```
***********************************************************
   11893 Mon Jul 28 07:44:05 2014
new/usr/src/common/atomic/amd64/atomic.s
5043 remove deprecated atomic functions' prototypes
***********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
  24  */

  26         .file   "atomic.s"

  28 #include <sys/asm_linkage.h>

  30 #if defined(_KERNEL)
  31         /*
  32          * Legacy kernel interfaces; they will go away the moment our closed
  33          * bins no longer require them.
  32          * Legacy kernel interfaces; they will go away (eventually).
  34          */
  35         ANSI_PRAGMA_WEAK2(cas8,atomic_cas_8,function)
  36         ANSI_PRAGMA_WEAK2(cas32,atomic_cas_32,function)
  37         ANSI_PRAGMA_WEAK2(cas64,atomic_cas_64,function)
  38         ANSI_PRAGMA_WEAK2(caslong,atomic_cas_ulong,function)
  39         ANSI_PRAGMA_WEAK2(casptr,atomic_cas_ptr,function)
  40         ANSI_PRAGMA_WEAK2(atomic_and_long,atomic_and_ulong,function)
  41         ANSI_PRAGMA_WEAK2(atomic_or_long,atomic_or_ulong,function)
  42 #endif

  44         ENTRY(atomic_inc_8)
  45         ALTENTRY(atomic_inc_uchar)
  46         lock
  47         incb    (%rdi)
  48         ret
  49         SET_SIZE(atomic_inc_uchar)
```

*unchanged_portion_omitted_*

```
***********************************************************
   16169 Mon Jul 28 07:44:05 2014
new/usr/src/common/atomic/i386/atomic.s
5043 remove deprecated atomic functions' prototypes
***********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

        .file   "atomic.s"

#include <sys/asm_linkage.h>

#if defined(_KERNEL)
        /*
         * Legacy kernel interfaces; they will go away the moment our closed
         * bins no longer require them.
         * Legacy kernel interfaces; they will go away (eventually).
         */
        ANSI_PRAGMA_WEAK2(cas8,atomic_cas_8,function)
        ANSI_PRAGMA_WEAK2(cas32,atomic_cas_32,function)
        ANSI_PRAGMA_WEAK2(cas64,atomic_cas_64,function)
        ANSI_PRAGMA_WEAK2(caslong,atomic_cas_ulong,function)
        ANSI_PRAGMA_WEAK2(casptr,atomic_cas_ptr,function)
        ANSI_PRAGMA_WEAK2(atomic_and_long,atomic_and_ulong,function)
        ANSI_PRAGMA_WEAK2(atomic_or_long,atomic_or_ulong,function)
#endif

        ENTRY(atomic_inc_8)
        ALTENTRY(atomic_inc_uchar)
        movl    4(%esp), %eax
        lock
        incb    (%eax)
        ret
        SET_SIZE(atomic_inc_uchar)
```

*_____unchanged_portion_omitted_*

****
23023 Mon Jul 28 07:44:05 2014
new/usr/src/common/atomic/sparc/atomic.s
5043 remove deprecated atomic functions' prototypes
****

```
/*
 * 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.
 */

        .file   "atomic.s"

#include <sys/asm_linkage.h>

#if defined(_KERNEL)
        /*
         * Legacy kernel interfaces; they will go away the moment our closed
         * bins no longer require them.
         * Legacy kernel interfaces; they will go away (eventually).
         */
        ANSI_PRAGMA_WEAK2(cas8,atomic_cas_8,function)
        ANSI_PRAGMA_WEAK2(cas32,atomic_cas_32,function)
        ANSI_PRAGMA_WEAK2(cas64,atomic_cas_64,function)
        ANSI_PRAGMA_WEAK2(caslong,atomic_cas_ulong,function)
        ANSI_PRAGMA_WEAK2(casptr,atomic_cas_ptr,function)
        ANSI_PRAGMA_WEAK2(atomic_and_long,atomic_and_ulong,function)
        ANSI_PRAGMA_WEAK2(atomic_or_long,atomic_or_ulong,function)
        ANSI_PRAGMA_WEAK2(swapl,atomic_swap_32,function)
#endif

        /*
         * NOTE: If atomic_inc_8 and atomic_inc_8_nv are ever
         * separated, you need to also edit the libc sparc platform
         * specific mapfile and remove the NODYNSORT attribute
         * from atomic_inc_8_nv.
