```
**********************************************************
    1758 Mon Mar  3 13:14:56 2008
new/usr/src/common/crypto/sha1/amd64/THIRDPARTYLICENSE
6662791 Need a SHA1 implementation optimized for 64-bit x86
**********************************************************
```

```
*********************************************************
      31 Mon Mar  3 13:14:57 2008
new/usr/src/common/crypto/sha1/amd64/THIRDPARTYLICENSE.descript
6662791 Need a SHA1 implementation optimized for 64-bit x86
*********************************************************
   1 PORTIONS OF SHA1 FUNCTIONALITY
```

```
**********************************************************
    9216 Mon Mar  3 13:14:58 2008
new/usr/src/common/crypto/sha1/amd64/sha1-x86_64.pl
6662791 Need a SHA1 implementation optimized for 64-bit x86
**********************************************************
```

```
#!/usr/bin/env perl
#
# ====================================================================
# Written by Andy Polyakov <appro@fy.chalmers.se> for the OpenSSL
# project. The module is, however, dual licensed under OpenSSL and
# CRYPTOGAMS licenses depending on where you obtain it. For further
# details see http://www.openssl.org/~appro/cryptogams/.
# ====================================================================
#
# sha1_block procedure for x86_64.
#
# It was brought to my attention that on EM64T compiler-generated code
# was far behind 32-bit assembler implementation. This is unlike on
# Opteron where compiler-generated code was only 15% behind 32-bit
# assembler, which originally made it hard to motivate the effort.
# There was suggestion to mechanically translate 32-bit code, but I
# dismissed it, reasoning that x86_64 offers enough register bank
# capacity to fully utilize SHA-1 parallelism. Therefore this fresh
# implementation:-) However! While 64-bit code does performs better
# on Opteron, I failed to beat 32-bit assembler on EM64T core. Well,
# x86_64 does offer larger *addressable* bank, but out-of-order core
# reaches for even more registers through dynamic aliasing, and EM64T
# core must have managed to run-time optimize even 32-bit code just as
# good as 64-bit one. Performance improvement is summarized in the
# following table:
#
#               gcc 3.4         32-bit asm      cycles/byte
# Opteron       +45%            +20%            6.8
# Xeon P4       +65%            +0%             9.9
# Core2         +60%            +10%            7.0

#
# OpenSolaris OS modifications
#
# Sun elects to use this software under the BSD license.
#
# This source originates from OpenSSL file sha1-x86_64.pl at
# ftp://ftp.openssl.org/snapshot/openssl-0.9.8-stable-SNAP-20080131.tar.gz
# (presumably for future OpenSSL release 0.9.8h), with these changes:
#
# 1. Added perl "use strict" and declared variables.
#
# 2. Added OpenSolaris ENTRY_NP/SET_SIZE macros from
# /usr/include/sys/asm_linkage.h, .ident keywords, and lint(1B) guards.
#
# 3. Added perl function &lea_offset_eax_register_register() to handle
#       Solaris as(1) bug.
#
# 4. Removed x86_64-xlate.pl script (not needed for as(1) or gas(1) assemblers).
#

use strict;
my ($code, $ctx, $inp, $num, $xi, $t0, $t1, $i, @V, $A, $B, $C, $D, $E, $T);
my $output = shift;
open STDOUT,">$output";


sub lea_offset_eax_register_register
# Workaround for a Solaris "gas" assembler bug where compiling the source
# errors out and does not generate a valid "lea" instruction.  Specifically,
#       &lea OFFSET(%eax, SOURCE_REGISTER),DESTINATION_REGISTER
```



```
#
# For Solaris as, "as -a32" must be used to compile this.
# For Solaris gas 2.15, this errors out with this message:
# Error: `0x5a827999(%eax,%r11d)' is not a valid 64 bit base/index expression
#
# This should be fixed in Solaris gas 2.16.
# It assembles with the Linux "as --64" gas 2.17 assembler and runs OK.
#
# For the ONBLD NV tools, the aw wrapper script fails when -a32 is used:
# /ws/onnv-tools/onbld/bin/i386/aw -xarch=amd64 -P -a32 -o lea.o lea.s
# aw: as->gas mapping failed at or near arg '-a32'
#
# For more information, see CRs 6644870 and 6628627.
{
        use Switch;
        my ($offset, $reg_src, $reg_dest) = @_;

        # Failed "lea" instruction.
        # This instruction errors out from the Solaris as assembler.
        # It assembles with the Linux "as --64" assembler and runs OK.
        $code .= "      /lea    $offset(%eax,$reg_src),$reg_dest\n";

        # Workaround
        # This workaround hand-generates hex machine code for lea.
        $code .= "      / Solaris as assembly bug CR 6628627 errors out for\n";
        $code .= "      / the above, so we specify the machine code in hex:\n";
        $code .= "      .byte   0x67    / lea\n";

        switch ($reg_src) {
        case "%ebp"     {
                        switch ($reg_dest) {
                        case "%r11d" { $code .=
                                "       .byte   0x44,0x8d,0x9c,0x28     "
                                . "/ (%eax,$reg_src),$reg_dest\n"; }
                        else    { $code .= "Unknown register $reg_dest\n"; }
                        }
        }
        case "%edi"     {
                        switch ($reg_dest) {
                        case "%ebp" { $code .=
                                "       .byte   0x8d,0xac,0x38  "
                                . "/ (%eax,$reg_src),$reg_dest\n"; }
                        else    { $code .= "Unknown register $reg_dest\n"; }
                        }
        }
        case "%edx"     {
                        switch ($reg_dest) {
                        case "%esi" { $code .=
                                "       .byte   0x8d,0xb4,0x10  "
                                . "/ (%eax,$reg_src),$reg_dest\n"; }
                        else    { $code .= "Unknown register $reg_dest\n"; }
                        }
        }
        case "%esi"     {
                        switch ($reg_dest) {
                        case "%edi" { $code .=
                                "       .byte   0x8d,0xbc,0x30  "
                                . "/ (%eax,$reg_src),$reg_dest\n"; }
                        else    { $code .= "Unknown register $reg_dest\n"; }
                        }
        }
        case "%r11d"    {
                        switch ($reg_dest) {
                        case "%r12d" { $code .=
                                "       .byte   0x46,0x8d,0xa4,0x18     "
                                . "/ (%eax,$reg_src),$reg_dest\n"; }
```

```
128                         else    { $code .= "Unknown register $reg_dest\n"; }
129                         }
130         }
131         case "%r12d"    {
132                         switch ($reg_dest) {
133                         case "%edx" { $code .=
134                                 "       .byte   0x42,0x8d,0x94,0x20     "
