32);
642         W(9)  = LOAD_BIG_32(blk + 36);
643         W(10) = LOAD_BIG_32(blk + 40);
644         W(11) = LOAD_BIG_32(blk + 44);
645         W(12) = LOAD_BIG_32(blk + 48);
646         W(13) = LOAD_BIG_32(blk + 52);
647         W(14) = LOAD_BIG_32(blk + 56);
648         W(15) = LOAD_BIG_32(blk + 60);

650 #endif  /* !defined(__sparc) */

652         /*
653          * general optimization:
654          *
655          * even though this approach is described in the standard as
656          * being slower algorithmically, it is 30-40% faster than the
657          * "faster" version under SPARC, because this version has more
658          * of the constraints specified at compile-time and uses fewer
659          * variables (and therefore has better register utilization)
660          * than its "speedier" brother.  (i've tried both, trust me)
661          *
662          * for either method given in the spec, there is an "assignment"
663          * phase where the following takes place:
664          *
665          *      tmp = (main_computation);
666          *      e = d; d = c; c = rotate_left(b, 30); b = a; a = tmp;
667          *
```



```
668          * we can make the algorithm go faster by not doing this work,
669          * but just pretending that 'd' is now 'e', etc. this works
670          * really well and obviates the need for a temporary variable.
671          * however, we still explicitly perform the rotate action,
672          * since it is cheaper on SPARC to do it once than to have to
673          * do it over and over again.
674          */

676         /* round 1 */
677         e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(0) + SHA1_CONST(0); /* 0 */
678         b = ROTATE_LEFT(b, 30);

680         d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(1) + SHA1_CONST(0); /* 1 */
681         a = ROTATE_LEFT(a, 30);

683         c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(2) + SHA1_CONST(0); /* 2 */
684         e = ROTATE_LEFT(e, 30);

686         b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(3) + SHA1_CONST(0); /* 3 */
687         d = ROTATE_LEFT(d, 30);

689         a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(4) + SHA1_CONST(0); /* 4 */
690         c = ROTATE_LEFT(c, 30);

692         e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(5) + SHA1_CONST(0); /* 5 */
693         b = ROTATE_LEFT(b, 30);

695         d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(6) + SHA1_CONST(0); /* 6 */
696         a = ROTATE_LEFT(a, 30);

698         c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(7) + SHA1_CONST(0); /* 7 */
699         e = ROTATE_LEFT(e, 30);

701         b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(8) + SHA1_CONST(0); /* 8 */
702         d = ROTATE_LEFT(d, 30);

704         a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(9) + SHA1_CONST(0); /* 9 */
705         c = ROTATE_LEFT(c, 30);

707         e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(10) + SHA1_CONST(0); /* 10 */
708         b = ROTATE_LEFT(b, 30);

710         d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(11) + SHA1_CONST(0); /* 11 */
711         a = ROTATE_LEFT(a, 30);

713         c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(12) + SHA1_CONST(0); /* 12 */
714         e = ROTATE_LEFT(e, 30);

716         b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(13) + SHA1_CONST(0); /* 13 */
717         d = ROTATE_LEFT(d, 30);

719         a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(14) + SHA1_CONST(0); /* 14 */
720         c = ROTATE_LEFT(c, 30);

722         e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(15) + SHA1_CONST(0); /* 15 */
723         b = ROTATE_LEFT(b, 30);

725         W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1);            /* 16 */
726         d = ROTATE_LEFT(e, 5) + F(a, b, c) + d + W(0) + SHA1_CONST(0);
727         a = ROTATE_LEFT(a, 30);

729         W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1);            /* 17 */
730         c = ROTATE_LEFT(d, 5) + F(e, a, b) + c + W(1) + SHA1_CONST(0);
731         e = ROTATE_LEFT(e, 30);

733         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 18 */
```

```
734         b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(2) + SHA1_CONST(0);
735         d = ROTATE_LEFT(d, 30);
736
737         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 19 */
738         a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(3) + SHA1_CONST(0);
739         c = ROTATE_LEFT(c, 30);
740
741         /* round 2 */
742         W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1);           /* 20 */
743         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(4) + SHA1_CONST(1);
744         b = ROTATE_LEFT(b, 30);
745
746         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 21 */
747         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(5) + SHA1_CONST(1);
748         a = ROTATE_LEFT(a, 30);
749
750         W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1);           /* 22 */
751         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(6) + SHA1_CONST(1);
752         e = ROTATE_LEFT(e, 30);
753
754         W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1);           /* 23 */
755         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(7) + SHA1_CONST(1);
756         d = ROTATE_LEFT(d, 30);
757
758         W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1);           /* 24 */
759         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(8) + SHA1_CONST(1);
760         c = ROTATE_LEFT(c, 30);
761
762         W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1);           /* 25 */
763         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(9) + SHA1_CONST(1);
764         b = ROTATE_LEFT(b, 30);
765
766         W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1);  /* 26 */
767         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(10) + SHA1_CONST(1);
768         a = ROTATE_LEFT(a, 30);
769
770         W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1);  /* 27 */
771         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(11) + SHA1_CONST(1);
772         e = ROTATE_LEFT(e, 30);
773
774         W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1);  /* 28 */
775         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(12) + SHA1_CONST(1);
776         d = ROTATE_LEFT(d, 30);
777
778         W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 29 */
779         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(13) + SHA1_CONST(1);
780         c = ROTATE_LEFT(c, 30);
781
782         W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1);  /* 30 */
783         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(14) + SHA1_CONST(1);
784         b = ROTATE_LEFT(b, 30);
785
786         W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1);  /* 31 */
787         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(15) + SHA1_CONST(1);
788         a = ROTATE_LEFT(a, 30);
789
790         W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1);           /* 32 */
791         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(0) + SHA1_CONST(1);
792         e = ROTATE_LEFT(e, 30);
793
794         W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1);           /* 33 */
795         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(1) + SHA1_CONST(1);
796         d = ROTATE_LEFT(d, 30);
797
798         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 34 */
799         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(2) + SHA1_CONST(1);
```



```
800         c = ROTATE_LEFT(c, 30);
801
802         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 35 */
803         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(3) + SHA1_CONST(1);
804         b = ROTATE_LEFT(b, 30);
805
806         W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1);           /* 36 */
807         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(4) + SHA1_CONST(1);
808         a = ROTATE_LEFT(a, 30);
809
810         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 37 */
811         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(5) + SHA1_CONST(1);
812         e = ROTATE_LEFT(e, 30);
813
814         W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1);           /* 38 */
815         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(6) + SHA1_CONST(1);
816         d = ROTATE_LEFT(d, 30);
817
818         W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1);           /* 39 */
819         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(7) + SHA1_CONST(1);
820         c = ROTATE_LEFT(c, 30);
821
822         /* round 3 */
823         W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1);           /* 40 */
824         e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + W(8) + SHA1_CONST(2);
825         b = ROTATE_LEFT(b, 30);
826
827         W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1);           /* 41 */
828         d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + W(9) + SHA1_CONST(2);
829         a = ROTATE_LEFT(a, 30);
830
831         W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1);  /* 42 */
832         c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + W(10) + SHA1_CONST(2);
833         e = ROTATE_LEFT(e, 30);
834
835         W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1);  /* 43 */
836         b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + W(11) + SHA1_CONST(2);
837         d = ROTATE_LEFT(d, 30);
838
839         W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1);  /* 44 */
840         a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + W(12) + SHA1_CONST(2);
841         c = ROTATE_LEFT(c, 30);
842
843         W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 45 */
844         e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + W(13) + SHA1_CONST(2);
845         b = ROTATE_LEFT(b, 30);
846
847         W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1);  /* 46 */
848         d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + W(14) + SHA1_CONST(2);
849         a = ROTATE_LEFT(a, 30);
850
851         W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1);  /* 47 */
852         c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + W(15) + SHA1_CONST(2);
853         e = ROTATE_LEFT(e, 30);
854
855         W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1);           /* 48 */
856         b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + W(0) + SHA1_CONST(2);
857         d = ROTATE_LEFT(d, 30);
858
859         W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1);           /* 49 */
860         a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + W(1) + SHA1_CONST(2);
861         c = ROTATE_LEFT(c, 30);
862
863         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 50 */
864         e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + W(2) + SHA1_CONST(2);
865         b = ROTATE_LEFT(b, 30);
```

```
867         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 51 */
868         d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + W(3) + SHA1_CONST(2);
869         a = ROTATE_LEFT(a, 30);

871         W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1);           /* 52 */
872         c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + W(4) + SHA1_CONST(2);
873         e = ROTATE_LEFT(e, 30);

875         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 53 */
876         b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + W(5) + SHA1_CONST(2);
877         d = ROTATE_LEFT(d, 30);

879         W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1);           /* 54 */
880         a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + W(6) + SHA1_CONST(2);
881         c = ROTATE_LEFT(c, 30);

883         W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1);           /* 55 */
884         e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + W(7) + SHA1_CONST(2);
885         b = ROTATE_LEFT(b, 30);

887         W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1);           /* 56 */
888         d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + W(8) + SHA1_CONST(2);
889         a = ROTATE_LEFT(a, 30);

891         W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1);           /* 57 */
892         c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + W(9) + SHA1_CONST(2);
893         e = ROTATE_LEFT(e, 30);

895         W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1);  /* 58 */
896         b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + W(10) + SHA1_CONST(2);
897         d = ROTATE_LEFT(d, 30);

899         W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1);  /* 59 */
900         a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + W(11) + SHA1_CONST(2);
901         c = ROTATE_LEFT(c, 30);

903         /* round 4 */
904         W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1);  /* 60 */
905         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(12) + SHA1_CONST(3);
906         b = ROTATE_LEFT(b, 30);

908         W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 61 */
909         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(13) + SHA1_CONST(3);
910         a = ROTATE_LEFT(a, 30);

912         W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1);  /* 62 */
913         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(14) + SHA1_CONST(3);
914         e = ROTATE_LEFT(e, 30);

916         W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1);  /* 63 */
917         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(15) + SHA1_CONST(3);
918         d = ROTATE_LEFT(d, 30);

920         W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1);           /* 64 */
921         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(0) + SHA1_CONST(3);
922         c = ROTATE_LEFT(c, 30);

924         W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1);           /* 65 */
925         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(1) + SHA1_CONST(3);
926         b = ROTATE_LEFT(b, 30);

928         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 66 */
929         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(2) + SHA1_CONST(3);
930         a = ROTATE_LEFT(a, 30);
```



```
932         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 67 */
933         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(3) + SHA1_CONST(3);
934         e = ROTATE_LEFT(e, 30);

936         W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1);           /* 68 */
937         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(4) + SHA1_CONST(3);
938         d = ROTATE_LEFT(d, 30);

940         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 69 */
941         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(5) + SHA1_CONST(3);
942         c = ROTATE_LEFT(c, 30);

944         W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1);           /* 70 */
945         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(6) + SHA1_CONST(3);
946         b = ROTATE_LEFT(b, 30);

948         W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1);           /* 71 */
949         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(7) + SHA1_CONST(3);
950         a = ROTATE_LEFT(a, 30);

952         W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1);           /* 72 */
953         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(8) + SHA1_CONST(3);
954         e = ROTATE_LEFT(e, 30);

956         W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1);           /* 73 */
957         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(9) + SHA1_CONST(3);
958         d = ROTATE_LEFT(d, 30);

960         W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1);  /* 74 */
961         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(10) + SHA1_CONST(3);
962         c = ROTATE_LEFT(c, 30);

964         W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1);  /* 75 */
965         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(11) + SHA1_CONST(3);
966         b = ROTATE_LEFT(b, 30);

968         W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1);  /* 76 */
969         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(12) + SHA1_CONST(3);
970         a = ROTATE_LEFT(a, 30);

972         W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 77 */
973         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(13) + SHA1_CONST(3);
974         e = ROTATE_LEFT(e, 30);

976         W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1);  /* 78 */
977         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(14) + SHA1_CONST(3);
978         d = ROTATE_LEFT(d, 30);

980         W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1);  /* 79 */

982         ctx->state[0] += ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(15) +
983             SHA1_CONST(3);
984         ctx->state[1] += b;
985         ctx->state[2] += ROTATE_LEFT(c, 30);
986         ctx->state[3] += d;
987         ctx->state[4] += e;

989         /* zeroize sensitive information */
990         W(0) = W(1) = W(2) = W(3) = W(4) = W(5) = W(6) = W(7) = W(8) = 0;
991         W(9) = W(10) = W(11) = W(12) = W(13) = W(14) = W(15) = 0;
992 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   31444 Wed Aug 27 14:23:01 2008
new/usr/src/common/crypto/sha2/sha2.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * The basic framework for this code came from the reference
 * implementation for MD5.  That implementation is Copyright (C)
 * 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
 *
 * License to copy and use this software is granted provided that it
 * is identified as the "RSA Data Security, Inc. MD5 Message-Digest
 * Algorithm" in all material mentioning or referencing this software
 * or this function.
 *
 * License is also granted to make and use derivative works provided
 * that such works are identified as "derived from the RSA Data
 * Security, Inc. MD5 Message-Digest Algorithm" in all material
 * mentioning or referencing the derived work.
 *
 * RSA Data Security, Inc. makes no representations concerning either
 * the merchantability of this software or the suitability of this
 * software for any particular purpose. It is provided "as is"
 * without express or implied warranty of any kind.
 *
 * These notices must be retained in any copies of any part of this
 * documentation and/or software.
 *
 * NOTE: Cleaned-up and optimized, version of SHA2, based on the FIPS 180-2
 * standard, available at
 * http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
 * standard, available at http://www.itl.nist.gov/div897/pubs/fip180-2.htm
 * Not as fast as one would like -- further optimizations are encouraged
 * and appreciated.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/sysmacros.h>
#define _SHA2_IMPL
#include <sys/sha2.h>
#include <sys/sha2_consts.h>

#ifdef _KERNEL
#include <sys/cmn_err.h>

#else
#include <strings.h>
#include <stdlib.h>
#include <errno.h>

#pragma weak SHA256Update = SHA2Update
#pragma weak SHA384Update = SHA2Update
#pragma weak SHA512Update = SHA2Update

#pragma weak SHA256Final = SHA2Final
```



```
#pragma weak SHA384Final = SHA2Final
#pragma weak SHA512Final = SHA2Final

#endif  /* _KERNEL */

#ifdef _LITTLE_ENDIAN
#include <sys/byteorder.h>
#define HAVE_HTONL
#endif

static void Encode(uint8_t *, uint32_t *, size_t);
static void Encode64(uint8_t *, uint64_t *, size_t);

#if     defined(__amd64)
#define SHA512Transform(ctx, in) SHA512TransformBlocks((ctx), (in), 1)
#define SHA256Transform(ctx, in) SHA256TransformBlocks((ctx), (in), 1)

void SHA512TransformBlocks(SHA2_CTX *ctx, const void *in, size_t num);
void SHA256TransformBlocks(SHA2_CTX *ctx, const void *in, size_t num);

#else
static void SHA256Transform(SHA2_CTX *, const uint8_t *);
static void SHA512Transform(SHA2_CTX *, const uint8_t *);
#endif  /* __amd64 */

static uint8_t PADDING[128] = { 0x80, /* all zeros */ };

/* Ch and Maj are the basic SHA2 functions. */
#define Ch(b, c, d)     (((b) & (c)) ^ ((~b) & (d)))
#define Maj(b, c, d)    (((b) & (c)) ^ ((b) & (d)) ^ ((c) & (d)))

/* Rotates x right n bits. */
#define ROTR(x, n)      \
        (((x) >> (n)) | ((x) << ((sizeof (x) * NBBY)-(n))))

/* Shift x right n bits */
#define SHR(x, n)       ((x) >> (n))

/* SHA256 Functions */
#define BIGSIGMA0_256(x)        (ROTR((x), 2) ^ ROTR((x), 13) ^ ROTR((x), 22))
#define BIGSIGMA1_256(x)        (ROTR((x), 6) ^ ROTR((x), 11) ^ ROTR((x), 25))
#define SIGMA0_256(x)           (ROTR((x), 7) ^ ROTR((x), 18) ^ SHR((x), 3))
#define SIGMA1_256(x)           (ROTR((x), 17) ^ ROTR((x), 19) ^ SHR((x), 10))

#define SHA256ROUND(a, b, c, d, e, f, g, h, i, w)                       \
        T1 = h + BIGSIGMA1_256(e) + Ch(e, f, g) + SHA256_CONST(i) + w;  \
        d += T1;                                                        \
        T2 = BIGSIGMA0_256(a) + Maj(a, b, c);                           \
        h = T1 + T2

/* SHA384/512 Functions */
#define BIGSIGMA0(x)    (ROTR((x), 28) ^ ROTR((x), 34) ^ ROTR((x), 39))
#define BIGSIGMA1(x)    (ROTR((x), 14) ^ ROTR((x), 18) ^ ROTR((x), 41))
#define SIGMA0(x)       (ROTR((x), 1) ^ ROTR((x), 8) ^ SHR((x), 7))
#define SIGMA1(x)       (ROTR((x), 19) ^ ROTR((x), 61) ^ SHR((x), 6))
#define SHA512ROUND(a, b, c, d, e, f, g, h, i, w)                       \
        T1 = h + BIGSIGMA1(e) + Ch(e, f, g) + SHA512_CONST(i) + w;      \
        d += T1;                                                        \
        T2 = BIGSIGMA0(a) + Maj(a, b, c);                               \
        h = T1 + T2

/*
 * sparc optimization:
 *
 * on the sparc, we can load big endian 32-bit data easily.  note that
 * special care must be taken to ensure the address is 32-bit aligned.
```

```
 * in the interest of speed, we don't check to make sure, since
 * careful programming can guarantee this for us.
 */

#if     defined(_BIG_ENDIAN)

#define LOAD_BIG_32(addr)       (*(uint32_t *)(addr))
#define LOAD_BIG_64(addr)       (*(uint64_t *)(addr))

#elif   defined(HAVE_HTONL)
#define LOAD_BIG_32(addr) htonl(*((uint32_t *)(addr)))
#define LOAD_BIG_64(addr) htonll(*((uint64_t *)(addr)))
#else   /* little endian -- will work on big endian, but slowly */

#else
/* little endian -- will work on big endian, but slowly */
#define LOAD_BIG_32(addr)       \
        (((addr)[0] << 24) | ((addr)[1] << 16) | ((addr)[2] << 8) | (addr)[3])
#endif


#if     defined(_BIG_ENDIAN)

#define LOAD_BIG_64(addr)       (*(uint64_t *)(addr))

#else   /* little endian -- will work on big endian, but slowly */

#define LOAD_BIG_64(addr)       \
        (((uint64_t)(addr)[0] << 56) | ((uint64_t)(addr)[1] << 48) |    \
            ((uint64_t)(addr)[2] << 40) | ((uint64_t)(addr)[3] << 32) | \
            ((uint64_t)(addr)[4] << 24) | ((uint64_t)(addr)[5] << 16) | \
            ((uint64_t)(addr)[6] << 8) | (uint64_t)(addr)[7])
#endif  /* _BIG_ENDIAN */
#endif


#if     !defined(__amd64)
/* SHA256 Transform */

static void
SHA256Transform(SHA2_CTX *ctx, const uint8_t *blk)
{
        uint32_t a = ctx->state.s32[0];
        uint32_t b = ctx->state.s32[1];
        uint32_t c = ctx->state.s32[2];
        uint32_t d = ctx->state.s32[3];
        uint32_t e = ctx->state.s32[4];
        uint32_t f = ctx->state.s32[5];
        uint32_t g = ctx->state.s32[6];
        uint32_t h = ctx->state.s32[7];

        uint32_t w0, w1, w2, w3, w4, w5, w6, w7;
        uint32_t w8, w9, w10, w11, w12, w13, w14, w15;
        uint32_t T1, T2;