         */
        ENTRY(atomic_inc_8)
        ALTENTRY(atomic_inc_8_nv)
        ALTENTRY(atomic_inc_uchar)
        ALTENTRY(atomic_inc_uchar_nv)
        ba      add_8
          add   %g0, 1, %o1
        SET_SIZE(atomic_inc_uchar_nv)
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   26996 Mon Jul 28 07:44:05 2014
new/usr/src/common/atomic/sparcv9/atomic.s
5043 remove deprecated atomic functions' prototypes
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

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

        .file   "atomic.s"

#include <sys/asm_linkage.h>

/*
 * ATOMIC_BO_ENABLE_SHIFT can be selectively defined by processors
 * to enable exponential backoff. No definition means backoff is
 * not desired i.e. backoff should be disabled.
 * By default, the shift value is used to generate a power of 2
 * value for backoff limit. In the kernel, processors scale this
 * shift value with the number of online cpus.
 */

#if defined(_KERNEL)
        /*
         * Legacy kernel interfaces; they will go away the moment our closed
         * bins no longer require them.
         * Legacy kernel interfaces; they will go away (eventually).
         */
        ANSI_PRAGMA_WEAK2(cas8,atomic_cas_8,function)
        ANSI_PRAGMA_WEAK2(cas32,atomic_cas_32,function)
        ANSI_PRAGMA_WEAK2(cas64,atomic_cas_64,function)
        ANSI_PRAGMA_WEAK2(caslong,atomic_cas_ulong,function)
        ANSI_PRAGMA_WEAK2(casptr,atomic_cas_ptr,function)
        ANSI_PRAGMA_WEAK2(atomic_and_long,atomic_and_ulong,function)
        ANSI_PRAGMA_WEAK2(atomic_or_long,atomic_or_ulong,function)
        ANSI_PRAGMA_WEAK2(swapl,atomic_swap_32,function)

#ifdef ATOMIC_BO_ENABLE_SHIFT

#if !defined(lint)
        .weak   cpu_atomic_delay
        .type   cpu_atomic_delay, #function
#endif  /* lint */
```



```
/*
 * For the kernel, invoke processor specific delay routine to perform
 * low-impact spin delay. The value of ATOMIC_BO_ENABLE_SHIFT is tuned
 * with respect to the specific spin delay implementation.
 */
#define DELAY_SPIN(label, tmp1, tmp2)                                   \
        /*                                                              ; \
         * Define a pragma weak reference to a cpu specific             ; \
         * delay routine for atomic backoff. For CPUs that              ; \
         * have no such delay routine defined, the delay becomes        ; \
         * just a simple tight loop.                                    ; \
         *                                                              ; \
         * tmp1 = holds CPU specific delay routine                      ; \
         * tmp2 = holds atomic routine's callee return address          ; \
         */                                                             ; \
        sethi   %hi(cpu_atomic_delay), tmp1                             ; \
        or      tmp1, %lo(cpu_atomic_delay), tmp1                       ; \
label/**/0:                                                             ; \
        brz,pn  tmp1, label/**/1                                        ; \
        mov     %o7, tmp2                                               ; \
        jmpl    tmp1, %o7       /* call CPU specific delay routine */   ; \
          nop                   /* delay slot : do nothing */           ; \
        mov     tmp2, %o7       /* restore callee's return address */   ; \
label/**/1:

/*
 * For the kernel, we take into consideration of cas failures
 * and also scale the backoff limit w.r.t. the number of cpus.
 * For cas failures, we reset the backoff value to 1 if the cas
 * failures exceed or equal to the number of online cpus. This
 * will enforce some degree of fairness and prevent starvation.
 * We also scale/normalize the processor provided specific
 * ATOMIC_BO_ENABLE_SHIFT w.r.t. the number of online cpus to
 * obtain the actual final limit to use.
 */
#define ATOMIC_BACKOFF_CPU(val, limit, ncpu, cas_cnt, label)            \
        brnz,pt ncpu, label/**/0                                        ; \
          inc   cas_cnt                                                 ; \
        sethi   %hi(ncpus_online), ncpu                                 ; \
        ld      [ncpu + %lo(ncpus_online)], ncpu                        ; \
label/**/0:                                                             ; \
        cmp     cas_cnt, ncpu                                           ; \
        blu,pt  %xcc, label/**/1                                        ; \
          sllx  ncpu, ATOMIC_BO_ENABLE_SHIFT, limit                     ; \
        mov     %g0, cas_cnt                                            ; \
        mov     1, val                                                  ; \
label/**/1:
#endif  /* ATOMIC_BO_ENABLE_SHIFT */

#else   /* _KERNEL */

/*
 * ATOMIC_BO_ENABLE_SHIFT may be enabled/defined here for generic
 * libc atomics. None for now.