135                                 . "/ (%eax,$reg_src),$reg_dest\n"; }
136                         else    { $code .= "Unknown register $reg_dest\n"; }
137                         }
138         }
139         else            { $code .= "Unknown register $reg_src\n"; }
140         }
141
142         $code .= "      .long   $offset / offset\n";
143 }


144 #
145 # void sha1_block_data_order(SHA1_CTX *ctx, const void *inpp, size_t blocks);
146 #

147 # Arguments:
148 $ctx="%rdi";    # 1st arg
149 $inp="%rsi";    # 2nd arg
150 $num="%rdx";    # 3rd arg

151 # reassign arguments in order to produce more compact code
152 $ctx="%r8";
153 $inp="%r9";
154 $num="%r10";

155 # Temporaries:
156 $xi="%eax";
157 $t0="%ebx";
158 $t1="%ecx";
159 # State information from SHA-1 context:
160 $A="%edx";
161 $B="%esi";
162 $C="%edi";
163 $D="%ebp";
164 $E="%r11d";
165 # Temporary:
166 $T="%r12d";

167 @V=($A,$B,$C,$D,$E,$T);

168 sub PROLOGUE {
169 my $func=shift;
170 $code.=<<___;
171 ENTRY_NP($func)
172         /* EXPORT DELETE START */
173         push    %rbx
174         push    %rbp
175         push    %r12
176         mov     %rsp,%rax
177         mov     %rdi,$ctx       # reassigned argument
178         sub     \$‘8+16*4‘,%rsp
179         mov     %rsi,$inp       # reassigned argument
180         and     \$-64,%rsp
181         mov     %rdx,$num       # reassigned argument
182         mov     %rax,‘16*4‘(%rsp)

183         mov     0($ctx),$A
184         mov     4($ctx),$B
185         mov     8($ctx),$C
```



```
186         mov     12($ctx),$D
187         mov     16($ctx),$E
188 ___
189 }

190 sub EPILOGUE {
191 my $func=shift;
192 $code.=<<___;
193         mov     ‘16*4‘(%rsp),%rsp
194         pop     %r12
195         pop     %rbp
196         pop     %rbx
197         /* EXPORT DELETE END */
198         ret
199 SET_SIZE($func)
200 ___
201 }

202 sub BODY_00_19 {
203 my ($i,$a,$b,$c,$d,$e,$f,$host)=@_;
204 my $j=$i+1;
205 $code.=<<___ if ($i==0);
206         mov     ‘4*$i‘($inp),$xi
207         ‘"bswap $xi"    if(!defined($host))‘
208         mov     $xi,‘4*$i‘(%rsp)
209 ___
210         &lea_offset_eax_register_register("0x5a827999", $e, $f) if ($i < 15);
211 $code.=<<___ if ($i<15);
212         /lea    0x5a827999($xi,$e),$f
213         mov     $c,$t0
214         mov     ‘4*$j‘($inp),$xi
215         mov     $a,$e
216         xor     $d,$t0
217         ‘"bswap $xi"    if(!defined($host))‘
218         rol     \$5,$e
219         and     $b,$t0
220         mov     $xi,‘4*$j‘(%rsp)
221         add     $e,$f
222         xor     $d,$t0
223         rol     \$30,$b
224         add     $t0,$f
225 ___
226         &lea_offset_eax_register_register("0x5a827999", $e, $f) if ($i >= 15);
227 $code.=<<___ if ($i>=15);
228         /lea    0x5a827999($xi,$e),$f
229         mov     ‘4*($j%16)‘(%rsp),$xi
230         mov     $c,$t0
231         mov     $a,$e
232         xor     ‘4*(($j+2)%16)‘(%rsp),$xi
233         xor     $d,$t0
234         rol     \$5,$e
235         xor     ‘4*(($j+8)%16)‘(%rsp),$xi
236         and     $b,$t0
237         add     $e,$f
238         xor     ‘4*(($j+13)%16)‘(%rsp),$xi
239         xor     $d,$t0
240         rol     \$30,$b
241         add     $t0,$f
242         rol     \$1,$xi
243         mov     $xi,‘4*($j%16)‘(%rsp)
244 ___
245 }

246 sub BODY_20_39 {
247 my ($i,$a,$b,$c,$d,$e,$f)=@_;
248 my $j=$i+1;
```

```
my $K=($i<40)?0x6ed9eba1:0xca62c1d6;
        &lea_offset_eax_register_register($K, $e, $f) if ($i < 79);
$code.=<<___ if ($i<79);
        /lea    $K($xi,$e),$f
        mov     '4*($j%16)'(%rsp),$xi
        mov     $c,$t0
        mov     $a,$e
        xor     '4*(($j+2)%16)'(%rsp),$xi
        xor     $b,$t0
        rol     \$5,$e
        xor     '4*(($j+8)%16)'(%rsp),$xi
        xor     $d,$t0
        add     $e,$f
        xor     '4*(($j+13)%16)'(%rsp),$xi
        rol     \$30,$b
        add     $t0,$f
        rol     \$1,$xi
___
$code.=<<___ if ($i<76);
        mov     $xi,'4*($j%16)'(%rsp)
___
        &lea_offset_eax_register_register($K, $e, $f) if ($i == 79);
$code.=<<___ if ($i==79);
        /lea    $K($xi,$e),$f
        mov     $c,$t0
        mov     $a,$e
        xor     $b,$t0
        rol     \$5,$e
        xor     $d,$t0
        add     $e,$f
        rol     \$30,$b
        add     $t0,$f
___
}

sub BODY_40_59 {
my ($i,$a,$b,$c,$d,$e,$f)=@_;
my $j=$i+1;
        &lea_offset_eax_register_register("0x8f1bbcdc", $e, $f);
$code.=<<___;
        /lea    0x8f1bbcdc($xi,$e),$f
        mov     '4*($j%16)'(%rsp),$xi
        mov     $b,$t0
        mov     $b,$t1
        xor     '4*(($j+2)%16)'(%rsp),$xi
        mov     $a,$e
        and     $c,$t0
        xor     '4*(($j+8)%16)'(%rsp),$xi
        or      $c,$t1
        rol     \$5,$e
        xor     '4*(($j+13)%16)'(%rsp),$xi
        and     $d,$t1
        add     $e,$f
        rol     \$1,$xi
        or      $t1,$t0
        rol     \$30,$b
        mov     $xi,'4*($j%16)'(%rsp)
        add     $t0,$f
___
}

$code=<<___;
#if !defined(lint) && !defined(__lint)
        .ident  "@(#)sha1-x86_64.pl     1.1     08/03/02 SMI"
#include <sys/asm_linkage.h>
___
```



```


&PROLOGUE("sha1_block_data_order");
$code.=".align  4\n.Lloop:\n";
for($i=0;$i<20;$i++)    { &BODY_00_19($i,@V); unshift(@V,pop(@V)); }
for(;$i<40;$i++)        { &BODY_20_39($i,@V); unshift(@V,pop(@V)); }
for(;$i<60;$i++)        { &BODY_40_59($i,@V); unshift(@V,pop(@V)); }
for(;$i<80;$i++)        { &BODY_20_39($i,@V); unshift(@V,pop(@V)); }
$code.=<<___;
        / Update and save state information in SHA-1 context
        add     0($ctx),$E
        add     4($ctx),$T
        add     8($ctx),$A
        add     12($ctx),$B
        add     16($ctx),$C
        mov     $E,0($ctx)
        mov     $T,4($ctx)
        mov     $A,8($ctx)
        mov     $B,12($ctx)
        mov     $C,16($ctx)

        xchg    $E,$A   # mov   $E,$A
        xchg    $T,$B   # mov   $T,$B
        xchg    $E,$C   # mov   $A,$C
        xchg    $T,$D   # mov   $B,$D
                        # mov   $C,$E
        lea     '16*4'($inp),$inp
        sub     \$1,$num
        jnz     .Lloop
___
&EPILOGUE("sha1_block_data_order");
$code.=<<___;
.asciz  "SHA1 block transform for x86_64, CRYPTOGAMS by <appro\@openssl.org>"

#else
        /* LINTED */
        /* Nothing to be linted in this file--it's pure assembly source. */
#endif /* !lint && !__lint */
___

######################################################################

$code =~ s/\'([^\']*)\'/eval $1/gem;
print $code;
close STDOUT;
```

```
**********************************************************
   31673 Mon Mar  3 13:14:59 2008
new/usr/src/common/crypto/sha1/sha1.c
6662791 Need a SHA1 implementation optimized for 64-bit x86
**********************************************************
```

```