#if     defined(__sparc)
        static const uint32_t sha256_consts[] = {
                SHA256_CONST_0, SHA256_CONST_1, SHA256_CONST_2,
                SHA256_CONST_3, SHA256_CONST_4, SHA256_CONST_5,
                SHA256_CONST_6, SHA256_CONST_7, SHA256_CONST_8,
                SHA256_CONST_9, SHA256_CONST_10, SHA256_CONST_11,
                SHA256_CONST_12, SHA256_CONST_13, SHA256_CONST_14,
                SHA256_CONST_15, SHA256_CONST_16, SHA256_CONST_17,
                SHA256_CONST_18, SHA256_CONST_19, SHA256_CONST_20,
                SHA256_CONST_21, SHA256_CONST_22, SHA256_CONST_23,
                SHA256_CONST_24, SHA256_CONST_25, SHA256_CONST_26,
```



```
                SHA256_CONST_27, SHA256_CONST_28, SHA256_CONST_29,
                SHA256_CONST_30, SHA256_CONST_31, SHA256_CONST_32,
                SHA256_CONST_33, SHA256_CONST_34, SHA256_CONST_35,
                SHA256_CONST_36, SHA256_CONST_37, SHA256_CONST_38,
                SHA256_CONST_39, SHA256_CONST_40, SHA256_CONST_41,
                SHA256_CONST_42, SHA256_CONST_43, SHA256_CONST_44,
                SHA256_CONST_45, SHA256_CONST_46, SHA256_CONST_47,
                SHA256_CONST_48, SHA256_CONST_49, SHA256_CONST_50,
                SHA256_CONST_51, SHA256_CONST_52, SHA256_CONST_53,
                SHA256_CONST_54, SHA256_CONST_55, SHA256_CONST_56,
                SHA256_CONST_57, SHA256_CONST_58, SHA256_CONST_59,
                SHA256_CONST_60, SHA256_CONST_61, SHA256_CONST_62,
                SHA256_CONST_63
        };
#endif  /* __sparc */

        if ((uintptr_t)blk & 0x3) {             /* not 4-byte aligned? */
                bcopy(blk, ctx->buf_un.buf32,  sizeof (ctx->buf_un.buf32));
                blk = (uint8_t *)ctx->buf_un.buf32;
        }

        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w0 =  LOAD_BIG_32(blk + 4 * 0);
        SHA256ROUND(a, b, c, d, e, f, g, h, 0, w0);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w1 =  LOAD_BIG_32(blk + 4 * 1);
        SHA256ROUND(h, a, b, c, d, e, f, g, 1, w1);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w2 =  LOAD_BIG_32(blk + 4 * 2);
        SHA256ROUND(g, h, a, b, c, d, e, f, 2, w2);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w3 =  LOAD_BIG_32(blk + 4 * 3);
        SHA256ROUND(f, g, h, a, b, c, d, e, 3, w3);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w4 =  LOAD_BIG_32(blk + 4 * 4);
        SHA256ROUND(e, f, g, h, a, b, c, d, 4, w4);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w5 =  LOAD_BIG_32(blk + 4 * 5);
        SHA256ROUND(d, e, f, g, h, a, b, c, 5, w5);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w6 =  LOAD_BIG_32(blk + 4 * 6);
        SHA256ROUND(c, d, e, f, g, h, a, b, 6, w6);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w7 =  LOAD_BIG_32(blk + 4 * 7);
        SHA256ROUND(b, c, d, e, f, g, h, a, 7, w7);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w8 =  LOAD_BIG_32(blk + 4 * 8);
        SHA256ROUND(a, b, c, d, e, f, g, h, 8, w8);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w9 =  LOAD_BIG_32(blk + 4 * 9);
        SHA256ROUND(h, a, b, c, d, e, f, g, 9, w9);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w10 =  LOAD_BIG_32(blk + 4 * 10);
        SHA256ROUND(g, h, a, b, c, d, e, f, 10, w10);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w11 =  LOAD_BIG_32(blk + 4 * 11);
        SHA256ROUND(f, g, h, a, b, c, d, e, 11, w11);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w12 =  LOAD_BIG_32(blk + 4 * 12);
        SHA256ROUND(e, f, g, h, a, b, c, d, 12, w12);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w13 =  LOAD_BIG_32(blk + 4 * 13);
        SHA256ROUND(d, e, f, g, h, a, b, c, 13, w13);
        /* LINTED E_BAD_PTR_CAST_ALIGN */
        w14 =  LOAD_BIG_32(blk + 4 * 14);
        SHA256ROUND(c, d, e, f, g, h, a, b, 14, w14);
```

```
243         /* LINTED E_BAD_PTR_CAST_ALIGN */
244         w15 =  LOAD_BIG_32(blk + 4 * 15);
245         SHA256ROUND(b, c, d, e, f, g, h, a, 15, w15);

247         w0 = SIGMA1_256(w14) + w9 + SIGMA0_256(w1) + w0;
248         SHA256ROUND(a, b, c, d, e, f, g, h, 16, w0);
249         w1 = SIGMA1_256(w15) + w10 + SIGMA0_256(w2) + w1;
250         SHA256ROUND(h, a, b, c, d, e, f, g, 17, w1);
251         w2 = SIGMA1_256(w0) + w11 + SIGMA0_256(w3) + w2;
252         SHA256ROUND(g, h, a, b, c, d, e, f, 18, w2);
253         w3 = SIGMA1_256(w1) + w12 + SIGMA0_256(w4) + w3;
254         SHA256ROUND(f, g, h, a, b, c, d, e, 19, w3);
255         w4 = SIGMA1_256(w2) + w13 + SIGMA0_256(w5) + w4;
256         SHA256ROUND(e, f, g, h, a, b, c, d, 20, w4);
257         w5 = SIGMA1_256(w3) + w14 + SIGMA0_256(w6) + w5;
258         SHA256ROUND(d, e, f, g, h, a, b, c, 21, w5);
259         w6 = SIGMA1_256(w4) + w15 + SIGMA0_256(w7) + w6;
260         SHA256ROUND(c, d, e, f, g, h, a, b, 22, w6);
261         w7 = SIGMA1_256(w5) + w0 + SIGMA0_256(w8) + w7;
262         SHA256ROUND(b, c, d, e, f, g, h, a, 23, w7);
263         w8 = SIGMA1_256(w6) + w1 + SIGMA0_256(w9) + w8;
264         SHA256ROUND(a, b, c, d, e, f, g, h, 24, w8);
265         w9 = SIGMA1_256(w7) + w2 + SIGMA0_256(w10) + w9;
266         SHA256ROUND(h, a, b, c, d, e, f, g, 25, w9);
267         w10 = SIGMA1_256(w8) + w3 + SIGMA0_256(w11) + w10;
268         SHA256ROUND(g, h, a, b, c, d, e, f, 26, w10);
269         w11 = SIGMA1_256(w9) + w4 + SIGMA0_256(w12) + w11;
270         SHA256ROUND(f, g, h, a, b, c, d, e, 27, w11);
271         w12 = SIGMA1_256(w10) + w5 + SIGMA0_256(w13) + w12;
272         SHA256ROUND(e, f, g, h, a, b, c, d, 28, w12);
273         w13 = SIGMA1_256(w11) + w6 + SIGMA0_256(w14) + w13;
274         SHA256ROUND(d, e, f, g, h, a, b, c, 29, w13);
275         w14 = SIGMA1_256(w12) + w7 + SIGMA0_256(w15) + w14;
276         SHA256ROUND(c, d, e, f, g, h, a, b, 30, w14);
277         w15 = SIGMA1_256(w13) + w8 + SIGMA0_256(w0) + w15;
278         SHA256ROUND(b, c, d, e, f, g, h, a, 31, w15);

280         w0 = SIGMA1_256(w14) + w9 + SIGMA0_256(w1) + w0;
281         SHA256ROUND(a, b, c, d, e, f, g, h, 32, w0);
282         w1 = SIGMA1_256(w15) + w10 + SIGMA0_256(w2) + w1;
283         SHA256ROUND(h, a, b, c, d, e, f, g, 33, w1);
284         w2 = SIGMA1_256(w0) + w11 + SIGMA0_256(w3) + w2;
285         SHA256ROUND(g, h, a, b, c, d, e, f, 34, w2);
286         w3 = SIGMA1_256(w1) + w12 + SIGMA0_256(w4) + w3;
287         SHA256ROUND(f, g, h, a, b, c, d, e, 35, w3);
288         w4 = SIGMA1_256(w2) + w13 + SIGMA0_256(w5) + w4;
289         SHA256ROUND(e, f, g, h, a, b, c, d, 36, w4);
290         w5 = SIGMA1_256(w3) + w14 + SIGMA0_256(w6) + w5;
291         SHA256ROUND(d, e, f, g, h, a, b, c, 37, w5);
292         w6 = SIGMA1_256(w4) + w15 + SIGMA0_256(w7) + w6;
293         SHA256ROUND(c, d, e, f, g, h, a, b, 38, w6);
294         w7 = SIGMA1_256(w5) + w0 + SIGMA0_256(w8) + w7;
295         SHA256ROUND(b, c, d, e, f, g, h, a, 39, w7);
296         w8 = SIGMA1_256(w6) + w1 + SIGMA0_256(w9) + w8;
297         SHA256ROUND(a, b, c, d, e, f, g, h, 40, w8);
298         w9 = SIGMA1_256(w7) + w2 + SIGMA0_256(w10) + w9;
299         SHA256ROUND(h, a, b, c, d, e, f, g, 41, w9);
300         w10 = SIGMA1_256(w8) + w3 + SIGMA0_256(w11) + w10;
301         SHA256ROUND(g, h, a, b, c, d, e, f, 42, w10);
302         w11 = SIGMA1_256(w9) + w4 + SIGMA0_256(w12) + w11;
303         SHA256ROUND(f, g, h, a, b, c, d, e, 43, w11);
304         w12 = SIGMA1_256(w10) + w5 + SIGMA0_256(w13) + w12;
305         SHA256ROUND(e, f, g, h, a, b, c, d, 44, w12);
306         w13 = SIGMA1_256(w11) + w6 + SIGMA0_256(w14) + w13;
307         SHA256ROUND(d, e, f, g, h, a, b, c, 45, w13);
308         w14 = SIGMA1_256(w12) + w7 + SIGMA0_256(w15) + w14;
```



```
309         SHA256ROUND(c, d, e, f, g, h, a, b, 46, w14);
310         w15 = SIGMA1_256(w13) + w8 + SIGMA0_256(w0) + w15;
311         SHA256ROUND(b, c, d, e, f, g, h, a, 47, w15);

313         w0 = SIGMA1_256(w14) + w9 + SIGMA0_256(w1) + w0;
314         SHA256ROUND(a, b, c, d, e, f, g, h, 48, w0);
315         w1 = SIGMA1_256(w15) + w10 + SIGMA0_256(w2) + w1;
316         SHA256ROUND(h, a, b, c, d, e, f, g, 49, w1);
317         w2 = SIGMA1_256(w0) + w11 + SIGMA0_256(w3) + w2;
318         SHA256ROUND(g, h, a, b, c, d, e, f, 50, w2);
319         w3 = SIGMA1_256(w1) + w12 + SIGMA0_256(w4) + w3;
320         SHA256ROUND(f, g, h, a, b, c, d, e, 51, w3);
321         w4 = SIGMA1_256(w2) + w13 + SIGMA0_256(w5) + w4;
322         SHA256ROUND(e, f, g, h, a, b, c, d, 52, w4);
323         w5 = SIGMA1_256(w3) + w14 + SIGMA0_256(w6) + w5;
324         SHA256ROUND(d, e, f, g, h, a, b, c, 53, w5);
325         w6 = SIGMA1_256(w4) + w15 + SIGMA0_256(w7) + w6;
326         SHA256ROUND(c, d, e, f, g, h, a, b, 54, w6);
327         w7 = SIGMA1_256(w5) + w0 + SIGMA0_256(w8) + w7;
328         SHA256ROUND(b, c, d, e, f, g, h, a, 55, w7);
329         w8 = SIGMA1_256(w6) + w1 + SIGMA0_256(w9) + w8;
330         SHA256ROUND(a, b, c, d, e, f, g, h, 56, w8);
331         w9 = SIGMA1_256(w7) + w2 + SIGMA0_256(w10) + w9;
332         SHA256ROUND(h, a, b, c, d, e, f, g, 57, w9);
333         w10 = SIGMA1_256(w8) + w3 + SIGMA0_256(w11) + w10;
334         SHA256ROUND(g, h, a, b, c, d, e, f, 58, w10);
335         w11 = SIGMA1_256(w9) + w4 + SIGMA0_256(w12) + w11;
336         SHA256ROUND(f, g, h, a, b, c, d, e, 59, w11);
337         w12 = SIGMA1_256(w10) + w5 + SIGMA0_256(w13) + w12;
338         SHA256ROUND(e, f, g, h, a, b, c, d, 60, w12);
339         w13 = SIGMA1_256(w11) + w6 + SIGMA0_256(w14) + w13;
340         SHA256ROUND(d, e, f, g, h, a, b, c, 61, w13);
341         w14 = SIGMA1_256(w12) + w7 + SIGMA0_256(w15) + w14;
342         SHA256ROUND(c, d, e, f, g, h, a, b, 62, w14);
343         w15 = SIGMA1_256(w13) + w8 + SIGMA0_256(w0) + w15;
344         SHA256ROUND(b, c, d, e, f, g, h, a, 63, w15);

346         ctx->state.s32[0] += a;
347         ctx->state.s32[1] += b;
348         ctx->state.s32[2] += c;
349         ctx->state.s32[3] += d;
350         ctx->state.s32[4] += e;
351         ctx->state.s32[5] += f;
352         ctx->state.s32[6] += g;
353         ctx->state.s32[7] += h;
354 }
```

_____***unchanged_portion_omitted_***

```
674 void
675 SHA2Init(uint64_t mech, SHA2_CTX *ctx)
676 {

678         switch (mech) {
679         case SHA256_MECH_INFO_TYPE:
680         case SHA256_HMAC_MECH_INFO_TYPE:
681         case SHA256_HMAC_GEN_MECH_INFO_TYPE:
682                 ctx->state.s32[0] = 0x6a09e667U;
683                 ctx->state.s32[1] = 0xbb67ae85U;
684                 ctx->state.s32[2] = 0x3c6ef372U;
685                 ctx->state.s32[3] = 0xa54ff53aU;
686                 ctx->state.s32[4] = 0x510e527fU;
687                 ctx->state.s32[5] = 0x9b05688cU;
688                 ctx->state.s32[6] = 0x1f83d9abU;
689                 ctx->state.s32[7] = 0x5be0cd19U;
690                 break;
```

```
691         case SHA384_MECH_INFO_TYPE:
692         case SHA384_HMAC_MECH_INFO_TYPE:
693         case SHA384_HMAC_GEN_MECH_INFO_TYPE:
694                 ctx->state.s64[0] = 0xcbbb9d5dc1059ed8ULL;
695                 ctx->state.s64[1] = 0x629a292a367cd507ULL;
696                 ctx->state.s64[2] = 0x9159015a3070dd17ULL;
697                 ctx->state.s64[3] = 0x152fecd8f70e5939ULL;
698                 ctx->state.s64[4] = 0x67332667ffc00b31ULL;
699                 ctx->state.s64[5] = 0x8eb44a8768581511ULL;
700                 ctx->state.s64[6] = 0xdb0c2e0d64f98fa7ULL;
701                 ctx->state.s64[7] = 0x47b5481dbefa4fa4ULL;
702                 break;
703         case SHA512_MECH_INFO_TYPE:
704         case SHA512_HMAC_MECH_INFO_TYPE:
705         case SHA512_HMAC_GEN_MECH_INFO_TYPE:
706                 ctx->state.s64[0] = 0x6a09e667f3bcc908ULL;
707                 ctx->state.s64[1] = 0xbb67ae8584caa73bULL;
708                 ctx->state.s64[2] = 0x3c6ef372fe94f82bULL;
709                 ctx->state.s64[3] = 0xa54ff53a5f1d36f1ULL;
710                 ctx->state.s64[4] = 0x510e527fade682d1ULL;
711                 ctx->state.s64[5] = 0x9b05688c2b3e6c1fULL;
712                 ctx->state.s64[6] = 0x1f83d9abfb41bd6bULL;
713                 ctx->state.s64[7] = 0x5be0cd19137e2179ULL;
714                 break;
715 #ifdef _KERNEL
716         default:
717                 cmn_err(CE_PANIC,
718                     "sha2_init: failed to find a supported algorithm: 0x%x",
718                 cmn_err(CE_PANIC, "sha2_init: "
719                     "failed to find a supported algorithm: 0x%x",
719                     (uint32_t)mech);

721 #endif /* _KERNEL */
722         }

724         ctx->algotype = mech;
725         ctx->count.c64[0] = ctx->count.c64[1] = 0;
726 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    2736 Wed Aug 27 14:23:06 2008
new/usr/src/lib/libc/amd64/gen/byteorder.s
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

        .file   "byteorder.s"

#include <sys/asm_linkage.h>
#include "SYS.h"

        /*
         * NOTE: htonll/ntohll, htonl/ntohl, and htons/ntohs are identical
         * routines. As such, they could be implemented as a single routine,
         * using multiple ALTENTRY/SET_SIZE definitions. We don't do this so
         * NOTE: htonl/ntohl are identical routines, as are htons/ntohs.
         * As such, they could be implemented as a single routine, using
         * multiple ALTENTRY/SET_SIZE definitions. We don't do this so
         * that they will have unique addresses, allowing DTrace and
         * other debuggers to tell them apart.
         */

/*
 *      unsigned long long htonll( hll )
 *      unsigned long long ntohll( hll )
 *      unsigned long long hll;
 *      reverses the byte order of 'uint64_t hll' on little endian machines
 */
        ENTRY(htonll)
        movq    %rdi, %rax      /* %rax = hll */
        bswapq  %rax            /* reverses the byte order of %rax */
        ret                     /* return (%rax) */
        SET_SIZE(htonll)

        ENTRY(ntohll)
        movq    %rdi, %rax      /* %rax = hll */
        bswapq  %rax            /* reverses the byte order of %rax */
        ret                     /* return (%rax) */
        SET_SIZE(ntohll)
```



```
/*
 *      unsigned long htonl( hl )
 *      unsigned long ntohl( hl )
 *      unsigned long hl;
 *      reverses the byte order of 'uint32_t hl' on little endian machines
 *      long hl;
 *      reverses the byte order of 'uint32_t hl'
 */
        ENTRY(htonl)
        movl    %edi, %eax      /* %eax = hl */
        bswap   %eax            /* reverses the byte order of %eax */
        ret                     /* return (%eax) */
        SET_SIZE(htonl)
______unchanged_portion_omitted_

/*
 *      unsigned short htons( hs )
 *      unsigned short hs;
 *      reverses the byte order of 'uint16_t hs' on little endian machines.
 *      short hs;
 *
 *      reverses the byte order in hs.
 */
        ENTRY(htons)
        movl    %edi, %eax      /* %eax = hs */
        bswap   %eax            /* reverses the byte order of %eax */
        shrl    $16, %eax       /* moves high 16-bit to low 16-bit */
        ret                     /* return (%eax) */
        SET_SIZE(htons)
______unchanged_portion_omitted_
```

```
**********************************************************
   22212 Wed Aug 27 14:23:11 2008
new/usr/src/lib/libc/i386/Makefile.com
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
#
# 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 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

LIB_PIC= libc_pic.a
VERS=   .1
CPP=    /usr/lib/cpp
TARGET_ARCH=    i386

VALUES= values-Xa.o

# objects are grouped by source directory

# local objects
STRETS=

CRTOBJS=                        \
        cerror.o                \
        cerror64.o