 */
#ifdef ATOMIC_BO_ENABLE_SHIFT
#define DELAY_SPIN(label, tmp1, tmp2)   \
label/**/0:

#define ATOMIC_BACKOFF_CPU(val, limit, ncpu, cas_cnt, label)  \
        set     1 << ATOMIC_BO_ENABLE_SHIFT, limit
#endif  /* ATOMIC_BO_ENABLE_SHIFT */
#endif  /* _KERNEL */

#ifdef ATOMIC_BO_ENABLE_SHIFT
/*
```

```
 127  * ATOMIC_BACKOFF_INIT macro for initialization.
 128  * backoff val is initialized to 1.
 129  * ncpu is initialized to 0
 130  * The cas_cnt counts the cas instruction failure and is
 131  * initialized to 0.
 132  */
 133 #define ATOMIC_BACKOFF_INIT(val, ncpu, cas_cnt) \
 134         mov     1, val                          ; \
 135         mov     %g0, ncpu                       ; \
 136         mov     %g0, cas_cnt

 138 #define ATOMIC_BACKOFF_BRANCH(cr, backoff, loop) \
 139         bne,a,pn cr, backoff

 141 /*
 142  * Main ATOMIC_BACKOFF_BACKOFF macro for backoff.
 143  */
 144 #define ATOMIC_BACKOFF_BACKOFF(val, limit, ncpu, cas_cnt, label, retlabel) \
 145         ATOMIC_BACKOFF_CPU(val, limit, ncpu, cas_cnt, label/**/_0)      ; \
 146         cmp     val, limit                                              ; \
 147         blu,a,pt %xcc, label/**/_1                                      ; \
 148           mov   val, limit                                              ; \
 149 label/**/_1:                                                            ; \
 150         mov     limit, val                                              ; \
 151         DELAY_SPIN(label/**/_2, %g2, %g3)                               ; \
 152         deccc   limit                                                   ; \
 153         bgu,pn  %xcc, label/**/_20 /* branch to middle of DELAY_SPIN */ ; \
 154           nop                                                           ; \
 155         ba      retlabel                                                ; \
 156         sllx    val, 1, val

 158 #else   /* ATOMIC_BO_ENABLE_SHIFT */
 159 #define ATOMIC_BACKOFF_INIT(val, ncpu, cas_cnt)

 161 #define ATOMIC_BACKOFF_BRANCH(cr, backoff, loop) \
 162         bne,a,pn cr, loop

 164 #define ATOMIC_BACKOFF_BACKOFF(val, limit, ncpu, cas_cnt, label, retlabel)
 165 #endif  /* ATOMIC_BO_ENABLE_SHIFT */

 167         /*
 168          * NOTE: If atomic_inc_8 and atomic_inc_8_nv are ever
 169          * separated, you need to also edit the libc sparcv9 platform
 170          * specific mapfile and remove the NODYNSORT attribute
 171          * from atomic_inc_8_nv.
 172          */
 173         ENTRY(atomic_inc_8)
 174         ALTENTRY(atomic_inc_8_nv)
 175         ALTENTRY(atomic_inc_uchar)
 176         ALTENTRY(atomic_inc_uchar_nv)
 177         ba      add_8
 178           add   %g0, 1, %o1
 179         SET_SIZE(atomic_inc_uchar_nv)
```

*unchanged_portion_omitted_*

```
**********************************************************
   14981 Mon Jul 28 07:44:06 2014
new/usr/src/uts/common/sys/atomic.h
5043 remove deprecated atomic functions' prototypes
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */

  27 #ifndef _SYS_ATOMIC_H
  28 #define _SYS_ATOMIC_H

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

  30 #include <sys/types.h>
  31 #include <sys/inttypes.h>

  33 #ifdef  __cplusplus
  34 extern "C" {
  35 #endif

  37 #if defined(_KERNEL) && defined(__GNUC__) && defined(_ASM_INLINES) && \
  38         (defined(__i386) || defined(__amd64))
  39 #include <asm/atomic.h>
  40 #endif

  42 #if defined(_KERNEL) || defined(__STDC__)
  43 /*
  44  * Increment target.