   1 /*
   2  * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
   2  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
   3  * Use is subject to license terms.
   4  */

   6 #pragma ident   "@(#)sha1.c     1.27    08/03/02 SMI"
   6 #pragma ident   "@(#)sha1.c     1.26    07/04/10 SMI"

   8 /*
   9  * The basic framework for this code came from the reference
  10  * implementation for MD5.  That implementation is Copyright (C)
  11  * 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
  12  *
  13  * License to copy and use this software is granted provided that it
  14  * is identified as the "RSA Data Security, Inc. MD5 Message-Digest
  15  * Algorithm" in all material mentioning or referencing this software
  16  * or this function.
  17  *
  18  * License is also granted to make and use derivative works provided
  19  * that such works are identified as "derived from the RSA Data
  20  * Security, Inc. MD5 Message-Digest Algorithm" in all material
  21  * mentioning or referencing the derived work.
  22  *
  23  * RSA Data Security, Inc. makes no representations concerning either
  24  * the merchantability of this software or the suitability of this
  25  * software for any particular purpose. It is provided "as is"
  26  * without express or implied warranty of any kind.
  27  *
  28  * These notices must be retained in any copies of any part of this
  29  * documentation and/or software.
  30  *
  31  * NOTE: Cleaned-up and optimized, version of SHA1, based on the FIPS 180-1
  32  * standard, available at http://www.itl.nist.gov/div897/pubs/fip180-1.htm
  33  * Not as fast as one would like -- further optimizations are encouraged
  34  * and appreciated.
  35  */

  37 #include <sys/types.h>
  38 #include <sys/param.h>
  39 #include <sys/systm.h>
  40 #include <sys/sysmacros.h>
  41 #include <sys/sha1.h>
  42 #include <sys/sha1_consts.h>

  44 #ifndef _KERNEL
  45 #include <strings.h>
  46 #include <stdlib.h>
  47 #include <errno.h>
  48 #include <sys/systeminfo.h>
  49 #endif  /* !_KERNEL */

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

  53 #if     defined(__sparc)

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

  59 static void SHA1Transform(uint32_t, uint32_t, uint32_t, uint32_t, uint32_t,
```



```
  60     SHA1_CTX *, const uint8_t *);

  62 #elif   defined(__amd64)

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

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

  70 #else

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

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

  76 #endif


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

  81 /*
  82  * F, G, and H are the basic SHA1 functions.
  83  */
  84 #define F(b, c, d)      (((b) & (c)) | ((~b) & (d)))
  85 #define G(b, c, d)      ((b) ^ (c) ^ (d))
  86 #define H(b, c, d)      (((b) & (c)) | (((b)|(c)) & (d)))

  88 /*
  89  * ROTATE_LEFT rotates x left n bits.
  90  */

  92 #if     defined(__GNUC__) && defined(_LP64)
  93 static __inline__ uint64_t
  94 ROTATE_LEFT(uint64_t value, uint32_t n)
  95 {
  96         uint32_t t32;

  98         t32 = (uint32_t)value;
  99         return ((t32 << n) | (t32 >> (32 - n)));
 100 }
______unchanged_portion_omitted_

 146 #ifdef VIS_SHA1
 147 #ifdef _KERNEL

 149 #include <sys/regset.h>
 150 #include <sys/vis.h>
 151 #include <sys/fpu/fpusystm.h>

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

 156 extern int sha1_savefp(kfpu_t *, int);
 157 extern void sha1_restorefp(kfpu_t *);

 159 uint32_t        vis_sha1_svfp_threshold = 128;

 161 #endif /* _KERNEL */

 163 /*
 164  * VIS SHA-1 consts.
 165  */
 166 static uint64_t VIS[] = {
 167         0x8000000080000000ULL,
 168         0x0002000200020002ULL,
```

```
169         0x5a8279996ed9eba1ULL,
170         0x8f1bbcdcca62c1d6ULL,
171         0x012389ab456789abULL};

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


176 /*
177  * SHA1Update()
178  *
179  * purpose: continues an sha1 digest operation, using the message block
180  *          to update the context.
181  *   input: SHA1_CTX *  : the context to update
182  *          void *      : the message block
183  *          size_t    : the length of the message block in bytes
184  *  output: void
185  */

187 void
188 SHA1Update(SHA1_CTX *ctx, const void *inptr, size_t input_len)
189 {
190         uint32_t i, buf_index, buf_len;
191         uint64_t X0[40], input64[8];
192         const uint8_t *input = inptr;
193 #ifdef _KERNEL
194         int usevis = 0;
195 #else
196         int usevis = 1;
197 #endif /* _KERNEL */

199         /* check for noop */
200         if (input_len == 0)
201                 return;

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

206         /* update number of bits */
207         if ((ctx->count[1] += (input_len << 3)) < (input_len << 3))
208                 ctx->count[0]++;

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

212         buf_len = 64 - buf_index;

214         /* transform as many times as possible */
215         i = 0;
216         if (input_len >= buf_len) {
217 #ifdef _KERNEL
218                 kfpu_t *fpu;
219                 if (fpu_exists) {
220                         uint8_t fpua[sizeof (kfpu_t) + GSR_SIZE + VIS_ALIGN];
221                         uint32_t len = (input_len + buf_index) & ~0x3f;
222                         int svfp_ok;

224                         fpu = (kfpu_t *)P2ROUNDUP((uintptr_t)fpua, 64);
225                         svfp_ok = ((len >= vis_sha1_svfp_threshold) ? 1 : 0);
226                         usevis = fpu_exists && sha1_savefp(fpu, svfp_ok);
227                 } else {
228                         usevis = 0;
229                 }
230 #endif /* _KERNEL */

232                 /*
233                  * general optimization:
234                  *
```



```
235                  * only do initial bcopy() and SHA1Transform() if
236                  * buf_index != 0.  if buf_index == 0, we're just
237                  * wasting our time doing the bcopy() since there
238                  * wasn't any data left over from a previous call to
239                  * SHA1Update().
240                  */

242                 if (buf_index) {
243                         bcopy(input, &ctx->buf_un.buf8[buf_index], buf_len);
244                         if (usevis) {
245                                 SHA1TransformVIS(X0,
246                                     ctx->buf_un.buf32,
247                                     &ctx->state[0], VIS);
248                         } else {
249                                 SHA1_TRANSFORM(ctx, ctx->buf_un.buf8);
250                         }
251                         i = buf_len;
252                 }

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