DYNOBJS=                        \
        _rtbootld.o

FPOBJS=                         \
        _D_cplx_div.o           \
        _D_cplx_div_ix.o        \
        _D_cplx_div_rx.o        \
        _D_cplx_lr_div.o        \
        _D_cplx_lr_div_ix.o     \
        _D_cplx_lr_div_rx.o     \
        _D_cplx_mul.o           \
        _F_cplx_div.o           \
        _F_cplx_div_ix.o        \
        _F_cplx_div_rx.o        \
        _F_cplx_lr_div.o        \
        _F_cplx_lr_div_ix.o     \
```



```
        _F_cplx_lr_div_rx.o     \
        _F_cplx_mul.o           \
        _X_cplx_div.o           \
        _X_cplx_div_ix.o        \
        _X_cplx_div_rx.o        \
        _X_cplx_lr_div.o        \
        _X_cplx_lr_div_ix.o     \
        _X_cplx_lr_div_rx.o     \
        _X_cplx_mul.o           \
        _base_il.o              \
        fpgetmask.o             \
        fpgetround.o            \
        fpgetsticky.o           \
        fpsetmask.o             \
        fpsetround.o            \
        fpsetsticky.o           \
        fpstart.o

FPASMOBJS=                      \
        __xgetRD.o              \
        _xtoll.o                \
        _xtoull.o               \
        fpcw.o

ATOMICOBJS=                     \
        atomic.o

XATTROBJS=                      \
        xattr_common.o

COMOBJS=                        \
        bcmp.o                  \
        bcopy.o                 \
        bsearch.o               \
        bzero.o                 \
        ffs.o                   \
        qsort.o                 \
        strtol.o                \
        strtoul.o

DTRACEOBJS=                     \
        dtrace_data.o

GENOBJS=                        \
        _div64.o                \
        _divdi3.o               \
        _getsp.o                \
        _mul64.o                \
        abs.o                   \
        alloca.o                \
        byteorder.o             \
        byteorder64.o           \
        cuexit.o                \
        ecvt.o                  \
        errlst.o                \
        i386_data.o             \
        ladd.o                  \
        ldivide.o               \
        lmul.o                  \
        lock.o                  \
        lshiftl.o               \
        lsign.o                 \
        lsub.o                  \
        makectxt.o              \
        memccpy.o               \
        memchr.o                \
```

```
124         memcmp.o                \
125         memcpy.o                \
126         memset.o                \
127         new_list.o              \
128         setjmp.o                \
129         siginfolst.o            \
130         siglongjmp.o            \
131         strcat.o                \
132         strchr.o                \
133         strcmp.o                \
134         strcpy.o                \
135         strlen.o                \
136         strncat.o               \
137         strncmp.o               \
138         strncpy.o               \
139         strnlen.o               \
140         strrchr.o               \
141         sync_instruction_memory.o

143 # sysobjs that contain large-file interfaces
144 COMSYSOBJS64=                   \
145         creat64.o               \
146         fstat64.o               \
147         fstatvfs64.o            \
148         getdents64.o            \
149         getrlimit64.o           \
150         lseek64.o               \
151         lstat64.o               \
152         open64.o                \
153         pread64.o               \
154         pwrite64.o              \
155         setrlimit64.o           \
156         stat64.o                \
157         statvfs64.o

159 SYSOBJS64=                      \
160         mmap64.o

162 COMSYSOBJS=                     \
163         __clock_timer.o         \
164         __getloadavg.o          \
165         __rusagesys.o           \
166         __signotify.o           \
167         __sigrt.o               \
168         __time.o                \
169         _lgrp_home_fast.o       \
170         _lgrpsys.o              \
171         _nfssys.o               \
172         _portfs.o               \
173         _pset.o                 \
174         _rpcsys.o               \
175         _sigaction.o            \
176         _so_accept.o            \
177         _so_bind.o              \
178         _so_connect.o           \
179         _so_getpeername.o       \
180         _so_getsockname.o       \
181         _so_getsockopt.o        \
182         _so_listen.o            \
183         _so_recv.o              \
184         _so_recvfrom.o          \
185         _so_recvmsg.o           \
186         _so_send.o              \
187         _so_sendmsg.o           \
188         _so_sendto.o            \
189         _so_setsockopt.o        \
```



```
190         _so_shutdown.o          \
191         _so_socket.o            \
192         _so_socketpair.o        \
193         _sockconfig.o           \
194         access.o                \
195         acct.o                  \
196         acl.o                   \
197         adjtime.o               \
198         alarm.o                 \
199         brk.o                   \
200         chdir.o                 \
201         chmod.o                 \
202         chown.o                 \
203         chroot.o                \
204         cladm.o                 \
205         close.o                 \
206         creat.o                 \
207         dup.o                   \
208         execve.o                \
209         exit.o                  \
210         facl.o                  \
211         fchdir.o                \
212         fchmod.o                \
213         fchown.o                \
214         fchroot.o               \
215         fcntl.o                 \
216         fdsync.o                \
217         fpathconf.o             \
218         fstat.o                 \
219         fstatfs.o               \
220         fstatvfs.o              \
221         getcpuid.o              \
222         getdents.o              \
223         getegid.o               \
224         geteuid.o               \
225         getgid.o                \
226         getgroups.o             \
227         gethrtime.o             \
228         getitimer.o             \
229         getmsg.o                \
230         getpid.o                \
231         getpmsg.o               \
232         getppid.o               \
233         getrlimit.o             \
234         getuid.o                \
235         gtty.o                  \
236         install_utrap.o         \
237         ioctl.o                 \
238         kaio.o                  \
239         kill.o                  \
240         lchown.o                \
241         link.o                  \
242         llseek.o                \
243         lseek.o                 \
244         lstat.o                 \
245         memcntl.o               \
246         mincore.o               \
247         mkdir.o                 \
248         mknod.o                 \
249         mmap.o                  \
250         modctl.o                \
251         mount.o                 \
252         mprotect.o              \
253         munmap.o                \
254         nice.o                  \
255         ntp_adjtime.o           \
```

```
256         ntp_gettime.o           \
257         open.o                  \
258         p_online.o              \
259         pathconf.o              \
260         pause.o                 \
261         pcsample.o              \
262         pollsys.o               \
263         pread.o                 \
264         priocntlset.o           \
265         processor_bind.o        \
266         processor_info.o        \
267         profil.o                \
268         putmsg.o                \
269         putpmsg.o               \
270         pwrite.o                \
271         read.o                  \
272         readlink.o              \
273         readv.o                 \
274         rename.o                \
275         resolvepath.o           \
276         rmdir.o                 \
277         seteguid.o              \
278         setgid.o                \
279         setgroups.o             \
280         setitimer.o             \
281         setreid.o               \
282         setrlimit.o             \
283         setuid.o                \
284         sigaltstk.o             \
285         sigprocmsk.o            \
286         sigsendset.o            \
287         sigsuspend.o            \
288         stat.o                  \
289         statfs.o                \
290         statvfs.o               \
291         stty.o                  \
292         symlink.o               \
293         sync.o                  \
294         sysconfig.o             \
295         sysfs.o                 \
296         sysinfo.o               \
297         syslwp.o                \
298         times.o                 \
299         ulimit.o                \
300         umask.o                 \
301         umount2.o               \
302         unlink.o                \
303         utime.o                 \
304         utimes.o                \
305         utssys.o                \
306         uucopy.o                \
307         vhangup.o               \
308         waitid.o                \
309         write.o                 \
310         writev.o                \
311         yield.o

313 SYSOBJS=                       \
314         __clock_gettime.o       \
315         __getcontext.o          \
316         __uadmin.o              \
317         _lwp_mutex_unlock.o     \
318         _stack_grow.o           \
319         door.o                  \
320         forkx.o                 \
321         forkallx.o              \
```



```
322         fxstat.o                \
323         getcontext.o            \
324         gettimeofday.o          \
325         lwp_private.o           \
326         lxstat.o                \
327         nuname.o                \
328         pipe.o                  \
329         ptrace.o                \
330         syscall.o               \
331         sysi86.o                \
332         tls_get_addr.o          \
333         uadmin.o                \
334         umount.o                \
335         uname.o                 \
336         vforkx.o                \
337         xmknod.o                \
338         xstat.o

340 # objects under ../port which contain transitional large file interfaces
341 PORTGEN64=                      \
342         _xftw64.o               \
343         attropen64.o            \
344         ftw64.o                 \
345         mkstemp64.o             \
346         nftw64.o                \
347         tell64.o                \
348         truncate64.o

350 # objects from source under ../port
351 PORTFP=                         \
352         __flt_decim.o           \
353         __flt_rounds.o          \
354         __tbl_10_b.o            \
355         __tbl_10_h.o            \
356         __tbl_10_s.o            \
357         __tbl_2_b.o             \
358         __tbl_2_h.o             \
359         __tbl_2_s.o             \
360         __tbl_fdq.o             \
361         __tbl_tens.o            \
362         __x_power.o             \
363         _base_sup.o             \
364         aconvert.o              \
365         decimal_bin.o           \
366         double_decim.o          \
367         econvert.o              \
368         fconvert.o              \
369         file_decim.o            \
370         finite.o                \
371         fp_data.o               \
372         func_decim.o            \
373         gconvert.o              \
374         hex_bin.o               \
375         ieee_globals.o          \
376         pack_float.o            \
377         sigfpe.o                \
378         string_decim.o

380 PORTGEN=                       \
381         _env_data.o             \
382         _xftw.o                 \
383         a64l.o                  \
384         abort.o                 \
385         addsev.o                \
386         assert.o                \
387         atof.o                  \
```

```
388         atoi.o                  \
389         atol.o                  \
390         atoll.o                 \
391         attrat.o                \
392         attropen.o              \
393         atexit.o                \
394         atfork.o                \
395         basename.o              \
396         calloc.o                \
397         catgets.o               \
398         catopen.o               \
399         cfgetispeed.o           \
400         cfgetospeed.o           \
401         cfree.o                 \
402         cfsetispeed.o           \
403         cfsetospeed.o           \
404         cftime.o                \
405         clock.o                 \
406         closedir.o              \
407         closefrom.o             \
408         confstr.o               \
409         crypt.o                 \
410         csetlen.o               \
411         ctime.o                 \
412         ctime_r.o               \
413         deflt.o                 \
414         directio.o              \
415         dirname.o               \
416         div.o                   \
417         drand48.o               \
418         dup2.o                  \
419         env_data.o              \
420         err.o                   \
421         errno.o                 \
422         euclen.o                \
423         event_port.o            \
424         execvp.o                \
425         fattach.o               \
426         fdetach.o               \
427         fdopendir.o             \
428         fmtmsg.o                \
429         ftime.o                 \
430         ftok.o                  \
431         ftw.o                   \
432         gcvt.o                  \
433         getauxv.o               \
434         getcwd.o                \
435         getdate_err.o           \
436         getdtblsize.o           \
437         getenv.o                \
438         getexecname.o           \
439         getgrnam.o              \
440         getgrnam_r.o            \
441         gethostid.o             \
442         gethostname.o           \
443         gethz.o                 \
444         getisax.o               \
445         getloadavg.o            \
446         getlogin.o              \
447         getmntent.o             \
448         getnetgrent.o           \
449         getopt.o                \
450         getopt_long.o           \
451         getpagesize.o           \
452         getpw.o                 \
453         getpwnam.o              \
```



```
454         getpwnam_r.o            \
455         getrusage.o             \
456         getspent.o              \
457         getspent_r.o            \
458         getsubopt.o             \
459         gettxt.o                \
460         getusershell.o          \
461         getut.o                 \
462         getutx.o                \
463         getvfsent.o             \
464         getwd.o                 \
465         getwidth.o              \
466         getxby_door.o           \
467         gtxt.o                  \
468         hsearch.o               \
469         iconv.o                 \
470         imaxabs.o               \
471         imaxdiv.o               \
472         index.o                 \
473         initgroups.o            \
474         insque.o                \
475         isaexec.o               \
476         isastream.o             \
477         isatty.o                \
478         killpg.o                \
479         klpdlib.o               \
480         l64a.o                  \
481         lckpwdf.o               \
482         lconstants.o            \
483         lexp10.o                \
484         lfind.o                 \
485         lfmt.o                  \
486         lfmt_log.o              \
487         llabs.o                 \
488         lldiv.o                 \
489         llog10.o                \
490         lltostr.o               \
491         localtime.o             \
492         lsearch.o               \
493         madvise.o               \
494         malloc.o                \
495         memalign.o              \
496         mkdev.o                 \
497         mkdtemp.o               \
498         mkfifo.o                \
499         mkstemp.o               \
500         mktemp.o                \
501         mlock.o                 \
502         mlockall.o              \
503         mon.o                   \
504         msync.o                 \
505         munlock.o               \
506         munlockall.o            \
507         ndbm.o                  \
508         nftw.o                  \
509         nlspath_checks.o        \
510         nsparse.o               \
511         nss_common.o            \
512         nss_dbdefs.o            \
513         nss_deffinder.o         \
514         opendir.o               \
515         opt_data.o              \
516         perror.o                \
517         pfmt.o                  \
518         pfmt_data.o             \
519         pfmt_print.o            \
```

```
520         plock.o                 \
521         poll.o                  \
522         posix_fadvise.o         \
523         posix_fallocate.o       \
524         posix_madvise.o         \
525         posix_memalign.o        \
526         priocntl.o              \
527         privlib.o               \
528         priv_str_xlate.o        \
529         psiginfo.o              \
530         psignal.o               \
531         pt.o                    \
532         putpwent.o              \
533         putspent.o              \
534         raise.o                 \
535         rand.o                  \
536         random.o                \
537         rctlops.o               \
538         readdir.o               \
539         readdir_r.o             \
540         realpath.o              \
541         reboot.o                \
542         regexpr.o               \
543         remove.o                \
544         rewinddir.o             \
545         rindex.o                \
546         scandir.o               \
547         seekdir.o               \
548         select.o                \
549         select_large_fdset.o    \
550         setlabel.o              \
551         setpriority.o           \
552         settimeofday.o          \
553         sh_locks.o              \
554         sigflag.o               \
555         siglist.o               \
556         sigsend.o               \
557         sigsetops.o             \
558         ssignal.o               \
559         stack.o                 \
560         str2sig.o               \
561         strcase_charmap.o       \
562         strcasecmp.o            \
563         strcspn.o               \
564         strdup.o                \
565         strerror.o              \
566         strlcat.o               \
567         strlcpy.o               \
568         strncasecmp.o           \
569         strpbrk.o               \
570         strsignal.o             \
571         strspn.o                \
572         strstr.o                \
573         strtod.o                \
574         strtoimax.o             \
575         strtok.o                \
576         strtok_r.o              \
577         strtoll.o               \
578         strtoull.o              \
579         strtoumax.o             \
580         swab.o                  \
581         swapctl.o               \
582         sysconf.o               \
583         syslog.o                \
584         tcdrain.o               \
585         tcflow.o                \
```



```
586         tcflush.o               \
587         tcgetattr.o             \
588         tcgetpgrp.o             \
589         tcgetsid.o              \
590         tcsendbreak.o           \
591         tcsetattr.o             \
592         tcsetpgrp.o             \
593         tell.o                  \
594         telldir.o               \
595         tfind.o                 \
596         time_data.o             \
597         time_gdata.o            \
598         truncate.o              \
599         tsdalloc.o              \
600         tsearch.o               \
601         ttyname.o               \
602         ttyslot.o               \
603         ualarm.o                \
604         ucred.o                 \
605         valloc.o                \
606         vlfmt.o                 \
607         vpfmt.o                 \
608         waitpid.o               \
609         walkstack.o             \
610         wdata.o                 \
611         xgetwidth.o             \
612         xpg4.o                  \
613         xpg6.o

615 PORTPRINT_W=                     \
616         doprnt_w.o

618 PORTPRINT=                      \
619         doprnt.o                \
620         fprintf.o               \
621         printf.o                \
622         snprintf.o              \
623         sprintf.o               \
624         vfprintf.o              \
625         vprintf.o               \
626         vsnprintf.o             \
627         vsprintf.o              \
628         vwprintf.o              \
629         wprintf.o

631 # c89 variants to support 32-bit size of c89 u/intmax_t (32-bit libc only)
632 PORTPRINT_C89=                  \
633         vfprintf_c89.o          \
634         vprintf_c89.o           \
635         vsnprintf_c89.o         \
636         vsprintf_c89.o          \
637         vwprintf_c89.o

639 PORTSTDIO_C89=                  \
640         vscanf_c89.o            \
641         vwscanf_c89.o           \

643 # portable stdio objects that contain large file interfaces.
644 # Note: fopen64 is a special case, as we build it small.
645 PORTSTDIO64=                    \
646         fopen64.o               \
647         fpos64.o

649 PORTSTDIO_W=                    \
650         doscan_w.o
```

```
652 PORTSTDIO=                      \
653         __extensions.o          \
654         _endopen.o              \
655         _filbuf.o               \
656         _findbuf.o              \
657         _flsbuf.o               \
658         _wrtchk.o               \
659         clearerr.o              \
660         ctermid.o               \
661         ctermid_r.o             \
662         cuserid.o               \
663         data.o                  \
664         doscan.o                \
665         fdopen.o                \
666         feof.o                  \
667         ferror.o                \
668         fgetc.o                 \
669         fgets.o                 \
670         fileno.o                \
671         flockf.o                \
672         flush.o                 \
673         fopen.o                 \
674         fpos.o                  \
675         fputc.o                 \
676         fputs.o                 \
677         fread.o                 \
678         fseek.o                 \
679         fseeko.o                \
680         ftell.o                 \
681         ftello.o                \
682         fwrite.o                \
683         getc.o                  \
684         getchar.o               \
685         getpass.o               \
686         gets.o                  \
687         getw.o                  \
688         mse.o                   \
689         popen.o                 \
690         putc.o                  \
691         putchar.o               \
692         puts.o                  \
693         putw.o                  \
694         rewind.o                \
695         scanf.o                 \
696         setbuf.o                \
697         setbuffer.o             \
698         setvbuf.o               \
699         system.o                \
700         tempnam.o               \
701         tmpfile.o               \
702         tmpnam_r.o              \
703         ungetc.o                \
704         vscanf.o                \
705         vwscanf.o               \
706         wscanf.o

708 PORTI18N=                       \
709         __fgetwc_xpg5.o         \
710         __fgetws_xpg5.o         \
711         __fputwc_xpg5.o         \
712         __fputws_xpg5.o         \
713         __ungetwc_xpg5.o        \
714         getwchar.o              \
715         putwchar.o              \
716         putws.o                 \
717         strtows.o               \
```



```
718         wcsstr.o                \
719         wcstoimax.o             \
720         wcstol.o                \
721         wcstoul.o               \
722         wcswcs.o                \
723         wmemchr.o               \
724         wmemcmp.o               \
725         wmemcpy.o               \
726         wmemmove.o              \
727         wmemset.o               \
728         wscasecmp.o             \
729         wscat.o                 \
730         wschr.o                 \
731         wscmp.o                 \
732         wscpy.o                 \
733         wscspn.o                \
734         wsdup.o                 \
735         wslen.o                 \
736         wsncasecmp.o            \
737         wsncat.o                \
738         wsncmp.o                \
739         wsncpy.o                \
740         wspbrk.o                \
741         wsprintf.o              \
742         wsrchr.o                \
743         wsscanf.o               \
744         wsspn.o                 \
745         wstod.o                 \
746         wstok.o                 \
747         wstol.o                 \
748         wstoll.o                \
749         wsxfrm.o                \
750         gettext.o               \
751         gettext_gnu.o           \
752         gettext_real.o          \
753         gettext_util.o          \
754         plural_parser.o         \
755         wdresolve.o             \
756         _ctype.o                \
757         isascii.o               \
758         toascii.o