  45  */
  46 extern void atomic_inc_8(volatile uint8_t *);
  47 extern void atomic_inc_uchar(volatile uchar_t *);
  48 extern void atomic_inc_16(volatile uint16_t *);
  49 extern void atomic_inc_ushort(volatile ushort_t *);
  50 extern void atomic_inc_32(volatile uint32_t *);
  51 extern void atomic_inc_uint(volatile uint_t *);
  52 extern void atomic_inc_ulong(volatile ulong_t *);
  53 #if defined(_KERNEL) || defined(_INT64_TYPE)
  54 extern void atomic_inc_64(volatile uint64_t *);
  55 #endif

  57 /*
  58  * Decrement target
  59  */
```



```
  60 extern void atomic_dec_8(volatile uint8_t *);
  61 extern void atomic_dec_uchar(volatile uchar_t *);
  62 extern void atomic_dec_16(volatile uint16_t *);
  63 extern void atomic_dec_ushort(volatile ushort_t *);
  64 extern void atomic_dec_32(volatile uint32_t *);
  65 extern void atomic_dec_uint(volatile uint_t *);
  66 extern void atomic_dec_ulong(volatile ulong_t *);
  67 #if defined(_KERNEL) || defined(_INT64_TYPE)
  68 extern void atomic_dec_64(volatile uint64_t *);
  69 #endif

  71 /*
  72  * Add delta to target
  73  */
  74 extern void atomic_add_8(volatile uint8_t *, int8_t);
  75 extern void atomic_add_char(volatile uchar_t *, signed char);
  76 extern void atomic_add_16(volatile uint16_t *, int16_t);
  77 extern void atomic_add_short(volatile ushort_t *, short);
  78 extern void atomic_add_32(volatile uint32_t *, int32_t);
  79 extern void atomic_add_int(volatile uint_t *, int);
  80 extern void atomic_add_ptr(volatile void *, ssize_t);
  81 extern void atomic_add_long(volatile ulong_t *, long);
  82 #if defined(_KERNEL) || defined(_INT64_TYPE)
  83 extern void atomic_add_64(volatile uint64_t *, int64_t);
  84 #endif

  86 /*
  87  * logical OR bits with target
  88  */
  89 extern void atomic_or_8(volatile uint8_t *, uint8_t);
  90 extern void atomic_or_uchar(volatile uchar_t *, uchar_t);
  91 extern void atomic_or_16(volatile uint16_t *, uint16_t);
  92 extern void atomic_or_ushort(volatile ushort_t *, ushort_t);
  93 extern void atomic_or_32(volatile uint32_t *, uint32_t);
  94 extern void atomic_or_uint(volatile uint_t *, uint_t);
  95 extern void atomic_or_ulong(volatile ulong_t *, ulong_t);
  96 #if defined(_KERNEL) || defined(_INT64_TYPE)
  97 extern void atomic_or_64(volatile uint64_t *, uint64_t);
  98 #endif

 100 /*
 101  * logical AND bits with target
 102  */
 103 extern void atomic_and_8(volatile uint8_t *, uint8_t);
 104 extern void atomic_and_uchar(volatile uchar_t *, uchar_t);
 105 extern void atomic_and_16(volatile uint16_t *, uint16_t);
 106 extern void atomic_and_ushort(volatile ushort_t *, ushort_t);
 107 extern void atomic_and_32(volatile uint32_t *, uint32_t);
 108 extern void atomic_and_uint(volatile uint_t *, uint_t);
 109 extern void atomic_and_ulong(volatile ulong_t *, ulong_t);
 110 #if defined(_KERNEL) || defined(_INT64_TYPE)
 111 extern void atomic_and_64(volatile uint64_t *, uint64_t);
 112 #endif

 114 /*
 115  * As above, but return the new value.  Note that these _nv() variants are
 116  * substantially more expensive on some platforms than the no-return-value
 117  * versions above, so don't use them unless you really need to know the
 118  * new value *atomically* (e.g. when decrementing a reference count and
 119  * checking whether it went to zero).
 120  */

 122 /*
 123  * Increment target and return new value.
 124  */
 125 extern uint8_t atomic_inc_8_nv(volatile uint8_t *);
```

```
126 extern uchar_t atomic_inc_uchar_nv(volatile uchar_t *);
127 extern uint16_t atomic_inc_16_nv(volatile uint16_t *);
128 extern ushort_t atomic_inc_ushort_nv(volatile ushort_t *);
129 extern uint32_t atomic_inc_32_nv(volatile uint32_t *);
130 extern uint_t atomic_inc_uint_nv(volatile uint_t *);
131 extern ulong_t atomic_inc_ulong_nv(volatile ulong_t *);
132 #if defined(_KERNEL) || defined(_INT64_TYPE)
133 extern uint64_t atomic_inc_64_nv(volatile uint64_t *);
134 #endif

136 /*
137  * Decrement target and return new value.