```
300                                             (uint32_t *)&input[i],
301                                             &ctx->state[0], VIS);
302                                 }

304                         }
305 #ifdef _KERNEL
306                         sha1_restorefp(fpu);
307 #endif /* _KERNEL */
308                 } else {
309                         for (; i + 63 < input_len; i += 64) {
310                                 SHA1_TRANSFORM(ctx, &input[i]);
311                         }
312                 }

314                 /*
315                  * general optimization:
316                  *
317                  * if i and input_len are the same, return now instead
318                  * of calling bcopy(), since the bcopy() in this case
319                  * will be an expensive nop.
320                  */

322                 if (input_len == i)
323                         return;

325                 buf_index = 0;
326         }

328         /* buffer remaining input */
329         bcopy(&input[i], &ctx->buf_un.buf8[buf_index], input_len - i);
330 }

332 #else /* VIS_SHA1 */

334 void
335 SHA1Update(SHA1_CTX *ctx, const void *inptr, size_t input_len)
336 {
337         uint32_t i, buf_index, buf_len;
338         const uint8_t *input = inptr;
339 #if defined(__amd64)
340         uint32_t        block_count;
341 #endif  /* __amd64 */

343         /* check for noop */
344         if (input_len == 0)
345                 return;

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

350         /* update number of bits */
351         if ((ctx->count[1] += (input_len << 3)) < (input_len << 3))
352                 ctx->count[0]++;

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

356         buf_len = 64 - buf_index;

358         /* transform as many times as possible */
359         i = 0;
360         if (input_len >= buf_len) {

362                 /*
363                  * general optimization:
364                  *
365                  * only do initial bcopy() and SHA1Transform() if
```



```
366                  * buf_index != 0.  if buf_index == 0, we're just
367                  * wasting our time doing the bcopy() since there
368                  * wasn't any data left over from a previous call to
369                  * SHA1Update().
370                  */

372                 if (buf_index) {
373                         bcopy(input, &ctx->buf_un.buf8[buf_index], buf_len);
374                         SHA1_TRANSFORM(ctx, ctx->buf_un.buf8);
375                         i = buf_len;
376                 }

378 #if !defined(__amd64)
379                 for (; i + 63 < input_len; i += 64)
380                         SHA1_TRANSFORM(ctx, &input[i]);
381 #else
382                 block_count = (input_len - i) >> 6;
383                 if (block_count > 0) {
384                         SHA1_TRANSFORM_BLOCKS(ctx, &input[i], block_count);
385                         i += block_count << 6;
386                 }
387 #endif  /* !__amd64 */

389                 /*
390                  * general optimization:
391                  *
392                  * if i and input_len are the same, return now instead
393                  * of calling bcopy(), since the bcopy() in this case
394                  * will be an expensive nop.
395                  */

397                 if (input_len == i)
398                         return;

400                 buf_index = 0;
401         }

403         /* buffer remaining input */
404         bcopy(&input[i], &ctx->buf_un.buf8[buf_index], input_len - i);
405 }

407 #endif /* VIS_SHA1 */

409 /*
410  * SHA1Final()
411  *
412  * purpose: ends an sha1 digest operation, finalizing the message digest and
413  *          zeroing the context.
414  *   input: uchar_t *   : A buffer to store the digest.
394  *   input: uchar_t *   : a buffer to store the digest in
415  *                      : The function actually uses void* because many
416  *                      : callers pass things other than uchar_t here.
417  *          SHA1_CTX *  : the context to finalize, save, and zero
418  *  output: void
419  */

421 void
422 SHA1Final(void *digest, SHA1_CTX *ctx)
423 {
424         uint8_t         bitcount_be[sizeof (ctx->count)];
425         uint32_t        index = (ctx->count[1] >> 3) & 0x3f;

427         /* store bit count, big endian */
428         Encode(bitcount_be, ctx->count, sizeof (bitcount_be));

430         /* pad out to 56 mod 64 */
```

```
431         SHA1Update(ctx, PADDING, ((index < 56) ? 56 : 120) - index);
432
433         /* append length (before padding) */
434         SHA1Update(ctx, bitcount_be, sizeof (bitcount_be));
435
436         /* store state in digest */
437         Encode(digest, ctx->state, sizeof (ctx->state));
438
439         /* zeroize sensitive information */
440         bzero(ctx, sizeof (*ctx));
441 }
442
443
444 #if !defined(__amd64)
445
446 typedef uint32_t sha1word;
447
448 /*
449  * sparc optimization:
450  *
451  * on the sparc, we can load big endian 32-bit data easily.  note that
452  * special care must be taken to ensure the address is 32-bit aligned.
453  * in the interest of speed, we don't check to make sure, since
454  * careful programming can guarantee this for us.
455  */
456
457 #if     defined(_BIG_ENDIAN)
458
459 #define LOAD_BIG_32(addr)       (*(uint32_t *)(addr))
460
461 #else   /* !defined(_BIG_ENDIAN) */
462
463 #if     defined(HAVE_BSWAP)
464
465 #define LOAD_BIG_32(addr) bswap(*((uint32_t *)(addr)))
466
467 #else   /* !defined(HAVE_BSWAP) */
468
469 /* little endian -- will work on big endian, but slowly */
470 #define LOAD_BIG_32(addr)       \
471         (((addr)[0] << 24) | ((addr)[1] << 16) | ((addr)[2] << 8) | (addr)[3])
472
473 #endif  /* !defined(HAVE_BSWAP) */
474
475 #endif  /* !defined(_BIG_ENDIAN) */
476
477 /*
478  * SHA1Transform()
479  */
480 #if     defined(W_ARRAY)
481 #define W(n) w[n]
482 #else   /* !defined(W_ARRAY) */
483 #define W(n) w_ ## n
484 #endif  /* !defined(W_ARRAY) */
485
486
487 #if     defined(__sparc)
488
489 /*
490  * sparc register window optimization:
491  *
492  * 'a', 'b', 'c', 'd', and 'e' are passed into SHA1Transform
493  * explicitly since it increases the number of registers available to
494  * the compiler.  under this scheme, these variables can be held in
495  * %i0 - %i4, which leaves more local and out registers available.
496  *
```