760 PORTI18N_COND=                  \
761         wcstol_longlong.o       \
762         wcstoul_longlong.o

764 AIOOBJS=                       \
765         aio.o                   \
766         aio_alloc.o             \
767         posix_aio.o

769 RTOBJS=                        \
770         clock_timer.o           \
771         mqueue.o                \
772         pos4obj.o               \
773         sched.o                 \
774         sem.o                   \
775         shm.o                   \
776         sigev_thread.o

778 TPOOLOBJS=                     \
779         thread_pool.o

781 THREADSOBJS=                   \
782         alloc.o                 \
783         assfail.o               \
```

```
784         cancel.o                \
785         door_calls.o            \
786         pthr_attr.o             \
787         pthr_barrier.o          \
788         pthr_cond.o             \
789         pthr_mutex.o            \
790         pthr_rwlock.o           \
791         pthread.o               \
792         rwlock.o                \
793         scalls.o                \
794         sema.o                  \
795         sigaction.o             \
796         spawn.o                 \
797         synch.o                 \
798         tdb_agent.o             \
799         thr.o                   \
800         thread_interface.o      \
801         tls.o                   \
802         tsd.o

804 THREADSMACHOBJS=               \
805         machdep.o

807 THREADSASMOBJS=                \
808         asm_subr.o

810 UNICODEOBJS=                   \
811         u8_textprep.o           \
812         uconv.o

814 UNWINDMACHOBJS=                \
815         unwind.o

817 UNWINDASMOBJS=                 \
818         unwind_frame.o

820 # objects that implement the transitional large file API
821 PORTSYS64=                     \
822         fstatat64.o             \
823         lockf64.o               \
824         openat64.o

826 PORTSYS=                       \
827         _autofssys.o            \
828         acctctl.o               \
829         bsd_signal.o            \
830         corectl.o               \
831         exacctsys.o             \
832         execl.o                 \
833         execle.o                \
834         execv.o                 \
835         faccessat.o             \
836         fsmisc.o                \
837         fstatat.o               \
838         getpagesizes.o          \
839         getpeerucred.o          \
840         inst_sync.o             \
841         issetugid.o             \
842         label.o                 \
843         libc_fcntl.o            \
844         libc_link.o             \
845         libc_open.o             \
846         lockf.o                 \
847         lwp.o                   \
848         lwp_cond.o              \
849         lwp_rwlock.o            \
```



```
850         lwp_sigmask.o           \
851         meminfosys.o            \
852         msgsys.o                \
853         nfssys.o                \
854         openat.o                \
855         pgrpsys.o               \
856         posix_sigwait.o         \
857         ppriv.o                 \
858         psetsys.o               \
859         rctlsys.o               \
860         sbrk.o                  \
861         semsys.o                \
862         set_errno.o             \
863         sharefs.o               \
864         shmsys.o                \
865         sidsys.o                \
866         siginterrupt.o          \
867         signal.o                \
868         sigpending.o            \
869         sigstack.o              \
870         tasksys.o               \
871         time.o                  \
872         time_util.o             \
873         ucontext.o              \
874         ustat.o                 \
875         zone.o

877 PORTREGEX=                     \
878         glob.o                  \
879         regcmp.o                \
880         regex.o                 \
881         wordexp.o

883 MOSTOBJS=                      \
884         $(STRETS)               \
885         $(CRTOBJS)              \
886         $(DYNOBJS)              \
887         $(FPOBJS)               \
888         $(FPASMOBJS)            \
889         $(ATOMICOBJS)           \
890         $(XATTROBJS)            \
891         $(COMOBJS)              \
892         $(DTRACEOBJS)           \
893         $(GENOBJS)              \
894         $(PORTFP)               \
895         $(PORTGEN)              \
896         $(PORTGEN64)            \
897         $(PORTI18N)             \
898         $(PORTI18N_COND)        \
899         $(PORTPRINT)            \
900         $(PORTPRINT_C89)        \
901         $(PORTPRINT_W)          \
902         $(PORTREGEX)            \
903         $(PORTSTDIO)            \
904         $(PORTSTDIO64)          \
905         $(PORTSTDIO_C89)        \
906         $(PORTSTDIO_W)          \
907         $(PORTSYS)              \
908         $(PORTSYS64)            \
909         $(AIOOBJS)              \
910         $(RTOBJS)               \
911         $(TPOOLOBJS)            \
912         $(THREADSOBJS)          \
913         $(THREADSMACHOBJS)      \
914         $(THREADSASMOBJS)       \
915         $(UNICODEOBJS)          \
```

```
916         $(UNWINDMACHOBJS)       \
917         $(UNWINDASMOBJS)        \
918         $(COMSYSOBJS)           \
919         $(SYSOBJS)              \
920         $(COMSYSOBJS64)         \
921         $(SYSOBJS64)            \
922         $(VALUES)

924 TRACEOBJS=                      \
925         plockstat.o

927 # NOTE: libc.so.1 must be linked with the minimal crti.o and crtn.o
928 # modules whose source is provided in the $(SRC)/lib/common directory.
929 # This must be done because otherwise the Sun C compiler would insert
930 # its own versions of these modules and those versions contain code
931 # to call out to C++ initialization functions.  Such C++ initialization
932 # functions can call back into libc before thread initialization is
933 # complete and this leads to segmentation violations and other problems.
934 # Since libc contains no C++ code, linking with the minimal crti.o and
935 # crtn.o modules is safe and avoids the problems described above.
936 OBJECTS= $(CRTI) $(MOSTOBJS) $(CRTN)
937 CRTSRCS= ../../common/i386

939 LDPASS_OFF=     $(POUND_SIGN)

941 # include common library definitions
942 include ../../Makefile.lib

944 # NOTE: libc_i18n.a will be part of libc.so.1.  Therefore, the compilation
945 # conditions such as the settings of CFLAGS and CPPFLAGS for the libc_i18n stuff
946 # need to be compatible with the ones for the libc stuff.  Whenever the changes
947 # that affect the compilation conditions of libc happened, those for libc_i18n
948 # also need to be updated.

950 # we need to override the default SONAME here because we might
951 # be building a variant object (still libc.so.1, but different filename)
952 SONAME = libc.so.1

954 CFLAGS += $(CCVERBOSE) $(CTF_FLAGS)

956 # This is necessary to avoid problems with calling _ex_unwind().
957 # We probably don't want any inlining anyway.
958 XINLINE = -xinline=
959 CFLAGS += $(XINLINE)

961 # Setting THREAD_DEBUG = -DTHREAD_DEBUG (make THREAD_DEBUG=-DTHREAD_DEBUG ...)
962 # enables ASSERT() checking in the threads portion of the library.
963 # This is automatically enabled for DEBUG builds, not for non-debug builds.
964 THREAD_DEBUG =
965 $(NOT_RELEASE_BUILD)THREAD_DEBUG = -DTHREAD_DEBUG

967 ALTPICS= $(TRACEOBJS:%=pics/%)

969 $(DYNLIB) := PICS += $(ROOTFS_LIBDIR)/libc_i18n.a
970 $(DYNLIB) := BUILD.SO = $(LD) -o $@ -G $(DYNFLAGS) $(PICS) $(ALTPICS) $(LDLIBS)

972 MAPFILES =      ../port/mapfile-vers ../i386/mapfile-vers

974 #
975 # EXTN_CPPFLAGS and EXTN_CFLAGS set in enclosing Makefile
976 #
977 CFLAGS +=       $(EXTN_CFLAGS)
978 CPPFLAGS=       -D_REENTRANT -Di386 $(EXTN_CPPFLAGS) $(THREAD_DEBUG) \
979                 -I$(LIBCBASE)/inc -I../inc $(CPPFLAGS.master)
980 ASFLAGS=        $(AS_PICFLAGS) -P -D__STDC__ -D_ASM $(CPPFLAGS) $(i386_AS_XARCH)
```



```
982 # Conditionally add support for making |wordexp()| check whether
983 # /usr/bin/ksh is ksh93 or not
984 include ../../../Makefile.ksh93switch
985 CPPFLAGS += -DWORDEXP_KSH93=$(ON_BUILD_KSH93_AS_BINKSH)

987 # Inform the run-time linker about libc specialized initialization
988 RTLDINFO =      -z rtldinfo=tls_rtldinfo
989 DYNFLAGS +=     $(RTLDINFO)

991 DYNFLAGS +=     -e __rtboot
992 DYNFLAGS +=     $(EXTN_DYNFLAGS)

994 # Inform the kernel about the initial DTrace area (in case
995 # libc is being used as the interpreter / runtime linker).
996 DTRACE_DATA =   -zdtrace=dtrace_data
997 DYNFLAGS +=     $(DTRACE_DATA)

999 # DTrace needs an executable data segment.
1000 MAPFILE.NED=

1002 BUILD.s=        $(AS) $(ASFLAGS) $< -o $@

1004 # Override this top level flag so the compiler builds in its native
1005 # C99 mode.  This has been enabled to support the complex arithmetic
1006 # added to libc.
1007 C99MODE=        $(C99_ENABLE)

1009 # libc method of building an archive
1010 BUILD.AR= $(RM) $@ ; \
1011         $(AR) q $@ '$(LORDER) $(MOSTOBJS:%=$(DIR)/%)| $(TSORT)'

1013 # extra files for the clean target
1014 CLEANFILES=                     \
1015         ../port/gen/errlst.c    \
1016         ../port/gen/new_list.c  \
1017         assym.h                 \
1018         genassym                \
1019         crt/_rtld.s             \
1020         crt/_rtbootld.s         \
1021         pics/_rtbootld.o        \
1022         pics/crti.o             \
1023         pics/crtn.o             \
1024         $(ALTPICS)

1026 CLOBBERFILES += $(LIB_PIC)

1028 # list of C source for lint
1029 SRCS=                                                   \
1030         $(ATOMICOBJS:%.o=$(SRC)/common/atomic/%.c)      \
1031         $(XATTROBJS:%.o=$(SRC)/common/xattr/%.c)        \
1032         $(COMOBJS:%.o=$(SRC)/common/util/%.c)           \
1033         $(DTRACEOBJS:%.o=$(SRC)/common/dtrace/%.c)      \
1034         $(PORTFP:%.o=../port/fp/%.c)                    \
1035         $(PORTGEN:%.o=../port/gen/%.c)                  \
1036         $(PORTI18N:%.o=../port/i18n/%.c)                \
1037         $(PORTPRINT:%.o=../port/print/%.c)              \
1038         $(PORTREGEX:%.o=../port/regex/%.c)              \
1039         $(PORTSTDIO:%.o=../port/stdio/%.c)              \
1040         $(PORTSYS:%.o=../port/sys/%.c)                  \
1041         $(AIOOBJS:%.o=../port/aio/%.c)                  \
1042         $(RTOBJS:%.o=../port/rt/%.c)                    \
1043         $(TPOOLOBJS:%.o=../port/tpool/%.c)              \
1044         $(THREADSOBJS:%.o=../port/threads/%.c)          \
1045         $(THREADSMACHOBJS:%.o=../$(MACH)/threads/%.c)   \
1046         $(UNICODEOBJS:%.o=$(SRC)/common/unicode/%.c)    \
1047         $(UNWINDMACHOBJS:%.o=../port/unwind/%.c)        \
```

```
1048         $(FPOBJS:%.o=../$(MACH)/fp/%.c)                \
1049         $(LIBCBASE)/gen/ecvt.c                         \
1050         $(LIBCBASE)/gen/makectxt.c                     \
1051         $(LIBCBASE)/gen/siginfolst.c                   \
1052         $(LIBCBASE)/gen/siglongjmp.c                   \
1053         $(LIBCBASE)/gen/strcmp.c                       \
1054         $(LIBCBASE)/gen/sync_instruction_memory.c      \
1055         $(LIBCBASE)/sys/ptrace.c                       \
1056         $(LIBCBASE)/sys/uadmin.c

1058 # conditional assignments
1059 $(DYNLIB) := CRTI = crti.o
1060 $(DYNLIB) := CRTN = crtn.o

1062 # Files which need the threads .il inline template
1063 TIL=                            \
1064         aio.o                   \
1065         alloc.o                 \
1066         assfail.o               \
1067         atexit.o                \
1068         atfork.o                \
1069         cancel.o                \
1070         door_calls.o            \
1071         errno.o                 \
1072         lwp.o                   \
1073         ma.o                    \
1074         machdep.o               \
1075         posix_aio.o             \
1076         pthr_attr.o             \
1077         pthr_barrier.o          \
1078         pthr_cond.o             \
1079         pthr_mutex.o            \
1080         pthr_rwlock.o           \
1081         pthread.o               \
1082         rand.o                  \
1083         rwlock.o                \
1084         scalls.o                \
1085         sched.o                 \
1086         sema.o                  \
1087         sigaction.o             \
1088         sigev_thread.o          \
1089         spawn.o                 \
1090         stack.o                 \
1091         synch.o                 \
1092         tdb_agent.o             \
1093         thr.o                   \
1094         thread_interface.o      \
1095         thread_pool.o           \
1096         tls.o                   \
1097         tsd.o                   \
1098         unwind.o

1100 THREADS_INLINES = $(LIBCBASE)/threads/i386.il
1101 $(TIL:%=pics/%) := CFLAGS += $(THREADS_INLINES)

1103 # pics/mul64.o := CFLAGS += $(LIBCBASE)/crt/mul64.il

1105 # large-file-aware components that should be built large

1107 $(COMSYSOBJS64:%=pics/%) := \
1108         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1110 $(SYSOBJS64:%=pics/%) := \
1111         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1113 $(PORTGEN64:%=pics/%) := \
```



```
1114         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1116 $(PORTSTDIO64:%=pics/%) := \
1117         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1119 $(PORTSYS64:%=pics/%) := \
1120         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1122 $(PORTSTDIO_W:%=pics/%) := \
1123         CPPFLAGS += -D_WIDE

1125 $(PORTPRINT_W:%=pics/%) := \
1126         CPPFLAGS += -D_WIDE

1128 $(PORTPRINT_C89:%=pics/%) := \
1129         CPPFLAGS += -D_C89_INTMAX32

1131 $(PORTSTDIO_C89:%=pics/%) := \
1132         CPPFLAGS += -D_C89_INTMAX32

1134 $(PORTI18N_COND:%=pics/%) := \
1135         CPPFLAGS += -D_WCS_LONGLONG

1137 .KEEP_STATE:

1139 all: $(LIBS) $(LIB_PIC)

1141 lint    :=      CPPFLAGS += -I../$(MACH)/fp
1142 lint    :=      CPPFLAGS += -D_MSE_INT_H -D_LCONV_C99
1143 lint    :=      LINTFLAGS += -mn -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED

1145 lint:
1146         @echo $(LINT.c) ...
1147         @$(LINT.c) $(SRCS) $(LDLIBS)

1149 $(LINTLIB):= SRCS=../port/llib-lc
1150 $(LINTLIB):= CPPFLAGS += -D_MSE_INT_H
1151 $(LINTLIB):= LINTFLAGS=-nvx

1153 # object files that depend on inline template
1154 $(TIL:%=pics/%): $(LIBCBASE)/threads/i386.il
1155 # pics/mul64.o: $(LIBCBASE)/crt/mul64.il

1157 # include common libc targets
1158 include ../Makefile.targ

1160 # We need to strip out all CTF and DOF data from the static library
1161 $(LIB_PIC) := DIR = pics
1162 $(LIB_PIC): pics $$(PICS)
1163         $(BUILD.AR)
1164         $(MCS) -d -n .SUNW_ctf $@ > /dev/null 2>&1
1165         $(MCS) -d -n .SUNW_dof $@ > /dev/null 2>&1
1166         $(AR) -ts $@ > /dev/null
1167         $(POST_PROCESS_A)

1169 $(LIBCBASE)/crt/_rtbootld.s: $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.c
1170         $(CC) $(CPPFLAGS) $(CTF_FLAGS) -O -S $(C_PICFLAGS) \
1171             $(LIBCBASE)/crt/_rtld.c -o $(LIBCBASE)/crt/_rtld.s
1172         $(CAT) $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.s > $@
1173         $(RM) $(LIBCBASE)/crt/_rtld.s

1175 # partially built from C source
1176 pics/_rtbootld.o: $(LIBCBASE)/crt/_rtbootld.s
1177         $(AS) $(ASFLAGS) $(LIBCBASE)/crt/_rtbootld.s -o $@
1178         $(CTFCONVERT_O)
```

```
1180 ASSYMDEP_OBJS=                  \
1181         _lwp_mutex_unlock.o     \
1182         _stack_grow.o           \
1183         getcontext.o            \
1184         tls_get_addr.o          \
1185         vforkx.o

1187 $(ASSYMDEP_OBJS:%=pics/%)       :=      CPPFLAGS += -I.

1189 $(ASSYMDEP_OBJS:%=pics/%): assym.h

1191 # assym.h build rules

1193 GENASSYM_C = ../$(MACH)/genassym.c

1195 # XXX   A hack.  Perhaps this should be 'CPPFLAGS.native' and
1196 #       live in Makefile.master

1198 CPPFLAGS.genassym = \
1199         $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4)

1201 genassym: $(GENASSYM_C)
1202         $(NATIVECC) -I$(LIBCBASE)/inc -I../inc  \
1203                 -D__EXTENSIONS__ $(CPPFLAGS.genassym) -o $@ $(GENASSYM_C)

1205 OFFSETS = ../$(MACH)/offsets.in

1207 assym.h: $(OFFSETS) genassym
1208         $(OFFSETS_CREATE) <$(OFFSETS) >$@
1209         ./genassym >>$@

1211 # derived C source and related explicit dependencies
1212 ../port/gen/errlst.c + \
1213 ../port/gen/new_list.c: ../port/gen/errlist ../port/gen/errlist.awk
1214         cd ../port/gen; pwd; $(AWK) -f errlist.awk < errlist

1216 pics/errlst.o: ../port/gen/errlst.c

1218 pics/new_list.o: ../port/gen/new_list.c
```

```
**********************************************************
    1405 Wed Aug 27 14:23:16 2008
new/usr/src/lib/libc/i386/gen/byteorder64.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/isa_defs.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <inttypes.h>

#if (defined(_BIG_ENDIAN) || defined(_LP64)) && !defined(__lint)

#error  Use ISA-dependent byteorder64.c only on a 32-bit little-endian machine.