138  */
139 extern uint8_t atomic_dec_8_nv(volatile uint8_t *);
140 extern uchar_t atomic_dec_uchar_nv(volatile uchar_t *);
141 extern uint16_t atomic_dec_16_nv(volatile uint16_t *);
142 extern ushort_t atomic_dec_ushort_nv(volatile ushort_t *);
143 extern uint32_t atomic_dec_32_nv(volatile uint32_t *);
144 extern uint_t atomic_dec_uint_nv(volatile uint_t *);
145 extern ulong_t atomic_dec_ulong_nv(volatile ulong_t *);
146 #if defined(_KERNEL) || defined(_INT64_TYPE)
147 extern uint64_t atomic_dec_64_nv(volatile uint64_t *);
148 #endif

150 /*
151  * Add delta to target
152  */
153 extern uint8_t atomic_add_8_nv(volatile uint8_t *, int8_t);
154 extern uchar_t atomic_add_char_nv(volatile uchar_t *, signed char);
155 extern uint16_t atomic_add_16_nv(volatile uint16_t *, int16_t);
156 extern ushort_t atomic_add_short_nv(volatile ushort_t *, short);
157 extern uint32_t atomic_add_32_nv(volatile uint32_t *, int32_t);
158 extern uint_t atomic_add_int_nv(volatile uint_t *, int);
159 extern void *atomic_add_ptr_nv(volatile void *, ssize_t);
160 extern ulong_t atomic_add_long_nv(volatile ulong_t *, long);
161 #if defined(_KERNEL) || defined(_INT64_TYPE)
162 extern uint64_t atomic_add_64_nv(volatile uint64_t *, int64_t);
163 #endif

165 /*
166  * logical OR bits with target and return new value.
167  */
168 extern uint8_t atomic_or_8_nv(volatile uint8_t *, uint8_t);
169 extern uchar_t atomic_or_uchar_nv(volatile uchar_t *, uchar_t);
170 extern uint16_t atomic_or_16_nv(volatile uint16_t *, uint16_t);
171 extern ushort_t atomic_or_ushort_nv(volatile ushort_t *, ushort_t);
172 extern uint32_t atomic_or_32_nv(volatile uint32_t *, uint32_t);
173 extern uint_t atomic_or_uint_nv(volatile uint_t *, uint_t);
174 extern ulong_t atomic_or_ulong_nv(volatile ulong_t *, ulong_t);
175 #if defined(_KERNEL) || defined(_INT64_TYPE)
176 extern uint64_t atomic_or_64_nv(volatile uint64_t *, uint64_t);
177 #endif

179 /*
180  * logical AND bits with target and return new value.
181  */
182 extern uint8_t atomic_and_8_nv(volatile uint8_t *, uint8_t);
183 extern uchar_t atomic_and_uchar_nv(volatile uchar_t *, uchar_t);
184 extern uint16_t atomic_and_16_nv(volatile uint16_t *, uint16_t);
185 extern ushort_t atomic_and_ushort_nv(volatile ushort_t *, ushort_t);
186 extern uint32_t atomic_and_32_nv(volatile uint32_t *, uint32_t);
187 extern uint_t atomic_and_uint_nv(volatile uint_t *, uint_t);
188 extern ulong_t atomic_and_ulong_nv(volatile ulong_t *, ulong_t);
189 #if defined(_KERNEL) || defined(_INT64_TYPE)
190 extern uint64_t atomic_and_64_nv(volatile uint64_t *, uint64_t);
191 #endif
```



```
193 /*
194  * If *arg1 == arg2, set *arg1 = arg3; return old value
195  */
196 extern uint8_t atomic_cas_8(volatile uint8_t *, uint8_t, uint8_t);
197 extern uchar_t atomic_cas_uchar(volatile uchar_t *, uchar_t, uchar_t);
198 extern uint16_t atomic_cas_16(volatile uint16_t *, uint16_t, uint16_t);
199 extern ushort_t atomic_cas_ushort(volatile ushort_t *, ushort_t, ushort_t);
200 extern uint32_t atomic_cas_32(volatile uint32_t *, uint32_t, uint32_t);
201 extern uint_t atomic_cas_uint(volatile uint_t *, uint_t, uint_t);
202 extern void *atomic_cas_ptr(volatile void *, void *, void *);
203 extern ulong_t atomic_cas_ulong(volatile ulong_t *, ulong_t, ulong_t);
204 #if defined(_KERNEL) || defined(_INT64_TYPE)
205 extern uint64_t atomic_cas_64(volatile uint64_t *, uint64_t, uint64_t);
206 #endif

208 /*
209  * Swap target and return old value
210  */
211 extern uint8_t atomic_swap_8(volatile uint8_t *, uint8_t);
212 extern uchar_t atomic_swap_uchar(volatile uchar_t *, uchar_t);