```
497  * purpose: sha1 transformation -- updates the digest based on 'block'
498  *   input: uint32_t    : bytes  1 -  4 of the digest
499  *          uint32_t    : bytes  5 -  8 of the digest
500  *          uint32_t    : bytes  9 - 12 of the digest
501  *          uint32_t    : bytes 12 - 16 of the digest
502  *          uint32_t    : bytes 16 - 20 of the digest
503  *          SHA1_CTX *  : the context to update
504  *          uint8_t [64]: the block to use to update the digest
505  *  output: void
506  */
507
508 void
509 SHA1Transform(uint32_t a, uint32_t b, uint32_t c, uint32_t d, uint32_t e,
510     SHA1_CTX *ctx, const uint8_t blk[64])
511 {
512         /*
513          * sparc optimization:
514          *
515          * while it is somewhat counter-intuitive, on sparc, it is
516          * more efficient to place all the constants used in this
517          * function in an array and load the values out of the array
518          * than to manually load the constants.  this is because
519          * setting a register to a 32-bit value takes two ops in most
520          * cases: a 'sethi' and an 'or', but loading a 32-bit value
521          * from memory only takes one 'ld' (or 'lduw' on v9).  while
522          * this increases memory usage, the compiler can find enough
523          * other things to do while waiting to keep the pipeline does
524          * not stall.  additionally, it is likely that many of these
525          * constants are cached so that later accesses do not even go
526          * out to the bus.
527          *
528          * this array is declared 'static' to keep the compiler from
529          * having to bcopy() this array onto the stack frame of
530          * SHA1Transform() each time it is called -- which is
531          * unacceptably expensive.
532          *
533          * the 'const' is to ensure that callers are good citizens and
534          * do not try to munge the array.  since these routines are
535          * going to be called from inside multithreaded kernelland,
536          * this is a good safety check. -- 'sha1_consts' will end up in
537          * .rodata.
538          *
539          * unfortunately, loading from an array in this manner hurts
540          * performance under intel.  so, there is a macro,
541          * SHA1_CONST(), used in SHA1Transform(), that either expands to
542          * a reference to this array, or to the actual constant,
543          * depending on what platform this code is compiled for.
544          */
545
546         static const uint32_t sha1_consts[] = {
547                 SHA1_CONST_0,   SHA1_CONST_1,   SHA1_CONST_2,   SHA1_CONST_3,
548         };
549
550         /*
551          * general optimization:
552          *
553          * use individual integers instead of using an array.  this is a
554          * win, although the amount it wins by seems to vary quite a bit.
555          */
556
557         uint32_t        w_0, w_1, w_2,  w_3,  w_4,  w_5,  w_6,  w_7;
558         uint32_t        w_8, w_9, w_10, w_11, w_12, w_13, w_14, w_15;
559
560         /*
561          * sparc optimization:
562          *
```

```
563          * if 'block' is already aligned on a 4-byte boundary, use
564          * LOAD_BIG_32() directly.  otherwise, bcopy() into a
565          * buffer that *is* aligned on a 4-byte boundary and then do
566          * the LOAD_BIG_32() on that buffer.  benchmarks have shown
567          * that using the bcopy() is better than loading the bytes
568          * individually and doing the endian-swap by hand.
569          *
570          * even though it's quite tempting to assign to do:
571          *
572          * blk = bcopy(ctx->buf_un.buf32, blk, sizeof (ctx->buf_un.buf32));
573          *
574          * and only have one set of LOAD_BIG_32()'s, the compiler
575          * *does not* like that, so please resist the urge.
576          */

578         if ((uintptr_t)blk & 0x3) {             /* not 4-byte aligned? */
579                 bcopy(blk, ctx->buf_un.buf32,  sizeof (ctx->buf_un.buf32));
580                 w_15 = LOAD_BIG_32(ctx->buf_un.buf32 + 15);
581                 w_14 = LOAD_BIG_32(ctx->buf_un.buf32 + 14);
582                 w_13 = LOAD_BIG_32(ctx->buf_un.buf32 + 13);
583                 w_12 = LOAD_BIG_32(ctx->buf_un.buf32 + 12);
584                 w_11 = LOAD_BIG_32(ctx->buf_un.buf32 + 11);
585                 w_10 = LOAD_BIG_32(ctx->buf_un.buf32 + 10);
586                 w_9  = LOAD_BIG_32(ctx->buf_un.buf32 +  9);
587                 w_8  = LOAD_BIG_32(ctx->buf_un.buf32 +  8);
588                 w_7  = LOAD_BIG_32(ctx->buf_un.buf32 +  7);
589                 w_6  = LOAD_BIG_32(ctx->buf_un.buf32 +  6);
590                 w_5  = LOAD_BIG_32(ctx->buf_un.buf32 +  5);
591                 w_4  = LOAD_BIG_32(ctx->buf_un.buf32 +  4);
592                 w_3  = LOAD_BIG_32(ctx->buf_un.buf32 +  3);
593                 w_2  = LOAD_BIG_32(ctx->buf_un.buf32 +  2);
594                 w_1  = LOAD_BIG_32(ctx->buf_un.buf32 +  1);
595                 w_0  = LOAD_BIG_32(ctx->buf_un.buf32 +  0);
596         } else {
597                 /*LINTED*/
598                 w_15 = LOAD_BIG_32(blk + 60);
599                 /*LINTED*/
600                 w_14 = LOAD_BIG_32(blk + 56);
601                 /*LINTED*/
602                 w_13 = LOAD_BIG_32(blk + 52);
603                 /*LINTED*/
604                 w_12 = LOAD_BIG_32(blk + 48);
605                 /*LINTED*/
606                 w_11 = LOAD_BIG_32(blk + 44);
607                 /*LINTED*/
608                 w_10 = LOAD_BIG_32(blk + 40);
609                 /*LINTED*/
610                 w_9  = LOAD_BIG_32(blk + 36);
611                 /*LINTED*/
612                 w_8  = LOAD_BIG_32(blk + 32);
613                 /*LINTED*/
614                 w_7  = LOAD_BIG_32(blk + 28);
615                 /*LINTED*/
616                 w_6  = LOAD_BIG_32(blk + 24);
617                 /*LINTED*/
618                 w_5  = LOAD_BIG_32(blk + 20);
619                 /*LINTED*/
620                 w_4  = LOAD_BIG_32(blk + 16);
621                 /*LINTED*/
622                 w_3  = LOAD_BIG_32(blk + 12);
623                 /*LINTED*/
624                 w_2  = LOAD_BIG_32(blk +  8);
625                 /*LINTED*/
626                 w_1  = LOAD_BIG_32(blk +  4);
627                 /*LINTED*/
628                 w_0  = LOAD_BIG_32(blk +  0);
```