#else

uint64_t
htonll(uint64_t in)
{
        return (htonl(in >> 32) | ((uint64_t)htonl(in) << 32));
}

uint64_t
ntohll(uint64_t in)
{
        return (ntohl(in >> 32) | (uint64_t)ntohl(in) << 32);
}

#endif  /* (_BIG_ENDIAN) || _LP64) && !__lint */
```

```
**********************************************************
   30660 Wed Aug 27 14:23:19 2008
new/usr/src/lib/libc/port/mapfile-vers
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
#
# 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 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# All function names added to this or any other libc mapfile
# must be placed under the 'protected:' designation.
# The 'global:' designation is used *only* for data
# items and for the members of the malloc() family.
#

SUNW_1.23 {             # SunOS 5.11 (Solaris 11)
    global:
        _nl_domain_bindings;
        _nl_msg_cat_cntr;
    protected:
        addrtosymstr;
        aio_cancel;
        aiocancel;
        aio_error;
        aio_fsync;
        aio_read;
        aioread;
        aio_return;
        aio_suspend;
        aiowait;
        aio_waitn;
        aio_write;
        aiowrite;
        assfail;
        backtrace;
        backtrace_symbols;
        backtrace_symbols_fd;
        clock_getres;
        clock_gettime;
        clock_nanosleep;
```



```
        clock_settime;
        dirfd;
        door_bind;
        door_call;
        door_create;
        door_cred;
        door_getparam;
        door_info;
        door_return;
        door_revoke;
        door_server_create;
        door_setparam;
        door_ucred;
        door_unbind;
        err;
        errx;
        fdatasync;
        fgetattr;
        forkallx;
        forkx;
        fsetattr;
        getattrat;
        getpagesizes2;
        htonl;
        htonll;
        htons;
        lio_listio;
        mkdtemp;
        mkstemps;
        mq_close;
        mq_getattr;
        mq_notify;
        mq_open;
        mq_receive;
        mq_reltimedreceive_np;
        mq_reltimedsend_np;
        mq_send;
        mq_setattr;
        mq_timedreceive;
        mq_timedsend;
        mq_unlink;
        nanosleep;
        ntohl;
        ntohll;
        ntohs;
        posix_fadvise;
        posix_fallocate;
        posix_madvise;
        posix_memalign;
        pthread_key_create_once_np;
        sched_getparam;
        sched_get_priority_max;
        sched_get_priority_min;
        sched_getscheduler;
        sched_rr_get_interval;
        sched_setparam;
        sched_setscheduler;
        sched_yield;
        sem_close;
        sem_destroy;
        sem_getvalue;
        sem_init;
        sem_open;
        sem_post;
        sem_reltimedwait_np;
        sem_timedwait;
```

```
 124         sem_trywait;
 125         sem_unlink;
 126         sem_wait;
 127         setattrat;
 128         _sharefs;
 129         shm_open;
 130         shm_unlink;
 131         sigqueue;
 132         sigtimedwait;
 133         sigwaitinfo;
 134         strnlen;
 135         thr_keycreate_once;
 136         timer_create;
 137         timer_delete;
 138         timer_getoverrun;
 139         timer_gettime;
 140         timer_settime;
 141         u8_strcmp;
 142         u8_textprep_str;
 143         u8_validate;
 144         uconv_u16tou32;
 145         uconv_u16tou8;
 146         uconv_u32tou16;
 147         uconv_u32tou8;
 148         uconv_u8tou16;
 149         uconv_u8tou32;
 150         uucopy;
 151         uucopystr;
 152         vforkx;
 153         verr;
 154         verrx;
 155         vwarn;
 156         vwarnx;
 157         warn;
 158         warnx;
 159 } SUNW_1.22.3;
```

*unchanged_portion_omitted_*

```
**********************************************************
    1687 Wed Aug 27 14:23:24 2008
new/usr/src/lib/libc/sparc/gen/byteorder.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/isa_defs.h>
#include <sys/types.h>


#if defined(_LITTLE_ENDIAN) && !defined(__lint)

#error  Use ISA-specific byteorder.s on a little-endian machine.

#else   /* !_LITTLE_ENDIAN */

/*
 * htonll(), ntohll(), htonl(), ntohl(), htons(), ntohs()
 * These functions just return the input parameter, as the host
 * byte order is the same as the network byte order (big endian).
 * On little endian machines, these functions byte swap.
 */

uint64_t
htonll(uint64_t in)
{
        return (in);
}

uint64_t
ntohll(uint64_t in)
{
        return (in);
}

uint32_t
```



```
htonl(uint32_t in)
{
        return (in);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1687 Wed Aug 27 14:23:29 2008
new/usr/src/lib/libc/sparcv9/gen/byteorder.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/isa_defs.h>
#include <sys/types.h>


#if defined(_LITTLE_ENDIAN) && !defined(__lint)

#error  Use ISA-specific byteorder.s on a little-endian machine.

#else   /* !_LITTLE_ENDIAN */

/*
 * htonll(), ntohll(), htonl(), ntohl(), htons(), ntohs()
 * These functions just return the input parameter, as the host
 * byte order is the same as the network byte order (big endian).
 * On little endian machines, these functions byte swap.
 */

uint64_t
htonll(uint64_t in)
{
        return (in);
}

uint64_t
ntohll(uint64_t in)
{
        return (in);
}

uint32_t
```



```
htonl(uint32_t in)
{
        return (in);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   47724 Wed Aug 27 14:23:34 2008
new/usr/src/lib/libdhcputil/common/dhcp_inittab.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/types.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <stdarg.h>
#include <limits.h>
#include <ctype.h>
#include <libgen.h>
#include <sys/isa_defs.h>
#include <sys/socket.h>
#include <net/if_arp.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/sysmacros.h>
#include <libinetutil.h>
#include <libdlpi.h>
#include <netinet/dhcp6.h>

#include "dhcp_symbol.h"
#include "dhcp_inittab.h"

static uint64_t         dhcp_htonll(uint64_t);
static uint64_t         dhcp_ntohll(uint64_t);
static void             inittab_msg(const char *, ...);
static uchar_t          category_to_code(const char *);
static boolean_t        encode_number(uint8_t, uint8_t, boolean_t, uint8_t,
                            const char *, uint8_t *, int *);
static boolean_t        decode_number(uint8_t, uint8_t, boolean_t, uint8_t,
                            const uint8_t *, char *, int *);
static dhcp_symbol_t    *inittab_lookup(uchar_t, char, const char *, int32_t,
```



```
                            size_t *);
static dsym_category_t  itabcode_to_dsymcode(uchar_t);
static boolean_t        parse_entry(char *, char **);

/*
 * forward declaration of our internal inittab_table[].  too bulky to put
 * up front -- check the end of this file for its definition.
 *
 * Note: we have only an IPv4 version here.  The inittab_verify() function is
 * used by the DHCP server and manager.  We'll need a new function if the
 * server is extended to DHCPv6.
 */
static dhcp_symbol_t    inittab_table[];

/*
 * the number of fields in the inittab and names for the fields.  note that
 * this order is meaningful to parse_entry(); other functions should just
 * use them as indexes into the array returned from parse_entry().
 */
#define ITAB_FIELDS     7
enum { ITAB_NAME, ITAB_CODE, ITAB_TYPE, ITAB_GRAN, ITAB_MAX, ITAB_CONS,
    ITAB_CAT };

/*
 * the category_map_entry_t is used to map the inittab category codes to
 * the dsym codes.  the reason the codes are different is that the inittab
 * needs to have the codes be ORable such that queries can retrieve more
 * than one category at a time.  this map is also used to map the inittab
 * string representation of a category to its numerical code.
 */
typedef struct category_map_entry {
        dsym_category_t cme_dsymcode;
        char            *cme_name;
        uchar_t         cme_itabcode;
} category_map_entry_t;
______unchanged_portion_omitted_

/*
 * decode_number(): decodes a sequence of numbers from binary into ascii;
 *                  binary is coming off of the network, so it is in nbo
 *
 *   input: uint8_t: the number of "granularity" numbers to decode
 *          uint8_t: the length of each number
 *          boolean_t: whether the numbers should be considered signed
 *          uint8_t: the number of numbers per granularity
 *          const uint8_t *: where to decode the numbers from
 *          char *: where to decode the numbers to
 *  output: boolean_t: true on successful conversion, false on failure
 */

static boolean_t
decode_number(uint8_t n_entries, uint8_t size, boolean_t is_signed,
    uint8_t granularity, const uint8_t *from, char *to, int *ierrnop)
{
        uint16_t        uint16;
        uint32_t        uint32;
        uint64_t        uint64;

        if (granularity != 0) {
                if ((granularity % n_entries) != 0) {
                        inittab_msg("decode_number: number of entries "
                            "not compatible with option granularity");
                        *ierrnop = ITAB_BAD_GRAN;
                        return (B_FALSE);
                }
        }
```

```
1447         for (; n_entries != 0; n_entries--, from += size) {

1449                 switch (size) {

1451                 case 1:
1452                         to += sprintf(to, is_signed ? "%d" : "%u", *from);
1453                         break;

1455                 case 2:
1456                         (void) memcpy(&uint16, from, 2);
1457                         to += sprintf(to, is_signed ? "%hd" : "%hu",
1458                             ntohs(uint16));
1459                         break;

1461                 case 3:
1462                         uint32 = 0;
1463                         (void) memcpy((uchar_t *)&uint32 + 1, from, 3);
1464                         to += sprintf(to, is_signed ? "%ld" : "%lu",
1465                             ntohl(uint32));
1466                         break;

1468                 case 4:
1469                         (void) memcpy(&uint32, from, 4);
1470                         to += sprintf(to, is_signed ? "%ld" : "%lu",
1471                             ntohl(uint32));
1472                         break;

1474                 case 8:
1475                         (void) memcpy(&uint64, from, 8);
1476                         to += sprintf(to, is_signed ? "%lld" : "%llu",
1477                             ntohll(uint64));
1481                             dhcp_ntohll(uint64));
1478                         break;

1480                 default:
1481                         *ierrnop = ITAB_BAD_NUMBER;
1482                         inittab_msg("decode_number: unknown integer size '%d'",
1483                             size);
1484                         return (B_FALSE);
1485                 }
1486                 if (n_entries > 0)
1487                         *to++ = ' ';
1488         }

1490         *to = '\0';
1491         return (B_TRUE);
1492 }

1494 /*
1495  * encode_number(): encodes a sequence of numbers from ascii into binary;
1496  *                  number will end up on the wire so it needs to be in nbo
1497  *
1498  *   input: uint8_t: the number of "granularity" numbers to encode
1499  *          uint8_t: the length of each number
1500  *          boolean_t: whether the numbers should be considered signed
1501  *          uint8_t: the number of numbers per granularity
1502  *          const uint8_t *: where to encode the numbers from
1503  *          char *: where to encode the numbers to
1504  *          int *: set to extended error code if error occurs.
1505  *  output: boolean_t: true on successful conversion, false on failure
1506  */

1508 static boolean_t /* ARGSUSED */
1509 encode_number(uint8_t n_entries, uint8_t size, boolean_t is_signed,
1510     uint8_t granularity, const char *from, uint8_t *to, int *ierrnop)
```



```
1511 {
1512         uint8_t         i;
1513         uint16_t        uint16;
1514         uint32_t        uint32;
1515         uint64_t        uint64;
1516         char            *endptr;

1518         if (granularity != 0) {
1519                 if ((granularity % n_entries) != 0) {
1520                         *ierrnop = ITAB_BAD_GRAN;
1521                         inittab_msg("encode_number: number of entries "
1522                             "not compatible with option granularity");
1523                         return (B_FALSE);
1524                 }
1525         }

1527         for (i = 0; i < n_entries; i++, from++, to += size) {

1529                 /*
1530                  * totally obscure c factoid: it is legal to pass a
1531                  * string representing a negative number to strtoul().
1532                  * in this case, strtoul() will return an unsigned
1533                  * long that if cast to a long, would represent the
1534                  * negative number.  we take advantage of this to
1535                  * cut down on code here.
1536                  */

1538                 errno = 0;
1539                 switch (size) {

1541                 case 1:
1542                         *to = strtoul(from, &endptr, 0);
1543                         if (errno != 0 || from == endptr) {
1544                                 goto error;
1545                         }
1546                         break;

1548                 case 2:
1549                         uint16 = htons(strtoul(from, &endptr, 0));
1550                         if (errno != 0 || from == endptr) {
1551                                 goto error;
1552                         }
1553                         (void) memcpy(to, &uint16, 2);
1554                         break;

1556                 case 3:
1557                         uint32 = htonl(strtoul(from, &endptr, 0));
1558                         if (errno != 0 || from == endptr) {
1559                                 goto error;
1560                         }
1561                         (void) memcpy(to, (uchar_t *)&uint32 + 1, 3);
1562                         break;

1564                 case 4:
1565                         uint32 = htonl(strtoul(from, &endptr, 0));
1566                         if (errno != 0 || from == endptr) {
1567                                 goto error;
1568                         }
1569                         (void) memcpy(to, &uint32, 4);
1570                         break;

1572                 case 8:
1573                         uint64 = htonll(strtoull(from, &endptr, 0));
1577                         uint64 = dhcp_htonll(strtoull(from, &endptr, 0));
1574                         if (errno != 0 || from == endptr) {
1575                                 goto error;
```

```
1576                        }
1577                        (void) memcpy(to, &uint64, 8);
1578                        break;

1580                default:
1581                        inittab_msg("encode_number: unsupported integer "
1582                            "size '%d'", size);
1583                        return (B_FALSE);
1584                }

1586                from = strchr(from, ' ');
1587                if (from == NULL)
1588                        break;
1589        }

1591        return (B_TRUE);

1593 error:
1594        *ierrnop = ITAB_BAD_NUMBER;
1595        inittab_msg("encode_number: cannot convert to integer");
1596        return (B_FALSE);
1597 }
```

*unchanged_portion_omitted_*

```
1697 /*
1698  * dhcp_htonll(): converts a 64-bit number from host to network byte order
1699  *
1700  *   input: uint64_t: the number to convert
1701  *  output: uint64_t: its value in network byte order
1702  */

1704 static uint64_t
1705 dhcp_htonll(uint64_t uint64_hbo)
1706 {
1707        return (dhcp_ntohll(uint64_hbo));
1708 }

1710 /*
1711  * dhcp_ntohll(): converts a 64-bit number from network to host byte order
1712  *
1713  *   input: uint64_t: the number to convert
1714  *  output: uint64_t: its value in host byte order
1715  */

1717 static uint64_t
1718 dhcp_ntohll(uint64_t uint64_nbo)
1719 {
1720 #ifdef  _LITTLE_ENDIAN
1721        return ((uint64_t)ntohl(uint64_nbo & 0xffffffff) << 32 |
1722            ntohl(uint64_nbo >> 32));
1723 #else
1724        return (uint64_nbo);
1725 #endif
1726 }
```

```
1693 /*
1694  * our internal table of DHCP option values, used by inittab_verify()
1695  */
1696 static dhcp_symbol_t inittab_table[] =
1697 {
1698 { DSYM_INTERNAL,        1024,   "Hostname",     DSYM_BOOL,      0,      0 },
1699 { DSYM_INTERNAL,        1025,   "LeaseNeg",     DSYM_BOOL,      0,      0 },
1700 { DSYM_INTERNAL,        1026,   "EchoVC",       DSYM_BOOL,      0,      0 },
1701 { DSYM_INTERNAL,        1027,   "BootPath",     DSYM_ASCII,     1,      128 },
1702 { DSYM_FIELD,           0,      "Opcode",       DSYM_UNUMBER8,  1,      1 },
1703 { DSYM_FIELD,           1,      "Htype",        DSYM_UNUMBER8,  1,      1 },
```



```
1704 { DSYM_FIELD,           2,      "HLen",         DSYM_UNUMBER8,  1,      1 },
1705 { DSYM_FIELD,           3,      "Hops",         DSYM_UNUMBER8,  1,      1 },
1706 { DSYM_FIELD,           4,      "Xid",          DSYM_UNUMBER32, 1,      1 },
1707 { DSYM_FIELD,           8,      "Secs",         DSYM_UNUMBER16, 1,      1 },
1708 { DSYM_FIELD,           10,     "Flags",        DSYM_OCTET,     1,      2 },
1709 { DSYM_FIELD,           12,     "Ciaddr",       DSYM_IP,        1,      1 },
1710 { DSYM_FIELD,           16,     "Yiaddr",       DSYM_IP,        1,      1 },
1711 { DSYM_FIELD,           20,     "BootSrvA",     DSYM_IP,        1,      1 },
1712 { DSYM_FIELD,           24,     "Giaddr",       DSYM_IP,        1,      1 },
1713 { DSYM_FIELD,           28,     "Chaddr",       DSYM_OCTET,     1,      16 },
1714 { DSYM_FIELD,           44,     "BootSrvN",     DSYM_ASCII,     1,      64 },
1715 { DSYM_FIELD,           108,    "BootFile",     DSYM_ASCII,     1,      128 },
1716 { DSYM_FIELD,           236,    "Magic",        DSYM_OCTET,     1,      4 },
1717 { DSYM_FIELD,           240,    "Options",      DSYM_OCTET,     1,      60 },
1718 { DSYM_STANDARD,        1,      "Subnet",       DSYM_IP,        1,      1 },
1719 { DSYM_STANDARD,        2,      "UTCoffst",     DSYM_SNUMBER32, 1,      1 },
1720 { DSYM_STANDARD,        3,      "Router",       DSYM_IP,        1,      0 },
1721 { DSYM_STANDARD,        4,      "Timeserv",     DSYM_IP,        1,      0 },
1722 { DSYM_STANDARD,        5,      "IEN116ns",     DSYM_IP,        1,      0 },
1723 { DSYM_STANDARD,        6,      "DNSserv",      DSYM_IP,        1,      0 },
1724 { DSYM_STANDARD,        7,      "Logserv",      DSYM_IP,        1,      0 },
1725 { DSYM_STANDARD,        8,      "Cookie",       DSYM_IP,        1,      0 },
1726 { DSYM_STANDARD,        9,      "Lprserv",      DSYM_IP,        1,      0 },
1727 { DSYM_STANDARD,        10,     "Impress",      DSYM_IP,        1,      0 },
1728 { DSYM_STANDARD,        11,     "Resource",     DSYM_IP,        1,      0 },
1729 { DSYM_STANDARD,        12,     "Hostname",     DSYM_ASCII,     1,      0 },
1730 { DSYM_STANDARD,        13,     "Bootsize",     DSYM_UNUMBER16, 1,      1 },
1731 { DSYM_STANDARD,        14,     "Dumpfile",     DSYM_ASCII,     1,      0 },
1732 { DSYM_STANDARD,        15,     "DNSdmain",     DSYM_ASCII,     1,      0 },
1733 { DSYM_STANDARD,        16,     "Swapserv",     DSYM_IP,        1,      1 },
1734 { DSYM_STANDARD,        17,     "Rootpath",     DSYM_ASCII,     1,      0 },
1735 { DSYM_STANDARD,        18,     "ExtendP",      DSYM_ASCII,     1,      0 },
1736 { DSYM_STANDARD,        19,     "IpFwdF",       DSYM_UNUMBER8,  1,      1 },
1737 { DSYM_STANDARD,        20,     "NLrouteF",     DSYM_UNUMBER8,  1,      1 },
1738 { DSYM_STANDARD,        21,     "PFilter",      DSYM_IP,        2,      0 },
1739 { DSYM_STANDARD,        22,     "MaxIpSiz",     DSYM_UNUMBER16, 1,      1 },
1740 { DSYM_STANDARD,        23,     "IpTTL",        DSYM_UNUMBER8,  1,      1 },
1741 { DSYM_STANDARD,        24,     "PathTO",       DSYM_UNUMBER32, 1,      1 },
1742 { DSYM_STANDARD,        25,     "PathTbl",      DSYM_UNUMBER16, 1,      0 },
1743 { DSYM_STANDARD,        26,     "MTU",          DSYM_UNUMBER16, 1,      1 },
1744 { DSYM_STANDARD,        27,     "SameMtuF",     DSYM_UNUMBER8,  1,      1 },
1745 { DSYM_STANDARD,        28,     "Broadcst",     DSYM_IP,        1,      1 },
1746 { DSYM_STANDARD,        29,     "MaskDscF",     DSYM_UNUMBER8,  1,      1 },
1747 { DSYM_STANDARD,        30,     "MaskSupF",     DSYM_UNUMBER8,  1,      1 },
1748 { DSYM_STANDARD,        31,     "RDiscvyF",     DSYM_UNUMBER8,  1,      1 },
1749 { DSYM_STANDARD,        32,     "RSolictS",     DSYM_IP,        1,      1 },
1750 { DSYM_STANDARD,        33,     "StaticRt",     DSYM_IP,        2,      0 },
1751 { DSYM_STANDARD,        34,     "TrailerF",     DSYM_UNUMBER8,  1,      1 },
1752 { DSYM_STANDARD,        35,     "ArpTimeO",     DSYM_UNUMBER32, 1,      1 },
1753 { DSYM_STANDARD,        36,     "EthEncap",     DSYM_UNUMBER8,  1,      1 },
1754 { DSYM_STANDARD,        37,     "TcpTTL",       DSYM_UNUMBER8,  1,      1 },
1755 { DSYM_STANDARD,        38,     "TcpKaInt",     DSYM_UNUMBER32, 1,      1 },
1756 { DSYM_STANDARD,        39,     "TcpKaGbF",     DSYM_UNUMBER8,  1,      1 },
1757 { DSYM_STANDARD,        40,     "NISdmain",     DSYM_ASCII,     1,      0 },
1758 { DSYM_STANDARD,        41,     "NISservs",     DSYM_IP,        1,      0 },
1759 { DSYM_STANDARD,        42,     "NTPservs",     DSYM_IP,        1,      0 },
1760 { DSYM_STANDARD,        43,     "Vendor",       DSYM_OCTET,     1,      0 },
1761 { DSYM_STANDARD,        44,     "NetBNms",      DSYM_IP,        1,      0 },
1762 { DSYM_STANDARD,        45,     "NetBDsts",     DSYM_IP,        1,      0 },
1763 { DSYM_STANDARD,        46,     "NetBNdT",      DSYM_UNUMBER8,  1,      1 },
1764 { DSYM_STANDARD,        47,     "NetBScop",     DSYM_ASCII,     1,      0 },
1765 { DSYM_STANDARD,        48,     "XFontSrv",     DSYM_IP,        1,      0 },
1766 { DSYM_STANDARD,        49,     "XDispMgr",     DSYM_IP,        1,      0 },
1767 { DSYM_STANDARD,        50,     "ReqIP",        DSYM_IP,        1,      1 },
1768 { DSYM_STANDARD,        51,     "LeaseTim",     DSYM_UNUMBER32, 1,      1 },
1769 { DSYM_STANDARD,        52,     "OptOvrld",     DSYM_UNUMBER8,  1,      1 },
```