```
629         }
630 #else   /* !defined(__sparc) */

632 void
633 SHA1Transform(SHA1_CTX *ctx, const uint8_t blk[64])
634 {
635         sha1word a = ctx->state[0];
636         sha1word b = ctx->state[1];
637         sha1word c = ctx->state[2];
638         sha1word d = ctx->state[3];
639         sha1word e = ctx->state[4];

641 #if     defined(W_ARRAY)
642         sha1word        w[16];
643 #else   /* !defined(W_ARRAY) */
644         sha1word        w_0, w_1, w_2,  w_3,  w_4,  w_5,  w_6,  w_7;
645         sha1word        w_8, w_9, w_10, w_11, w_12, w_13, w_14, w_15;
646 #endif  /* !defined(W_ARRAY) */

648         W(0)  = LOAD_BIG_32(blk +  0);
649         W(1)  = LOAD_BIG_32(blk +  4);
650         W(2)  = LOAD_BIG_32(blk +  8);
651         W(3)  = LOAD_BIG_32(blk + 12);
652         W(4)  = LOAD_BIG_32(blk + 16);
653         W(5)  = LOAD_BIG_32(blk + 20);
654         W(6)  = LOAD_BIG_32(blk + 24);
655         W(7)  = LOAD_BIG_32(blk + 28);
656         W(8)  = LOAD_BIG_32(blk + 32);
657         W(9)  = LOAD_BIG_32(blk + 36);
658         W(10) = LOAD_BIG_32(blk + 40);
659         W(11) = LOAD_BIG_32(blk + 44);
660         W(12) = LOAD_BIG_32(blk + 48);
661         W(13) = LOAD_BIG_32(blk + 52);
662         W(14) = LOAD_BIG_32(blk + 56);
663         W(15) = LOAD_BIG_32(blk + 60);

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

667         /*
668          * general optimization:
669          *
670          * even though this approach is described in the standard as
671          * being slower algorithmically, it is 30-40% faster than the
672          * "faster" version under SPARC, because this version has more
673          * of the constraints specified at compile-time and uses fewer
674          * variables (and therefore has better register utilization)
675          * than its "speedier" brother.  (i've tried both, trust me)
676          *
677          * for either method given in the spec, there is an "assignment"
678          * phase where the following takes place:
679          *
680          *      tmp = (main_computation);
681          *      e = d; d = c; c = rotate_left(b, 30); b = a; a = tmp;
682          *
683          * we can make the algorithm go faster by not doing this work,
684          * but just pretending that 'd' is now 'e', etc. this works
685          * really well and obviates the need for a temporary variable.
686          * however, we still explicitly perform the rotate action,
663          * however, we still explictly perform the rotate action,
687          * since it is cheaper on SPARC to do it once than to have to
688          * do it over and over again.
689          */

691         /* round 1 */
692         e = ROTATE_LEFT(a, 5) + F(b, c, d) + e + W(0) + SHA1_CONST(0); /* 0 */
693         b = ROTATE_LEFT(b, 30);
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

748         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 18 */
749         b = ROTATE_LEFT(c, 5) + F(d, e, a) + b + W(2) + SHA1_CONST(0);
750         d = ROTATE_LEFT(d, 30);

752         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 19 */
753         a = ROTATE_LEFT(b, 5) + F(c, d, e) + a + W(3) + SHA1_CONST(0);
754         c = ROTATE_LEFT(c, 30);

756         /* round 2 */
757         W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1);           /* 20 */
758         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(4) + SHA1_CONST(1);
759         b = ROTATE_LEFT(b, 30);
```



```
761         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 21 */
762         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(5) + SHA1_CONST(1);
763         a = ROTATE_LEFT(a, 30);

765         W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1);           /* 22 */
766         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(6) + SHA1_CONST(1);
767         e = ROTATE_LEFT(e, 30);

769         W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1);           /* 23 */
770         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(7) + SHA1_CONST(1);
771         d = ROTATE_LEFT(d, 30);

773         W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1);           /* 24 */
774         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(8) + SHA1_CONST(1);
775         c = ROTATE_LEFT(c, 30);

777         W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1);           /* 25 */
778         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(9) + SHA1_CONST(1);
779         b = ROTATE_LEFT(b, 30);

781         W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1);  /* 26 */
782         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(10) + SHA1_CONST(1);
783         a = ROTATE_LEFT(a, 30);

785         W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1);  /* 27 */
786         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(11) + SHA1_CONST(1);
787         e = ROTATE_LEFT(e, 30);

789         W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1);  /* 28 */
790         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(12) + SHA1_CONST(1);
791         d = ROTATE_LEFT(d, 30);

793         W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 29 */
794         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(13) + SHA1_CONST(1);
795         c = ROTATE_LEFT(c, 30);

797         W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1);  /* 30 */
798         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(14) + SHA1_CONST(1);
799         b = ROTATE_LEFT(b, 30);

801         W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1);  /* 31 */
802         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(15) + SHA1_CONST(1);
803         a = ROTATE_LEFT(a, 30);

805         W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1);           /* 32 */
806         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(0) + SHA1_CONST(1);
807         e = ROTATE_LEFT(e, 30);

809         W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1);           /* 33 */
810         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(1) + SHA1_CONST(1);
811         d = ROTATE_LEFT(d, 30);

813         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 34 */
814         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(2) + SHA1_CONST(1);
815         c = ROTATE_LEFT(c, 30);

817         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 35 */
818         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(3) + SHA1_CONST(1);
819         b = ROTATE_LEFT(b, 30);

821         W(4) = ROTATE_LEFT((W(1) ^ W(12) ^ W(6) ^ W(4)), 1);           /* 36 */
822         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(4) + SHA1_CONST(1);
823         a = ROTATE_LEFT(a, 30);

825         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 37 */
```

```
826         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(5) + SHA1_CONST(1);
827         e = ROTATE_LEFT(e, 30);

829         W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1);           /* 38 */
830         b = ROTATE_LEFT(c, 5) + G(d, e, a) + b + W(6) + SHA1_CONST(1);
831         d = ROTATE_LEFT(d, 30);

833         W(7) = ROTATE_LEFT((W(4) ^ W(15) ^ W(9) ^ W(7)), 1);           /* 39 */
834         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(7) + SHA1_CONST(1);
835         c = ROTATE_LEFT(c, 30);

837         /* round 3 */
838         W(8) = ROTATE_LEFT((W(5) ^ W(0) ^ W(10) ^ W(8)), 1);           /* 40 */
839         e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + W(8) + SHA1_CONST(2);
840         b = ROTATE_LEFT(b, 30);

842         W(9) = ROTATE_LEFT((W(6) ^ W(1) ^ W(11) ^ W(9)), 1);           /* 41 */
843         d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + W(9) + SHA1_CONST(2);
844         a = ROTATE_LEFT(a, 30);

846         W(10) = ROTATE_LEFT((W(7) ^ W(2) ^ W(12) ^ W(10)), 1);  /* 42 */
847         c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + W(10) + SHA1_CONST(2);
848         e = ROTATE_LEFT(e, 30);