```
1770 { DSYM_STANDARD,        53,     "DHCPType",     DSYM_UNUMBER8,  1,      1 },
1771 { DSYM_STANDARD,        54,     "ServerID",     DSYM_IP,        1,      1 },
1772 { DSYM_STANDARD,        55,     "ReqList",      DSYM_OCTET,     1,      0 },
1773 { DSYM_STANDARD,        56,     "Message",      DSYM_ASCII,     1,      0 },
1774 { DSYM_STANDARD,        57,     "DHCP_MTU",     DSYM_UNUMBER16, 1,      1 },
1775 { DSYM_STANDARD,        58,     "T1Time",       DSYM_UNUMBER32, 1,      1 },
1776 { DSYM_STANDARD,        59,     "T2Time",       DSYM_UNUMBER32, 1,      1 },
1777 { DSYM_STANDARD,        60,     "ClassID",      DSYM_ASCII,     1,      0 },
1778 { DSYM_STANDARD,        61,     "ClientID",     DSYM_OCTET,     1,      0 },
1779 { DSYM_STANDARD,        62,     "NW_dmain",     DSYM_ASCII,     1,      0 },
1780 { DSYM_STANDARD,        63,     "NWIPOpts",     DSYM_OCTET,     1,      128 },
1781 { DSYM_STANDARD,        64,     "NIS+dom",      DSYM_ASCII,     1,      0 },
1782 { DSYM_STANDARD,        65,     "NIS+serv",     DSYM_IP,        1,      0 },
1783 { DSYM_STANDARD,        66,     "TFTPsrvN",     DSYM_ASCII,     1,      64 },
1784 { DSYM_STANDARD,        67,     "OptBootF",     DSYM_ASCII,     1,      128 },
1785 { DSYM_STANDARD,        68,     "MblIPAgt",     DSYM_IP,        1,      0 },
1786 { DSYM_STANDARD,        69,     "SMTPserv",     DSYM_IP,        1,      0 },
1787 { DSYM_STANDARD,        70,     "POP3serv",     DSYM_IP,        1,      0 },
1788 { DSYM_STANDARD,        71,     "NNTPserv",     DSYM_IP,        1,      0 },
1789 { DSYM_STANDARD,        72,     "WWWservs",     DSYM_IP,        1,      0 },
1790 { DSYM_STANDARD,        73,     "Fingersv",     DSYM_IP,        1,      0 },
1791 { DSYM_STANDARD,        74,     "IRCservs",     DSYM_IP,        1,      0 },
1792 { DSYM_STANDARD,        75,     "STservs",      DSYM_IP,        1,      0 },
1793 { DSYM_STANDARD,        76,     "STDAservs",    DSYM_IP,        1,      0 },
1794 { DSYM_STANDARD,        77,     "UserClas",     DSYM_ASCII,     1,      0 },
1795 { DSYM_STANDARD,        78,     "SLP_DA",       DSYM_OCTET,     1,      0 },
1796 { DSYM_STANDARD,        79,     "SLP_SS",       DSYM_OCTET,     1,      0 },
1797 { DSYM_STANDARD,        82,     "AgentOpt",     DSYM_OCTET,     1,      0 },
1798 { DSYM_STANDARD,        89,     "FQDN",         DSYM_OCTET,     1,      0 },
1799 { 0,                    0,      "",             0,              0,      0 }
1800 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2674 Wed Aug 27 14:23:39 2008
new/usr/src/stand/lib/xdr/byteorder.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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.
 * Common Development and Distribution License, Version 1.0 only
 * (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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 1991-2000, 2003 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/types.h>
#include <netinet/in.h>

/*
 * htonll(), ntohll(), htonl(), ntohl(), htons(), ntohs()
 *
 * On little endian machines these functions reverse the byte order of the
 * input parameter and returns the result.  This is to convert the byte order
 * from host byte order (little endian) to network byte order (big endian),
 * or vice versa.
 *
 * On big endian machines these functions just return the input parameter,
 * as the host byte order is the same as the network byte order (big endian).
 */


#ifdef  _LITTLE_ENDIAN
uint64_t
htonll(uint64_t in)
{
        return ((uint64_t)htonl((in >> 32) & 0xffffffff) |
            ((uint64_t)htonl(in & 0xffffffff) << 32));
}

uint64_t
ntohll(uint64_t in)
{
        return ((uint64_t)ntohl((in >> 32) & 0xffffffff) |
```



```
            ((uint64_t)ntohl(in & 0xffffffff) << 32));
}

uint32_t
htonl(uint32_t in)
{
        uint32_t        i;

        i = (uint32_t)((in & (uint32_t)0xff000000) >> 24) +
            (uint32_t)((in & (uint32_t)0x00ff0000) >> 8) +
            (uint32_t)((in & (uint32_t)0x0000ff00) << 8) +
            (uint32_t)((in & (uint32_t)0x000000ff) << 24);
        return (i);
}
______unchanged_portion_omitted_

#else   /* _LITTLE_ENDIAN */

#if defined(lint)
uint64_t
htonll(uint64_t in)
{
        return (in);
}

uint64_t
ntohll(uint64_t in)
{
        return (in);
}

uint32_t
htonl(uint32_t in)
{
        return (in);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   21831 Wed Aug 27 14:23:43 2008
new/usr/src/uts/common/crypto/io/dca_3des.c
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * Deimos - cryptographic acceleration based upon Broadcom 582x.
 */

#include <sys/types.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/kmem.h>
#include <sys/note.h>
#include <sys/crypto/common.h>
#include <sys/crypto/spi.h>
#include <sys/crypto/dca.h>

#if defined(__i386) || defined(__amd64)
#include <sys/byteorder.h>
#define UNALIGNED_POINTERS_PERMITTED
#endif

/*
 * 3DES implementation.
 */

static int dca_3desstart(dca_t *, uint32_t, dca_request_t *);
static void dca_3desdone(dca_request_t *, int);


int
dca_3des(crypto_ctx_t *ctx, crypto_data_t *in,
    crypto_data_t *out, crypto_req_handle_t req, int flags)
```



```
{
        int                     len;
        int                     rv;
        dca_request_t           *reqp = ctx->cc_provider_private;
        dca_request_t           *des_ctx = ctx->cc_provider_private;
        dca_t                   *dca = ctx->cc_provider;
        crypto_data_t           *nin = &reqp->dr_ctx.in_dup;

        len = dca_length(in);
        if (len % DESBLOCK) {
                DBG(dca, DWARN, "input not an integral number of DES blocks");
                (void) dca_free_context(ctx);
                if (flags & DR_DECRYPT) {
                        return (CRYPTO_ENCRYPTED_DATA_LEN_RANGE);
                } else {
                        return (CRYPTO_DATA_LEN_RANGE);
                }
        }

        /*
         * If cd_miscdata non-null then this contains the IV.
         */
        if (in->cd_miscdata != NULL) {
#ifdef UNALIGNED_POINTERS_PERMITTED
                uint32_t        *p = (uint32_t *)in->cd_miscdata;
                des_ctx->dr_ctx.iv[0] = htonl(p[0]);
                des_ctx->dr_ctx.iv[1] = htonl(p[1]);
#else
                uchar_t *p = (uchar_t *)in->cd_miscdata;
                des_ctx->dr_ctx.iv[0] = p[0]<<24 | p[1]<<16 | p[2]<<8 | p[3];
                des_ctx->dr_ctx.iv[1] = p[4]<<24 | p[5]<<16 | p[6]<<8 | p[7];
#endif  /* UNALIGNED_POINTERS_PERMITTED */
        }

        if (len > dca_length(out)) {
                DBG(dca, DWARN, "inadequate output space (need %d, got %d)",
                    len, dca_length(out));
                out->cd_length = len;
                /* Do not free the context since the app will call again */
                return (CRYPTO_BUFFER_TOO_SMALL);
        }

        if ((rv = dca_verifyio(in, out)) != CRYPTO_SUCCESS) {
                (void) dca_free_context(ctx);
                return (rv);
        }

        /* special handling for null-sized input buffers */
        if (len == 0) {
                out->cd_length = 0;
                (void) dca_free_context(ctx);
                return (CRYPTO_SUCCESS);
        }

        /*
         * Make a local copy of the input crypto_data_t structure. This
         * allows it to be manipulated locally and for dealing with in-place
         * data (ie in == out). Note that "nin" has been pre-allocated,
         * and only fields are copied, not actual data.
         */
        if ((rv = dca_dupcrypto(in, nin)) != CRYPTO_SUCCESS) {
                (void) dca_free_context(ctx);
                return (rv);
        }

        /* Set output to zero ready to take the processed data */
```

```
123         out->cd_length = 0;

125         reqp->dr_kcf_req = req;
126         reqp->dr_in = nin;
127         reqp->dr_out = out;
128         reqp->dr_job_stat = DS_3DESJOBS;
129         reqp->dr_byte_stat = DS_3DESBYTES;

131         rv = dca_3desstart(dca, flags, reqp);

133         /* Context will be freed in the kCF callback function otherwise */
134         if (rv != CRYPTO_QUEUED && rv != CRYPTO_BUFFER_TOO_SMALL) {
135                 (void) dca_free_context(ctx);
136         }
137         return (rv);
138 }
```

*unchanged_portion_omitted_*

```
161 int
162 dca_3desupdate(crypto_ctx_t *ctx, crypto_data_t *in,
163     crypto_data_t *out, crypto_req_handle_t req, int flags)
164 {
165         int                     len;
166         int                     rawlen;
167         int                     rv;
168         dca_request_t           *reqp = ctx->cc_provider_private;
169         dca_request_t           *des_ctx = ctx->cc_provider_private;
170         dca_t                   *dca = ctx->cc_provider;
171         crypto_data_t           *nin = &reqp->dr_ctx.in_dup;

173         rawlen = dca_length(in) + des_ctx->dr_ctx.residlen;

175         len = ROUNDDOWN(rawlen, DESBLOCK);
176         /*
177          * If cd_miscdata non-null then this contains the IV.
178          */
179         if (in->cd_miscdata != NULL) {
180 #ifdef UNALIGNED_POINTERS_PERMITTED
181                 uint32_t        *p = (uint32_t *)in->cd_miscdata;
182                 des_ctx->dr_ctx.iv[0] = htonl(p[0]);
183                 des_ctx->dr_ctx.iv[1] = htonl(p[1]);
184 #else
185                 uchar_t *p = (uchar_t *)in->cd_miscdata;
186                 des_ctx->dr_ctx.iv[0] = p[0]<<24 | p[1]<<16 | p[2]<<8 | p[3];
187                 des_ctx->dr_ctx.iv[1] = p[4]<<24 | p[5]<<16 | p[6]<<8 | p[7];
188 #endif  /* UNALIGNED_POINTERS_PERMITTED */
189         }

191         if (len > dca_length(out)) {
192                 DBG(dca, DWARN, "not enough output space (need %d, got %d)",
193                     len, dca_length(out));
194                 out->cd_length = len;
195                 /* Do not free the context since the app will call again */
196                 return (CRYPTO_BUFFER_TOO_SMALL);
197         }

199         if ((rv = dca_verifyio(in, out)) != CRYPTO_SUCCESS) {
200                 (void) dca_free_context(ctx);
201                 return (rv);
202         }

204         reqp->dr_kcf_req = req;

206         /*
207          * From here on out, we are committed.
208          */
```



```
210         if (len == 0) {
211                 /*
212                  * No blocks being encrypted, so we just accumulate the
213                  * input for the next pass and return.
214                  */
215                 if ((rv = dca_getbufbytes(in, 0,
216                     (rawlen % DESBLOCK) - des_ctx->dr_ctx.residlen,
217                     des_ctx->dr_ctx.resid + des_ctx->dr_ctx.residlen)) !=
218                     CRYPTO_SUCCESS) {
219                         DBG(dca, DWARN,
220                     "dca_3desupdate: dca_getbufbytes() failed for residual only pass");
221                         dca_freereq(reqp);
222                         return (rv);
223                 }
224                 des_ctx->dr_ctx.residlen = rawlen % DESBLOCK;

226                 out->cd_length = 0;
227                 /*
228                  * Do not free the context here since it will be done
229                  * in the final function
230                  */
231                 return (CRYPTO_SUCCESS);
232         }

234         /*
235          * Set up rbuf for previous residual data.
236          */
237         if (des_ctx->dr_ctx.residlen) {
238                 bcopy(des_ctx->dr_ctx.resid, des_ctx->dr_ctx.activeresid,
239                     des_ctx->dr_ctx.residlen);
240                 des_ctx->dr_ctx.activeresidlen = des_ctx->dr_ctx.residlen;
241         }

243         /*
244          * Locate and save residual data for next encrypt_update.
245          */
246         if ((rv = dca_getbufbytes(in, len - des_ctx->dr_ctx.residlen,
247             rawlen % DESBLOCK, des_ctx->dr_ctx.resid)) != CRYPTO_SUCCESS) {
248                 DBG(dca, DWARN, "dca_3desupdate: dca_getbufbytes() failed");
249                 (void) dca_free_context(ctx);
250                 return (rv);
251         }

253         /* Calculate new residual length. */
254         des_ctx->dr_ctx.residlen = rawlen % DESBLOCK;

256         /*
257          * Make a local copy of the input crypto_data_t structure. This
258          * allows it to be manipulated locally and for dealing with in-place
259          * data (ie in == out).
260          */
261         if ((rv = dca_dupcrypto(in, nin)) != CRYPTO_SUCCESS) {
262                 (void) dca_free_context(ctx);
263                 return (rv);
264         }

266         /* Set output to zero ready to take the processed data */
267         out->cd_length = 0;

269         reqp->dr_in = nin;
270         reqp->dr_out = out;
271         reqp->dr_job_stat = DS_3DESJOBS;
272         reqp->dr_byte_stat = DS_3DESBYTES;

274         rv = dca_3desstart(dca, flags, reqp);
```

```
276         /*
277          * As this is multi-part the context is cleared on success
278          * (CRYPTO_QUEUED) in dca_3desfinal().
279          */

281         if (rv != CRYPTO_QUEUED && rv != CRYPTO_BUFFER_TOO_SMALL) {
282                 (void) dca_free_context(ctx);
283         }
284         return (rv);
285 }
______unchanged_portion_omitted_

377 int
378 dca_3desstart(dca_t *dca, uint32_t flags, dca_request_t *reqp)
379 {
380         size_t          len;
381         crypto_data_t   *in = reqp->dr_in;
382         int             rv;
383         dca_request_t   *ctx = reqp;
384         uint32_t        iv[2];

386         /*
387          * Preconditions:
388          * 1) in and out point to the "right" buffers.
389          * 2) in->b_bcount - in->b_resid == initial offset
390          * 3) likewise for out
391          * 4) there is enough space in the output
392          * 5) we perform a block for block encrypt
393          */
394         len = ctx->dr_ctx.activeresidlen + dca_length(in);
395         len = ROUNDDOWN(min(len, MAXPACKET), DESBLOCK);
396         reqp->dr_pkt_length = (uint16_t)len;

398         /* collect IVs for this pass */
399         iv[0] = ctx->dr_ctx.iv[0];
400         iv[1] = ctx->dr_ctx.iv[1];

402         /*
403          * And also, for decrypt, collect the IV for the next pass.  For
404          * decrypt, the IV must be collected BEFORE decryption, or else
405          * we will lose it.  (For encrypt, we grab the IV AFTER encryption,
406          * in dca_3desdone.
407          */
408         if (flags & DR_DECRYPT) {
409                 uchar_t         ivstore[DESBLOCK];
410 #ifdef UNALIGNED_POINTERS_PERMITTED
411                 uint32_t        *ivp = (uint32_t *)ivstore;
412 #else
413                 uchar_t         *ivp = ivstore;
414 #endif  /* UNALIGNED_POINTERS_PERMITTED */

416                 /* get last 8 bytes of ciphertext for IV of next op */
417                 /*
418                  * If we're processing only a DESBLOCKS worth of data
419                  * and there is active residual present then it will be
420                  * needed for the IV also.
421                  */
422                 if ((len == DESBLOCK) && ctx->dr_ctx.activeresidlen) {
423                         /* Bring the active residual into play */
424                         bcopy(ctx->dr_ctx.activeresid, ivstore,
425                             ctx->dr_ctx.activeresidlen);
426                         rv = dca_getbufbytes(in, 0,
427                             DESBLOCK - ctx->dr_ctx.activeresidlen,
428                             ivstore + ctx->dr_ctx.activeresidlen);
429                 } else {
```



```
430                         rv = dca_getbufbytes(in,
431                             len - DESBLOCK - ctx->dr_ctx.activeresidlen,
432                             DESBLOCK, ivstore);
433                 }

435                 if (rv != CRYPTO_SUCCESS) {
436                         DBG(dca, DWARN,
437                             "dca_3desstart: dca_getbufbytes() failed");
438                         return (rv);
439                 }

441                 /* store as a pair of native 32-bit values */
442 #ifdef UNALIGNED_POINTERS_PERMITTED
443                 ctx->dr_ctx.iv[0] = htonl(ivp[0]);
444                 ctx->dr_ctx.iv[1] = htonl(ivp[1]);
445 #else
446                 ctx->dr_ctx.iv[0] =
447                     ivp[0]<<24 | ivp[1]<<16 | ivp[2]<<8 | ivp[3];
448                 ctx->dr_ctx.iv[1] =
449                     ivp[4]<<24 | ivp[5]<<16 | ivp[6]<<8 | ivp[7];
450 #endif  /* UNALIGNED_POINTERS_PERMITTED */
451         }

453         /* For now we force a pullup.  Add direct DMA later. */
454         reqp->dr_flags &= ~(DR_SCATTER | DR_GATHER);
455         if ((len < dca_mindma) || (ctx->dr_ctx.activeresidlen > 0) ||
456             dca_sgcheck(dca, reqp->dr_in, DCA_SG_CONTIG) ||
457             dca_sgcheck(dca, reqp->dr_out, DCA_SG_WALIGN)) {
458                 reqp->dr_flags |= DR_SCATTER | DR_GATHER;
459         }

461         /* Try to do direct DMA. */
462         if (!(reqp->dr_flags & (DR_SCATTER | DR_GATHER))) {
463                 if (dca_bindchains(reqp, len, len) == DDI_SUCCESS) {
464                         reqp->dr_in->cd_offset += len;
465                         reqp->dr_in->cd_length -= len;
466                 } else {
467                         DBG(dca, DWARN,
468                             "dca_3desstart: dca_bindchains() failed");
469                         return (CRYPTO_DEVICE_ERROR);
470                 }
471         }

473         /* gather the data into the device */
474         if (reqp->dr_flags & DR_GATHER) {
475                 rv = dca_resid_gather(in, (char *)ctx->dr_ctx.activeresid,
476                     &ctx->dr_ctx.activeresidlen, reqp->dr_ibuf_kaddr, len);
477                 if (rv != CRYPTO_SUCCESS) {
478                         DBG(dca, DWARN,
479                             "dca_3desstart: dca_resid_gather() failed");
480                         return (rv);
481                 }
482                 /*
483                  * Setup for scattering the result back out
484                  * The output buffer is a multi-entry chain for x86 and
485                  * a single entry chain for Sparc.
486                  * Use the actual length if the first entry is sufficient.
487                  */
488                 (void) ddi_dma_sync(reqp->dr_ibuf_dmah, 0, len,
489                     DDI_DMA_SYNC_FORDEV);
490                 if (dca_check_dma_handle(dca, reqp->dr_ibuf_dmah,
491                     DCA_FM_ECLASS_NONE) != DDI_SUCCESS) {
492                         reqp->destroy = TRUE;
493                         return (CRYPTO_DEVICE_ERROR);
494                 }
```

```
496                 reqp->dr_in_paddr = reqp->dr_ibuf_head.dc_buffer_paddr;
497                 reqp->dr_in_next = reqp->dr_ibuf_head.dc_next_paddr;
498                 if (len > reqp->dr_ibuf_head.dc_buffer_length)
499                         reqp->dr_in_len = reqp->dr_ibuf_head.dc_buffer_length;
500                 else
501                         reqp->dr_in_len = len;
502         }
503         /*
504          * Setup for scattering the result back out
505          * The output buffer is a multi-entry chain for x86 and
506          * a single entry chain for Sparc.
507          * Use the actual length if the first entry is sufficient.
508          */
509         if (reqp->dr_flags & DR_SCATTER) {
510                 reqp->dr_out_paddr = reqp->dr_obuf_head.dc_buffer_paddr;
511                 reqp->dr_out_next = reqp->dr_obuf_head.dc_next_paddr;
512                 if (len > reqp->dr_obuf_head.dc_buffer_length)
513                         reqp->dr_out_len = reqp->dr_obuf_head.dc_buffer_length;
514                 else
515                         reqp->dr_out_len = len;
516         }

518         reqp->dr_flags |= flags;
519         reqp->dr_callback = dca_3desdone;

521         /* write out the context structure */
522         PUTCTX32(reqp, CTX_3DESIVHI, iv[0]);
523         PUTCTX32(reqp, CTX_3DESIVLO, iv[1]);

525         /* schedule the work by doing a submit */
526         return (dca_start(dca, reqp, MCR1, 1));
527 }

529 void
530 dca_3desdone(dca_request_t *reqp, int errno)
531 {
532         crypto_data_t   *out = reqp->dr_out;
533         dca_request_t   *ctx = reqp;
534         ASSERT(ctx != NULL);

536         if (errno == CRYPTO_SUCCESS) {
537                 size_t          off;
538                 /*
539                  * Save the offset: this has to be done *before* dca_scatter
540                  * modifies the buffer.  We take the initial offset into the
541                  * first buf, and add that to the total packet size to find
542                  * the end of the packet.
543                  */
544                 off = dca_length(out) + reqp->dr_pkt_length - DESBLOCK;

546                 if (reqp->dr_flags & DR_SCATTER) {
547                         (void) ddi_dma_sync(reqp->dr_obuf_dmah, 0,
548                             reqp->dr_out_len, DDI_DMA_SYNC_FORKERNEL);
549                         if (dca_check_dma_handle(reqp->dr_dca,
550                             reqp->dr_obuf_dmah, DCA_FM_ECLASS_NONE) !=
551                             DDI_SUCCESS) {
552                                 reqp->destroy = TRUE;
553                                 errno = CRYPTO_DEVICE_ERROR;
554                                 goto errout;
555                         }

557                         errno = dca_scatter(reqp->dr_obuf_kaddr,
558                             reqp->dr_out, reqp->dr_out_len, 0);
559                         if (errno != CRYPTO_SUCCESS) {
560                                 DBG(NULL, DWARN,
561                                     "dca_3desdone: dca_scatter() failed");
```



```
562                                 goto errout;
563                         }

565                 } else {
566                         /* we've processed some more data */
567                         out->cd_length += reqp->dr_pkt_length;
568                 }


571                 /*
572                  * For encryption only, we have to grab the IV for the
573                  * next pass AFTER encryption.
574                  */
575                 if (reqp->dr_flags & DR_ENCRYPT) {
576                         uchar_t         ivstore[DESBLOCK];
577 #ifdef UNALIGNED_POINTERS_PERMITTED
578                         uint32_t        *iv = (uint32_t *)ivstore;
579 #else
580                         uchar_t         *iv = ivstore;
581 #endif  /* UNALIGNED_POINTERS_PERMITTED */

583                         /* get last 8 bytes for IV of next op */
584                         errno = dca_getbufbytes(out, off, DESBLOCK,
585                             (uchar_t *)iv);
586                         errno = dca_getbufbytes(out, off, DESBLOCK, iv);
587                         if (errno != CRYPTO_SUCCESS) {
588                                 DBG(NULL, DWARN,
589                                     "dca_3desdone: dca_getbufbytes() failed");
590                                 goto errout;
591                         }

592                         /* store as a pair of native 32-bit values */
593 #ifdef UNALIGNED_POINTERS_PERMITTED
594                         ctx->dr_ctx.iv[0] = htonl(iv[0]);
595                         ctx->dr_ctx.iv[1] = htonl(iv[1]);
596 #else
597                         ctx->dr_ctx.iv[0] =
598                             iv[0]<<24 | iv[1]<<16 | iv[2]<<8 | iv[3];
599                         ctx->dr_ctx.iv[1] =
600                             iv[4]<<24 | iv[5]<<16 | iv[6]<<8 | iv[7];
601 #endif  /* UNALIGNED_POINTERS_PERMITTED */
602                 }

604                 /*
605                  * If there is more to do, then reschedule another
606                  * pass.
607                  */
608                 if (dca_length(reqp->dr_in) >= 8) {
609                         errno = dca_3desstart(reqp->dr_dca, reqp->dr_flags,
610                             reqp);
611                         if (errno == CRYPTO_QUEUED) {
612                                 return;
613                         }
614                 }
615         }

617 errout:

619         /*
620          * If this is an atomic operation perform the final function
621          * tasks (equivalent to to dca_3desfinal()).
622          */
623         if (reqp->dr_ctx.atomic) {
624                 if ((errno == CRYPTO_SUCCESS) && (ctx->dr_ctx.residlen != 0)) {
625                         DBG(NULL, DWARN,
626                             "dca_3desdone: invalid nonzero residual");
```

```
627                         if (reqp->dr_flags & DR_DECRYPT) {
628                                 errno = CRYPTO_ENCRYPTED_DATA_LEN_RANGE;
629                         } else {
630                                 errno = CRYPTO_DATA_LEN_RANGE;
631                         }
632                 }
633         }

635         ASSERT(reqp->dr_kcf_req != NULL);
636         /* notify framework that request is completed */
637         crypto_op_notification(reqp->dr_kcf_req, errno);
638         DBG(NULL, DINTR,
639             "dca_3desdone: returning %d to the kef via crypto_op_notification",
640             errno);

642         /* This has to be done after notifing the framework */
643         if (reqp->dr_ctx.atomic) {
644                 reqp->dr_context = NULL;
645                 reqp->dr_ctx.atomic = 0;
646                 reqp->dr_ctx.ctx_cm_type = 0;
647                 if (reqp->destroy)
648                         dca_destroyreq(reqp);
649                 else
650                         dca_freereq(reqp);
651         }
652 }

654 /* ARGSUSED */
655 int
656 dca_3desctxinit(crypto_ctx_t *ctx, crypto_mechanism_t *mechanism,
657     crypto_key_t *key, int kmflag, int flags)
658 {
659         dca_request_t   *des_ctx;
660         dca_t           *dca = ctx->cc_provider;
661 #ifdef UNALIGNED_POINTERS_PERMITTED
662         uint32_t        *param;
663         uint32_t        *value32;
664 #else
665         uchar_t         *param;
666 #endif  /* UNALIGNED_POINTERS_PERMITTED */
667         uchar_t         *value;
668         size_t          paramsz;
669         unsigned        len;
670         int             i, j;

672         paramsz = mechanism->cm_param_len;
673 #ifdef UNALIGNED_POINTERS_PERMITTED
674         param = (uint32_t *)mechanism->cm_param;
675 #else
676         param = (uchar_t *)mechanism->cm_param;
677 #endif  /* UNALIGNED_POINTERS_PERMITTED */

679         if ((paramsz != 0) && (paramsz != DES_IV_LEN)) {
680                 DBG(NULL, DWARN,
681                     "dca_3desctxinit: parameter(IV) length not %d (%d)",
682                     DES_IV_LEN, paramsz);
683                 return (CRYPTO_MECHANISM_PARAM_INVALID);
684         }

686         if ((des_ctx = dca_getreq(dca, MCR1, 1)) == NULL) {
687                 dca_error(dca, "unable to allocate request for 3DES");
688                 return (CRYPTO_HOST_MEMORY);
689         }
690         /*
691          * Identify and store the IV as a pair of native 32-bit words.
692          *
```



```
693          * If cm_param == NULL then the IV comes from the cd_miscdata field
694          * in the crypto_data structure.
695          */
696         if (param != NULL) {
697                 ASSERT(paramsz == DES_IV_LEN);
698 #ifdef UNALIGNED_POINTERS_PERMITTED
699                 des_ctx->dr_ctx.iv[0] = htonl(param[0]);
700                 des_ctx->dr_ctx.iv[1] = htonl(param[1]);
701 #else
702                 des_ctx->dr_ctx.iv[0] = param[0]<<24 | param[1]<<16 |
703                     param[2]<<8 | param[3];
704                 des_ctx->dr_ctx.iv[1] = param[4]<<24 | param[5]<<16 |
705                     param[6]<<8 | param[7];
706 #endif  /* UNALIGNED_POINTERS_PERMITTED */
707         }
708         des_ctx->dr_ctx.residlen = 0;
709         des_ctx->dr_ctx.activeresidlen = 0;
710         des_ctx->dr_ctx.ctx_cm_type = mechanism->cm_type;
711         ctx->cc_provider_private = des_ctx;

713         if (key->ck_format != CRYPTO_KEY_RAW) {
714                 DBG(NULL, DWARN,
715         "dca_3desctxinit: only raw crypto key type support with DES/3DES");
716                 dca_3desctxfree(ctx);
717                 return (CRYPTO_KEY_TYPE_INCONSISTENT);
718         }

720         len = key->ck_length;
721         value = (uchar_t *)key->ck_data;

723         if (flags & DR_TRIPLE) {
724                 /* 3DES */
725                 switch (len) {
726                 case 192:
727                         for (i = 0; i < 6; i++) {
728                                 des_ctx->dr_ctx.key[i] = 0;
729                                 for (j = 0; j < 4; j++) {
730                                         des_ctx->dr_ctx.key[i] <<= 8;
731                                         des_ctx->dr_ctx.key[i] |= *value;
732                                         value++;
733                                 }
734                         }
735                         break;

737                 case 128:
738                         for (i = 0; i < 4; i++) {
739                                 des_ctx->dr_ctx.key[i] = 0;
740                                 for (j = 0; j < 4; j++) {
741                                         des_ctx->dr_ctx.key[i] <<= 8;
742                                         des_ctx->dr_ctx.key[i] |= *value;
743                                         value++;
744                                 }
745                         }
746                         des_ctx->dr_ctx.key[4] = des_ctx->dr_ctx.key[0];
747                         des_ctx->dr_ctx.key[5] = des_ctx->dr_ctx.key[1];
748                         break;

750                 default:
751                         DBG(NULL, DWARN, "Incorrect 3DES keysize (%d)", len);
752                         dca_3desctxfree(ctx);
753                         return (CRYPTO_KEY_SIZE_RANGE);
754                 }
755         } else {
756                 /* single DES */
757                 if (len != 64) {
758                         DBG(NULL, DWARN, "Incorrect DES keysize (%d)", len);
```

```
                        dca_3desctxfree(ctx);
                        return (CRYPTO_KEY_SIZE_RANGE);
                }

#ifdef UNALIGNED_POINTERS_PERMITTED
                value32 = (uint32_t *)value;
                des_ctx->dr_ctx.key[0] = htonl(value32[0]);
                des_ctx->dr_ctx.key[1] = htonl(value32[1]);
#else
                des_ctx->dr_ctx.key[0] =
                    value[0]<<24 | value[1]<<16 | value[2]<<8 | value[3];
                des_ctx->dr_ctx.key[1] =
                    value[4]<<24 | value[5]<<16 | value[6]<<8 | value[7];
#endif  /* UNALIGNED_POINTERS_PERMITTED */

                /* for single des just repeat des key */
                des_ctx->dr_ctx.key[4] =
                    des_ctx->dr_ctx.key[2] = des_ctx->dr_ctx.key[0];
                des_ctx->dr_ctx.key[5] =
                    des_ctx->dr_ctx.key[3] = des_ctx->dr_ctx.key[1];
        }

        /*
         * Setup the context here so that we do not need to setup it up
         * for every update
         */
        PUTCTX16(des_ctx, CTX_LENGTH, CTX_3DES_LENGTH);
        PUTCTX16(des_ctx, CTX_CMD, CMD_3DES);
        PUTCTX32(des_ctx, CTX_3DESDIRECTION,
            flags & DR_ENCRYPT ? CTX_3DES_ENCRYPT : CTX_3DES_DECRYPT);
        PUTCTX32(des_ctx, CTX_3DESKEY1HI, des_ctx->dr_ctx.key[0]);
        PUTCTX32(des_ctx, CTX_3DESKEY1LO, des_ctx->dr_ctx.key[1]);
        PUTCTX32(des_ctx, CTX_3DESKEY2HI, des_ctx->dr_ctx.key[2]);
        PUTCTX32(des_ctx, CTX_3DESKEY2LO, des_ctx->dr_ctx.key[3]);
        PUTCTX32(des_ctx, CTX_3DESKEY3HI, des_ctx->dr_ctx.key[4]);
        PUTCTX32(des_ctx, CTX_3DESKEY3LO, des_ctx->dr_ctx.key[5]);

        return (CRYPTO_SUCCESS);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    5361 Wed Aug 27 14:23:48 2008
new/usr/src/uts/common/sys/byteorder.h
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1983, 1984, 1985, 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.
 */

#ifndef _SYS_BYTEORDER_H
#define _SYS_BYTEORDER_H

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

#include <sys/isa_defs.h>
#include <sys/int_types.h>

#if defined(__GNUC__) && defined(_ASM_INLINES) && \
        (defined(__i386) || defined(__amd64))
#include <asm/byteorder.h>
#endif

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * macros for conversion between host and (internet) network byte order
```



```
 */

#if defined(_BIG_ENDIAN) && !defined(ntohl) && !defined(__lint)
/* big-endian */
#define ntohl(x)        (x)
#define ntohll(x)       (x)
#define ntohs(x)        (x)
#define htonl(x)        (x)
#define htonll(x)       (x)
#define htons(x)        (x)

#elif !defined(ntohl) /* little-endian */

#ifndef _IN_PORT_T
#define _IN_PORT_T
typedef uint16_t in_port_t;
#endif

#ifndef _IN_ADDR_T
#define _IN_ADDR_T
typedef uint32_t in_addr_t;
#endif

#if !defined(_XPG4_2) || defined(__EXTENSIONS__) || defined(_XPG5)
extern  uint32_t htonl(uint32_t);
extern  uint16_t htons(uint16_t);
extern  uint32_t ntohl(uint32_t);
extern  uint16_t ntohs(uint16_t);
#else
extern  in_addr_t htonl(in_addr_t);
extern  in_port_t htons(in_port_t);
extern  in_addr_t ntohl(in_addr_t);
extern  in_port_t ntohs(in_port_t);
#endif  /* !defined(_XPG4_2) || defined(__EXTENSIONS__) || defined(_XPG5) */
#if !(defined(_XPG4_2) || defined(_XPG5)) || defined(__EXTENSIONS__)
extern  uint64_t htonll(uint64_t);
extern  uint64_t ntohll(uint64_t);
#endif  /* !(_XPG4_2||_XPG5) || __EXTENSIONS__ */
#endif

#if !defined(_XPG4_2) || defined(__EXTENSIONS__)

/*
 * Macros to reverse byte order
 */
#define BSWAP_8(x)      ((x) & 0xff)
#define BSWAP_16(x)     ((BSWAP_8(x) << 8) | BSWAP_8((x) >> 8))
#define BSWAP_32(x)     ((BSWAP_16(x) << 16) | BSWAP_16((x) >> 16))
#define BSWAP_64(x)     ((BSWAP_32(x) << 32) | BSWAP_32((x) >> 32))

#define BMASK_8(x)      ((x) & 0xff)
#define BMASK_16(x)     ((x) & 0xffff)
#define BMASK_32(x)     ((x) & 0xffffffff)
#define BMASK_64(x)     (x)

/*
 * Macros to convert from a specific byte order to/from native byte order
 */
#ifdef _BIG_ENDIAN
#define BE_8(x)         BMASK_8(x)
#define BE_16(x)        BMASK_16(x)
#define BE_32(x)        BMASK_32(x)
#define BE_64(x)        BMASK_64(x)
#define LE_8(x)         BSWAP_8(x)
#define LE_16(x)        BSWAP_16(x)
#define LE_32(x)        BSWAP_32(x)
```

```
123 #define LE_64(x)        BSWAP_64(x)
124 #else
125 #define LE_8(x)         BMASK_8(x)
126 #define LE_16(x)        BMASK_16(x)
127 #define LE_32(x)        BMASK_32(x)
128 #define LE_64(x)        BMASK_64(x)
129 #define BE_8(x)         BSWAP_8(x)
130 #define BE_16(x)        BSWAP_16(x)
131 #define BE_32(x)        BSWAP_32(x)
132 #define BE_64(x)        BSWAP_64(x)
133 #endif

135 /*
136  * Macros to read unaligned values from a specific byte order to
137  * native byte order
138  */

140 #define BE_IN8(xa) \
141         *((uint8_t *)(xa))

143 #define BE_IN16(xa) \
144         (((uint16_t)BE_IN8(xa) << 8) | BE_IN8((uint8_t *)(xa)+1))

146 #define BE_IN32(xa) \
147         (((uint32_t)BE_IN16(xa) << 16) | BE_IN16((uint8_t *)(xa)+2))

149 #define BE_IN64(xa) \
150         (((uint64_t)BE_IN32(xa) << 32) | BE_IN32((uint8_t *)(xa)+4))

152 #define LE_IN8(xa) \
153         *((uint8_t *)(xa))

155 #define LE_IN16(xa) \
156         (((uint16_t)LE_IN8((uint8_t *)(xa) + 1) << 8) | LE_IN8(xa))

158 #define LE_IN32(xa) \
159         (((uint32_t)LE_IN16((uint8_t *)(xa) + 2) << 16) | LE_IN16(xa))

161 #define LE_IN64(xa) \
162         (((uint64_t)LE_IN32((uint8_t *)(xa) + 4) << 32) | LE_IN32(xa))

164 /*
165  * Macros to write unaligned values from native byte order to a specific byte
166  * order.
167  */