850         W(11) = ROTATE_LEFT((W(8) ^ W(3) ^ W(13) ^ W(11)), 1);  /* 43 */
851         b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + W(11) + SHA1_CONST(2);
852         d = ROTATE_LEFT(d, 30);

854         W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1);  /* 44 */
855         a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + W(12) + SHA1_CONST(2);
856         c = ROTATE_LEFT(c, 30);

858         W(13) = ROTATE_LEFT((W(10) ^ W(5) ^ W(15) ^ W(13)), 1); /* 45 */
859         e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + W(13) + SHA1_CONST(2);
860         b = ROTATE_LEFT(b, 30);

862         W(14) = ROTATE_LEFT((W(11) ^ W(6) ^ W(0) ^ W(14)), 1);  /* 46 */
863         d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + W(14) + SHA1_CONST(2);
864         a = ROTATE_LEFT(a, 30);

866         W(15) = ROTATE_LEFT((W(12) ^ W(7) ^ W(1) ^ W(15)), 1);  /* 47 */
867         c = ROTATE_LEFT(d, 5) + H(e, a, b) + c + W(15) + SHA1_CONST(2);
868         e = ROTATE_LEFT(e, 30);

870         W(0) = ROTATE_LEFT((W(13) ^ W(8) ^ W(2) ^ W(0)), 1);           /* 48 */
871         b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + W(0) + SHA1_CONST(2);
872         d = ROTATE_LEFT(d, 30);

874         W(1) = ROTATE_LEFT((W(14) ^ W(9) ^ W(3) ^ W(1)), 1);           /* 49 */
875         a = ROTATE_LEFT(b, 5) + H(c, d, e) + a + W(1) + SHA1_CONST(2);
876         c = ROTATE_LEFT(c, 30);

878         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 50 */
879         e = ROTATE_LEFT(a, 5) + H(b, c, d) + e + W(2) + SHA1_CONST(2);
880         b = ROTATE_LEFT(b, 30);

882         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 51 */
883         d = ROTATE_LEFT(e, 5) + H(a, b, c) + d + W(3) + SHA1_CONST(2);
884         a = ROTATE_LEFT(a, 30);

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

890         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 53 */
891         b = ROTATE_LEFT(c, 5) + H(d, e, a) + b + W(5) + SHA1_CONST(2);
```



```
892         d = ROTATE_LEFT(d, 30);

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

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

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

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

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

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

918         /* round 4 */
919         W(12) = ROTATE_LEFT((W(9) ^ W(4) ^ W(14) ^ W(12)), 1);  /* 60 */
920         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(12) + SHA1_CONST(3);
921         b = ROTATE_LEFT(b, 30);

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

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

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

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

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

943         W(2) = ROTATE_LEFT((W(15) ^ W(10) ^ W(4) ^ W(2)), 1);   /* 66 */
944         d = ROTATE_LEFT(e, 5) + G(a, b, c) + d + W(2) + SHA1_CONST(3);
945         a = ROTATE_LEFT(a, 30);

947         W(3) = ROTATE_LEFT((W(0) ^ W(11) ^ W(5) ^ W(3)), 1);           /* 67 */
948         c = ROTATE_LEFT(d, 5) + G(e, a, b) + c + W(3) + SHA1_CONST(3);
949         e = ROTATE_LEFT(e, 30);

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

955         W(5) = ROTATE_LEFT((W(2) ^ W(13) ^ W(7) ^ W(5)), 1);           /* 69 */
956         a = ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(5) + SHA1_CONST(3);
957         c = ROTATE_LEFT(c, 30);
```

```
 959         W(6) = ROTATE_LEFT((W(3) ^ W(14) ^ W(8) ^ W(6)), 1);            /* 70 */
 960         e = ROTATE_LEFT(a, 5) + G(b, c, d) + e + W(6) + SHA1_CONST(3);
 961         b = ROTATE_LEFT(b, 30);

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

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

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

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

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

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

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

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

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

 997         ctx->state[0] += ROTATE_LEFT(b, 5) + G(c, d, e) + a + W(15) +
 998             SHA1_CONST(3);
 999         ctx->state[1] += b;
1000         ctx->state[2] += ROTATE_LEFT(c, 30);
1001         ctx->state[3] += d;
1002         ctx->state[4] += e;

1004         /* zeroize sensitive information */
1005         W(0) = W(1) = W(2) = W(3) = W(4) = W(5) = W(6) = W(7) = W(8) = 0;
1006         W(9) = W(10) = W(11) = W(12) = W(13) = W(14) = W(15) = 0;
1007 }
1008 #endif  /* !__amd64 */


1011 /*
1012  * Encode()
1013  *
1014  * purpose: to convert a list of numbers from little endian to big endian
1015  *   input: uint8_t *   : place to store the converted big endian numbers
1016  *          uint32_t *  : place to get numbers to convert from
1017  *          size_t      : the length of the input in bytes
1018  *  output: void
1019  */

1021 static void
1022 Encode(uint8_t *_RESTRICT_KYWD output, const uint32_t *_RESTRICT_KYWD input,
1023     size_t len)
```



```
1024 {
1025         size_t          i, j;

1027 #if     defined(__sparc)
1028         if (IS_P2ALIGNED(output, sizeof (uint32_t))) {
1029                 for (i = 0, j = 0; j < len; i++, j += 4) {
1030                         /* LINTED: pointer alignment */
1031                         *((uint32_t *)(output + j)) = input[i];
1032                 }
1033         } else {
1034 #endif  /* little endian -- will work on big endian, but slowly */
1035                 for (i = 0, j = 0; j < len; i++, j += 4) {
1036                         output[j]       = (input[i] >> 24) & 0xff;
1037                         output[j + 1]   = (input[i] >> 16) & 0xff;
1038                         output[j + 2]   = (input[i] >>  8) & 0xff;
1039                         output[j + 3]   = input[i] & 0xff;
1040                 }
1041 #if     defined(__sparc)
1042         }
1043 #endif
1044 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3380 Mon Mar  3 13:15:02 2008
new/usr/src/lib/libmd/Makefile.com
6662791 Need a SHA1 implementation optimized for 64-bit x86
**********************************************************
```

```
#
# 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 "@(#)Makefile.com       1.4     08/03/02 SMI"
# ident "@(#)Makefile.com       1.3     08/01/02 SMI"
#

# $LIBRARY is set in lower makefiles so we can have platform and
# processor optimised versions of this library via libmd_psr and libmd_hwcapN

#LIBRARY= libmd.a
VERS= .1

OBJECTS= md4.o md5.o $(MD5_PSR_OBJECTS) sha1.o $(SHA1_PSR_OBJECTS) sha2.o
OBJECTS= md4.o md5.o $(MD5_PSR_OBJECTS) sha1.o sha2.o

# Use $(SRC) to include makefiles rather than ../../ because the
# platform subdirs are one level deeper so it would be ../../../ for them
include $(SRC)/lib/Makefile.lib
include $(SRC)/lib/Makefile.rootfs

LIBS =          $(DYNLIB) $(LINTLIB)
SRCS = \
        $(COMDIR)/md4/md4.c \
        $(COMDIR)/md5/md5.c \
        $(COMDIR)/sha1/sha1.c \
        $(COMDIR)/sha2/sha2.c

COMDIR= $(SRC)/common/crypto

$(LINTLIB) :=   SRCS = $(SRCDIR)/$(LINTSRC)
LDLIBS +=       -lc

SRCDIR =        ../common
COMDIR =        $(SRC)/common/crypto

CFLAGS += $(CCVERBOSE) $(C_BIGPICFLAGS)
CFLAGS64 += $(C_BIGPICFLAGS)
CPPFLAGS += -I$(SRCDIR)
```



```
# The md5 and sha1 code is very careful about data alignment
# but lint doesn't know that, so just shut lint up.
LINTFLAGS += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTFLAGS64 += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED


ROOTLINT= $(LINTSRC:%=$(ROOTLIBDIR)/%)

.KEEP_STATE:

all: $(LIBS) fnamecheck

lint: lintcheck

pics/%.o: $(COMDIR)/md4/%.c
        $(COMPILE.c) -I$(COMDIR)/md4 -o $@ $<
        $(POST_PROCESS_O)

pics/%.o: $(COMDIR)/md5/%.c
        $(COMPILE.c) -I$(COMDIR)/md5 $(INLINES) -o $@ $<
        $(POST_PROCESS_O)

pics/%.o: $(COMDIR)/sha1/%.c
        $(COMPILE.c) -I$(COMDIR)/sha1 -o $@ $<
        $(POST_PROCESS_O)

pics/%.o: $(COMDIR)/sha1/sparc/$(PLATFORM)/sha1_asm.s
        $(COMPILE.s) -P -DPIC -D_ASM -o $@ $<
        $(POST_PROCESS_O)

pics/%.o: $(COMDIR)/sha2/%.c
        $(COMPILE.c) -I$(COMDIR)/sha2 -o $@ $<
        $(POST_PROCESS_O)

#
# Used when building links in /platform/$(PLATFORM)/lib for libmd_psr.so.1
#

LIBMD_PSR_DIRS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib)
LIBMD_PSR_LINKS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib/$(MODULE))

LIBMD_PSR64_DIRS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib/$(MACH64))
LIBMD_PSR64_LINKS = $(LINKED_PLATFORMS:%=$(ROOT_PLAT_DIR)/%/lib/$(MACH64)/$(MODU

INS.slink6 = $(RM) -r $@; $(SYMLINK) ../../$(PLATFORM)/lib/$(MODULE) $@ $(CHOWNL

INS.slink64 = $(RM) -r $@; $(SYMLINK) ../../../$(PLATFORM)/lib/$(MACH64)/$(MODUL

$(LIBMD_PSR_DIRS):
        -$(INS.dir.root.bin)

$(LIBMD_PSR_LINKS): $(LIBMD_PSR_DIRS)
        -$(INS.slink6)

$(LIBMD_PSR64_DIRS):
        -$(INS.dir.root.bin)

$(LIBMD_PSR64_LINKS): $(LIBMD_PSR64_DIRS)
        -$(INS.slink64)

include $(SRC)/lib/Makefile.targ
```

```
**********************************************************
    1563 Mon Mar  3 13:15:04 2008
new/usr/src/lib/libmd/amd64/Makefile
6662791 Need a SHA1 implementation optimized for 64-bit x86
**********************************************************
#
# 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 "@(#)Makefile   1.3     08/03/02 SMI"
# ident "@(#)Makefile   1.2     08/01/02 SMI"
#

LIBRARY= libmd.a

MD5_PSR_OBJECTS = md5_amd64.o
SHA1_PSR_OBJECTS = sha1-x86_64.o

include ../Makefile.com
include $(SRC)/lib/Makefile.lib.64

CLEANFILES += md5_amd64.s sha1-x86_64.s
CLEANFILES += md5_amd64.s

# This prevents <sys/asm_linkage.h> from including C source:
AS_CPPFLAGS += -D_ASM

install: all $(ROOTLIBS64) $(ROOTLINKS64) $(ROOTLINT64)

pics/md5_amd64.o: md5_amd64.s
        $(COMPILE.s) -o $@ ${@F:.o=.s}
        $(POST_PROCESS_O)

pics/sha1-x86_64.o: sha1-x86_64.s
        $(COMPILE.s) -o $@ ${@F:.o=.s}
        $(POST_PROCESS_O)

md5_amd64.s: $(COMDIR)/md5/amd64/md5_amd64.pl
        $(PERL) $? $@

sha1-x86_64.s: $(COMDIR)/sha1/amd64/sha1-x86_64.pl
        $(PERL) $? $@
pics/md5_amd64.o: md5_amd64.s
        $(COMPILE.s) -o $@ md5_amd64.s
        $(POST_PROCESS_O)
```

```
**********************************************************
    2655 Mon Mar  3 13:15:06 2008
new/usr/src/uts/intel/sha1/Makefile
6662791 Need a SHA1 implementation optimized for 64-bit x86
**********************************************************
```

```
#
# 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
#
#
# uts/intel/sha1/Makefile
#
# Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "@(#)Makefile   1.7     08/03/02 SMI"
#ident  "@(#)Makefile   1.6     06/11/02 SMI"
#
#       This makefile drives the production of the sha1 crypto kernel module.
#
#       intel architecture dependent
#


#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..
COMDIR = $(COMMONBASE)/crypto/sha1

#
#       Define the module and object file sets.
#
MODULE          = sha1
LINTS           = $(SHA1_OBJS:%.o=$(LINTS_DIR)/%.ln)
SHA1_OBJS_32    =
SHA1_OBJS_64    = sha1-x86_64.o
SHA1_OBJS       += $(SHA1_OBJS_$(CLASS))
OBJECTS         = $(SHA1_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(SHA1_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_CRYPTO_DIR)/$(MODULE)
ROOTLINK        = $(ROOT_MISC_DIR)/$(MODULE)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Override defaults
```



```
#
CLEANFILES += sha1-x86_64.s

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW


#
#       Define targets
#
ALL_TARGET      = $(BINARY)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOTLINK)

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

$(ROOTLINK):    $(ROOT_MISC_DIR) $(ROOTMODULE)
        -$(RM) $@; ln $(ROOTMODULE) $@

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ

$(OBJS_DIR)/sha1-x86_64.o: sha1-x86_64.s
        $(COMPILE.s) -o $@ ${@F:.o=.s}
        $(POST_PROCESS_O)

$(OBJS_DIR)/sha1-x86_64.ln: sha1-x86_64.s
        @($(LHEAD) $(LINT.c) ${@F:.ln=.s} $(LTAIL))

sha1-x86_64.s: $(COMDIR)/amd64/sha1-x86_64.pl
        $(PERL) $? $@
```