169 #define BE_OUT8(xa, yv) *((uint8_t *)(xa)) = (uint8_t)(yv);

171 #define BE_OUT16(xa, yv) \
172         BE_OUT8((uint8_t *)(xa) + 1, yv); \
173         BE_OUT8((uint8_t *)(xa), (yv) >> 8);

175 #define BE_OUT32(xa, yv) \
176         BE_OUT16((uint8_t *)(xa) + 2, yv); \
177         BE_OUT16((uint8_t *)(xa), (yv) >> 16);

179 #define BE_OUT64(xa, yv) \
180         BE_OUT32((uint8_t *)(xa) + 4, yv); \
181         BE_OUT32((uint8_t *)(xa), (yv) >> 32);

183 #define LE_OUT8(xa, yv) *((uint8_t *)(xa)) = (uint8_t)(yv);

185 #define LE_OUT16(xa, yv) \
186         LE_OUT8((uint8_t *)(xa), yv); \
187         LE_OUT8((uint8_t *)(xa) + 1, (yv) >> 8);
```



```
189 #define LE_OUT32(xa, yv) \
190         LE_OUT16((uint8_t *)(xa), yv); \
191         LE_OUT16((uint8_t *)(xa) + 2, (yv) >> 16);

193 #define LE_OUT64(xa, yv) \
194         LE_OUT32((uint8_t *)(xa), yv); \
195         LE_OUT32((uint8_t *)(xa) + 4, (yv) >> 32);

197 #endif  /* !defined(_XPG4_2) || defined(__EXTENSIONS__) */

199 #ifdef  __cplusplus
200 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4104 Wed Aug 27 14:23:53 2008
new/usr/src/uts/intel/amd64/ml/amd64.il
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/
/ In-line functions for amd64 kernels.
/

/
/ return current thread pointer
/
/ NOTE: the "0x18" should be replaced by the computed value of the
/       offset of "cpu_thread" from the beginning of the struct cpu.
/       Including "assym.h" does not work, however, since that stuff
/       is PSM-specific and is only visible to the 'unix' build anyway.
/       Same with current cpu pointer, where "0xc" should be replaced
/       by the computed value of the offset of "cpu_self".
/       Ugh -- what a disaster.
/
        .inline threadp,0
        movq    %gs:0x18, %rax
        .end

/
/ return current cpu pointer
/
        .inline curcpup,0
        movq    %gs:0x10, %rax
        .end

/
/ return caller
/
        .inline caller,0
        movq    8(%rbp), %rax
```



```
        .end

/
/ convert ipl to spl.  This is the identity function for i86
/
        .inline ipltospl,0
        movq    %rdi, %rax
        .end

/
/ find the low order bit in a word
/
        .inline lowbit,4
        movq    $-1, %rax
        bsfq    %rdi, %rax
        incq    %rax
        .end

/
/ Networking byte order functions (too bad, Intel has the wrong byte order)
/

        .inline htonll,4
        movq    %rdi, %rax
        bswapq  %rax
        .end

        .inline ntohll,4
        movq    %rdi, %rax
        bswapq  %rax
        .end

        .inline htonl,4
        movl    %edi, %eax
        bswap   %eax
        .end

        .inline ntohl,4
        movl    %edi, %eax
        bswap   %eax
        .end

        .inline htons,4
        movl    %edi, %eax
        bswap   %eax
        shrl    $16, %eax
        .end

        .inline ntohs,4
        movl    %edi, %eax
        bswap   %eax
        shrl    $16, %eax
        .end

/*
 * multiply two long numbers and yield a u_lonlong_t result
 * Provided to manipulate hrtime_t values.
 */
        /* XX64 These don't work correctly with SOS9 build 13.0 yet
        .inline mul32, 8
        xorl    %edx, %edx
        movl    %edi, %eax
        mull    %esi
        shlq    $32, %rdx
        orq     %rdx, %rax
        ret
```

```
123         .end
124         */
125 /*
126  * Unlock hres_lock and increment the count value. (See clock.h)
127  */
128         .inline unlock_hres_lock, 0
129         lock
130         incl    hres_lock
131         .end

133         .inline atomic_orb,8
134         movl    %esi, %eax
135         lock
136         orb     %al,(%rdi)
137         .end

139         .inline atomic_andb,8
140         movl    %esi, %eax
141         lock
142         andb    %al,(%rdi)
143         .end

145 /*
146  * atomic inc/dec operations.
147  *      void atomic_inc16(uint16_t *addr) { ++*addr; }
148  *      void atomic_dec16(uint16_t *addr) { --*addr; }
149  */
150         .inline atomic_inc16,4
151         lock
152         incw    (%rdi)
153         .end

155         .inline atomic_dec16,4
156         lock
157         decw    (%rdi)
158         .end

160 /*
161  * atomic bit clear
162  */
163         .inline atomic_btr32,8
164         lock
165         btrl %esi, (%rdi)
166         setc %al
167         .end

169 /*
170  * Call the pause instruction.  To the Pentium 4 Xeon processor, it acts as
171  * a hint that the code sequence is a busy spin-wait loop.  Without a pause
172  * instruction in these loops, the P4 Xeon processor may suffer a severe
173  * penalty when exiting the loop because the processor detects a possible
174  * memory violation.  Inserting the pause instruction significantly reduces
175  * the likelihood of a memory order violation, improving performance.
176  * The pause instruction is a NOP on all other IA-32 processors.
177  */
178         .inline ht_pause, 0
179         pause
180         .end

182 /*
183  * inlines for update_sregs().
184  */
185         .inline __set_ds, 0
186         movw    %di, %ds
187         .end
```



```
189         .inline __set_es, 0
190         movw    %di, %es
191         .end

193         .inline __set_fs, 0
194         movw    %di, %fs
195         .end

197         .inline __set_gs, 0
198         movw    %di, %gs
199         .end

201         /*
202          * OPTERON_ERRATUM_88 requires mfence
203          */
204         .inline __swapgs, 0
205         mfence
206         swapgs
207         .end
```

```
**********************************************************
    2749 Wed Aug 27 14:23:58 2008
new/usr/src/uts/intel/asm/byteorder.h
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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.
 * Common Development and Distribution License, Version 1.0 only
 * (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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _ASM_BYTEORDER_H
#define _ASM_BYTEORDER_H

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

#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

#if !defined(__lint) && defined(__GNUC__)

/*
 * htonll(), ntohll(), htonl(), ntohl(), htons(), ntohs()
 * These functions reverse the byte order of the input parameter and returns
 * the result.  This is to convert the byte order from host byte order
 * (little endian) to network byte order (big endian), or vice versa.
 */
#if defined(__i386) || defined(__amd64)

extern __inline__ uint32_t htonl(uint32_t value)
{
        __asm__("bswap %0" : "+r" (value));
        return (value);
}

#if defined(__i386) || defined(__amd64)
extern __inline__ uint32_t ntohl(uint32_t value)
{
```



```
        __asm__("bswap %0" : "+r" (value));
        return (value);
}

extern __inline__ uint16_t htons(uint16_t value)
{
#if defined(__amd64)
        __asm__("xchgb %h0, %b0" : "+Q" (value));
#elif defined(__i386)
        __asm__("xchgb %h0, %b0" : "+q" (value));
#endif
        return (value);
}
______unchanged_portion_omitted_

extern __inline__ uint32_t htonl(uint32_t value)
{
        __asm__("bswap %0" : "+r" (value));
        return (value);
}

extern __inline__ uint32_t ntohl(uint32_t value)
{
        __asm__("bswap %0" : "+r" (value));
        return (value);
}

#if defined(__amd64)
extern __inline__ uint64_t htonll(uint64_t value)
{
        __asm__("bswapq %0" : "+r" (value));
        return (value);
}

extern __inline__ uint64_t ntohll(uint64_t value)
{
        __asm__("bswapq %0" : "+r" (value));
        return (value);
}

#elif defined(__i386)
/* Use the htonl() and ntohl() inline functions defined above */
extern __inline__ uint64_t htonll(uint64_t value)
{
        return (htonl(value >> 32) | ((uint64_t)htonl(value) << 32));
}

extern __inline__ uint64_t ntohll(uint64_t value)
{
        return (ntohl(value >> 32) | (uint64_t)ntohl(value) << 32);
}
#endif  /* __amd64 */

#endif  /* __i386 || __amd64 */

#endif  /* !__lint && __GNUC__ */

#ifdef  __cplusplus
}
______unchanged_portion_omitted_
```

```
**********************************************************
   78530 Wed Aug 27 14:24:03 2008
new/usr/src/uts/intel/ia32/ml/i86_subr.s
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
______unchanged_portion_omitted_

1072 #endif  /* __i386 */

1074 #ifdef DEBUG
1075         .text
1076 .str_panic_msg:
1077         .string "strlen: argument below kernelbase"
1078 #endif /* DEBUG */

1080 #endif  /* __lint */

1082         /*
1083          * Berkeley 4.3 introduced symbolically named interrupt levels
1083          * Berkley 4.3 introduced symbolically named interrupt levels
1084          * as a way deal with priority in a machine independent fashion.
1085          * Numbered priorities are machine specific, and should be
1086          * discouraged where possible.
1087          *
1088          * Note, for the machine specific priorities there are
1089          * examples listed for devices that use a particular priority.
1090          * It should not be construed that all devices of that
1091          * type should be at that priority.  It is currently were
1092          * the current devices fit into the priority scheme based
1093          * upon time criticalness.
1094          *
1095          * The underlying assumption of these assignments is that
1096          * IPL 10 is the highest level from which a device
1097          * routine can call wakeup.  Devices that interrupt from higher
1098          * levels are restricted in what they can do.  If they need
1099          * kernels services they should schedule a routine at a lower
1100          * level (via software interrupt) to do the required
1101          * processing.
1102          *
1103          * Examples of this higher usage:
1104          *      Level   Usage
1105          *      14      Profiling clock (and PROM uart polling clock)
1106          *      12      Serial ports
1107          *
1108          * The serial ports request lower level processing on level 6.
1109          *
1110          * Also, almost all splN routines (where N is a number or a
1111          * mnemonic) will do a RAISE(), on the assumption that they are
1112          * never used to lower our priority.
1113          * The exceptions are:
1114          *      spl8()          Because you can't be above 15 to begin with!
1115          *      splzs()         Because this is used at boot time to lower our
1116          *                      priority, to allow the PROM to poll the uart.
1117          *      spl0()          Used to lower priority to 0.
1118          */

1120 #if defined(__lint)

1122 int spl0(void)          { return (0); }
1123 int spl6(void)          { return (0); }
1124 int spl7(void)          { return (0); }
1125 int spl8(void)          { return (0); }
1126 int splhigh(void)       { return (0); }
1127 int splhi(void)         { return (0); }
1128 int splzs(void)         { return (0); }

1130 /* ARGSUSED */
1131 void
1132 splx(int level)
1133 {}
```



```
1135 #else   /* __lint */

1137 #if defined(__amd64)

1139 #define SETPRI(level) \
1140         movl    $/**/level, %edi;       /* new priority */              \
1141         jmp     do_splx                 /* redirect to do_splx */

1143 #define RAISE(level) \
1144         movl    $/**/level, %edi;       /* new priority */              \
1145         jmp     splr                    /* redirect to splr */

1147 #elif defined(__i386)

1149 #define SETPRI(level) \
1150         pushl   $/**/level;     /* new priority */                      \
1151         call    do_splx;        /* invoke common splx code */           \
1152         addl    $4, %esp;       /* unstack arg */                       \
1153         ret

1155 #define RAISE(level) \
1156         pushl   $/**/level;     /* new priority */                      \
1157         call    splr;           /* invoke common splr code */           \
1158         addl    $4, %esp;       /* unstack args */                      \
1159         ret

1161 #endif  /* __i386 */

1163         /* locks out all interrupts, including memory errors */
1164         ENTRY(spl8)
1165         SETPRI(15)
1166         SET_SIZE(spl8)
______unchanged_portion_omitted_


1196         /* splx implementation */
1196         /* splx implentation */
1197         ENTRY(splx)
1198         jmp     do_splx         /* redirect to common splx code */
1199         SET_SIZE(splx)
______unchanged_portion_omitted_

2027 #endif  /* __i386 */
2028 #endif  /* __lint */

2030 /* htonll(), ntohll(), htonl(), ntohl(), htons(), ntohs()
2031  * These functions reverse the byte order of the input parameter and returns
2032  * the result.  This is to convert the byte order from host byte order
2033  * (little endian) to network byte order (big endian), or vice versa.
2034  */

2036 #if defined(__lint)

2038 uint64_t
2039 htonll(uint64_t i)
2040 { return (i); }

2042 uint64_t
2043 ntohll(uint64_t i)
2044 { return (i); }

2032 /* ARGSUSED */
2046 uint32_t
2047 htonl(uint32_t i)
2048 { return (i); }
2035 { return (0); }
```

```
2037 /* ARGSUSED */
2050 uint32_t
2051 ntohl(uint32_t i)
2052 { return (i); }
2040 { return (0); }

2054 uint16_t
2055 htons(uint16_t i)
2056 { return (i); }

2058 uint16_t
2059 ntohs(uint16_t i)
2060 { return (i); }

2062 #else   /* __lint */

2064 #if defined(__amd64)

2066         ENTRY(htonll)
2067         ALTENTRY(ntohll)
2068         movq    %rdi, %rax
2069         bswapq  %rax
2070         ret
2071         SET_SIZE(ntohll)
2072         SET_SIZE(htonll)

2074         /* XX64 there must be shorter sequences for this */
2075         ENTRY(htonl)
2076         ALTENTRY(ntohl)
2077         movl    %edi, %eax
2078         bswap   %eax
2079         ret
2080         SET_SIZE(ntohl)
______unchanged_portion_omitted_

2055 #elif defined(__i386)

2057         ENTRY(htonl)
2058         ALTENTRY(ntohl)
2059         movl    4(%esp), %eax
2060         bswap   %eax
2061         ret
2062         SET_SIZE(ntohl)
2063         SET_SIZE(htonl)

2065 #endif  /* __i386 */
2066 #endif  /* __lint */

2068 #if defined(__lint)

2070 /* ARGSUSED */
2071 uint16_t
2072 htons(uint16_t i)
2073 { return (0); }

2075 /* ARGSUSED */
2076 uint16_t
2077 ntohs(uint16_t i)
2078 { return (0); }


2081 #else   /* __lint */

2083 #if defined(__amd64)
```



```
2083         /* XX64 there must be better sequences for this */
2084         ENTRY(htons)
2085         ALTENTRY(ntohs)
2086         movl    %edi, %eax
2087         bswap   %eax
2088         shrl    $16, %eax
2089         ret
2090         SET_SIZE(ntohs)
______unchanged_portion_omitted_

2093 #elif defined(__i386)

2095         ENTRY(htonll)
2096         ALTENTRY(ntohll)
2097         movl    4(%esp), %edx
2098         movl    8(%esp), %eax
2099         bswap   %edx
2100         bswap   %eax
2101         ret
2102         SET_SIZE(ntohll)
2103         SET_SIZE(htonll)

2105         ENTRY(htonl)
2106         ALTENTRY(ntohl)
2107         movl    4(%esp), %eax
2108         bswap   %eax
2109         ret
2110         SET_SIZE(ntohl)
2111         SET_SIZE(htonl)

2113         ENTRY(htons)
2114         ALTENTRY(ntohs)
2115         movl    4(%esp), %eax
2116         bswap   %eax
2117         shrl    $16, %eax
2118         ret
2119         SET_SIZE(ntohs)
______unchanged_portion_omitted_
```

```
**********************************************************
    3908 Wed Aug 27 14:24:08 2008
new/usr/src/uts/intel/ia32/ml/ia32.il
5007142 Add ntohll and htonll to sys/byteorder.h
6717509 Need to use bswap/bswapq for byte swap of 64-bit integer on x32/x64
PSARC/2008/474
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/
/ Inline functions for i386 kernels.
/       Shared between all x86 platform variants.
/

/
/ return current thread pointer
/
/ NOTE: the "0x10" should be replaced by the computed value of the
/       offset of "cpu_thread" from the beginning of the struct cpu.
/       Including "assym.h" does not work, however, since that stuff
/       is PSM-specific and is only visible to the 'unix' build anyway.
/       Same with current cpu pointer, where "0xc" should be replaced
/       by the computed value of the offset of "cpu_self".
/       Ugh -- what a disaster.
/
        .inline threadp,0
        movl    %gs:0x10, %eax
        .end

/
/ return current cpu pointer
/
        .inline curcpup,0
        movl    %gs:0xc, %eax
        .end

/
/ return caller
/
```



```
        .inline caller,0
        movl    4(%ebp), %eax
        .end

/
/ convert ipl to spl.  This is the identity function for i86
/
        .inline ipltospl,0
        movl    (%esp), %eax
        .end

/
/ find the low order bit in a word
/
        .inline lowbit,4
        movl    $-1, %eax
        bsfl    (%esp), %eax
        incl    %eax
        .end

/
/ find the high order bit in a word
/
        .inline highbit,4
        movl    $-1, %eax
        bsrl    (%esp), %eax
        incl    %eax
        .end

/
/ Networking byte order functions (too bad, Intel has the wrong byte order)
/
        .inline htonll,4
        movl    (%esp), %edx
        movl    4(%esp), %eax
        bswap   %edx
        bswap   %eax
        .end

        .inline ntohll,4
        movl    (%esp), %edx
        movl    4(%esp), %eax
        bswap   %edx
        bswap   %eax
        .end

        .inline htonl,4
        movl    (%esp), %eax
        bswap   %eax
        .end

        .inline ntohl,4
        movl    (%esp), %eax
        bswap   %eax
        .end

        .inline htons,4
        movl    (%esp), %eax
        bswap   %eax
        shrl    $16, %eax
        .end

        .inline ntohs,4
        movl    (%esp), %eax
        bswap   %eax
        shrl    $16, %eax
```

```
123         .end

125 /*
126  * multiply two long numbers and yield a u_longlong_t result
127  * Provided to manipulate hrtime_t values.
128  */
129         .inline mul32, 8
130         movl    4(%esp), %eax
131         movl    (%esp), %ecx
132         mull    %ecx
133         .end

135 /*
136  * Unlock hres_lock and increment the count value. (See clock.h)
137  */
138         .inline unlock_hres_lock, 0
139         lock
140         incl    hres_lock
141         .end

143         .inline atomic_orb,8
144         movl    (%esp), %eax
145         movl    4(%esp), %edx
146         lock
147         orb     %dl,(%eax)
148         .end

150         .inline atomic_andb,8
151         movl    (%esp), %eax
152         movl    4(%esp), %edx
153         lock
154         andb    %dl,(%eax)
155         .end

157 /*
158  * atomic inc/dec operations.
159  *      void atomic_inc16(uint16_t *addr) { ++*addr; }
160  *      void atomic_dec16(uint16_t *addr) { --*addr; }
161  */
162         .inline atomic_inc16,4
163         movl    (%esp), %eax
164         lock
165         incw    (%eax)
166         .end

168         .inline atomic_dec16,4
169         movl    (%esp), %eax
170         lock
171         decw    (%eax)
172         .end

174 /*
175  * Call the pause instruction.  To the Pentium 4 Xeon processor, it acts as
176  * a hint that the code sequence is a busy spin-wait loop.  Without a pause
177  * instruction in these loops, the P4 Xeon processor may suffer a severe
178  * penalty when exiting the loop because the processor detects a possible
179  * memory violation.  Inserting the pause instruction significantly reduces
180  * the likelihood of a memory order violation, improving performance.
181  * The pause instruction is a NOP on all other IA-32 processors.
182  */
183         .inline ht_pause, 0
184         rep                     / our compiler doesn't support "pause" yet,
185         nop                     / so we're using "F3 90" opcode directly
186         .end
```