213 extern uint16_t atomic_swap_16(volatile uint16_t *, uint16_t);
214 extern ushort_t atomic_swap_ushort(volatile ushort_t *, ushort_t);
215 extern uint32_t atomic_swap_32(volatile uint32_t *, uint32_t);
216 extern uint_t atomic_swap_uint(volatile uint_t *, uint_t);
217 extern void *atomic_swap_ptr(volatile void *, void *);
218 extern ulong_t atomic_swap_ulong(volatile ulong_t *, ulong_t);
219 #if defined(_KERNEL) || defined(_INT64_TYPE)
220 extern uint64_t atomic_swap_64(volatile uint64_t *, uint64_t);
221 #endif

223 /*
224  * Perform an exclusive atomic bit set/clear on a target.
225  * Returns 0 if bit was sucessfully set/cleared, or -1
226  * if the bit was already set/cleared.
227  */
228 extern int atomic_set_long_excl(volatile ulong_t *, uint_t);
229 extern int atomic_clear_long_excl(volatile ulong_t *, uint_t);

231 /*
232  * Generic memory barrier used during lock entry, placed after the
233  * memory operation that acquires the lock to guarantee that the lock
234  * protects its data.  No stores from after the memory barrier will
235  * reach visibility, and no loads from after the barrier will be
236  * resolved, before the lock acquisition reaches global visibility.
237  */
238 extern void membar_enter(void);

240 /*
241  * Generic memory barrier used during lock exit, placed before the
242  * memory operation that releases the lock to guarantee that the lock
243  * protects its data.  All loads and stores issued before the barrier
244  * will be resolved before the subsequent lock update reaches visibility.
245  */
246 extern void membar_exit(void);

248 /*
249  * Arrange that all stores issued before this point in the code reach
250  * global visibility before any stores that follow; useful in producer
251  * modules that update a data item, then set a flag that it is available.
252  * The memory barrier guarantees that the available flag is not visible
253  * earlier than the updated data, i.e. it imposes store ordering.
254  */
255 extern void membar_producer(void);

257 /*
```

```
 258  * Arrange that all loads issued before this point in the code are
 259  * completed before any subsequent loads; useful in consumer modules
 260  * that check to see if data is available and read the data.
 261  * The memory barrier guarantees that the data is not sampled until
 262  * after the available flag has been seen, i.e. it imposes load ordering.
 263  */
 264 extern void membar_consumer(void);
 265 #endif

 267 #if !defined(_KERNEL) && !defined(__STDC__)
 268 extern void atomic_inc_8();
 269 extern void atomic_inc_uchar();
 270 extern void atomic_inc_16();
 271 extern void atomic_inc_ushort();
 272 extern void atomic_inc_32();
 273 extern void atomic_inc_uint();
 274 extern void atomic_inc_ulong();
 275 #if defined(_INT64_TYPE)
 276 extern void atomic_inc_64();
 277 #endif /* defined(_INT64_TYPE) */
 278 extern void atomic_dec_8();
 279 extern void atomic_dec_uchar();
 280 extern void atomic_dec_16();
 281 extern void atomic_dec_ushort();
 282 extern void atomic_dec_32();
 283 extern void atomic_dec_uint();
 284 extern void atomic_dec_ulong();
 285 #if defined(_INT64_TYPE)
 286 extern void atomic_dec_64();
 287 #endif /* defined(_INT64_TYPE) */
 288 extern void atomic_add_8();
 289 extern void atomic_add_char();
 290 extern void atomic_add_16();
 291 extern void atomic_add_short();
 292 extern void atomic_add_32();
 293 extern void atomic_add_int();
 294 extern void atomic_add_ptr();
 295 extern void atomic_add_long();
 296 #if defined(_INT64_TYPE)
 297 extern void atomic_add_64();
 298 #endif /* defined(_INT64_TYPE) */
 299 extern void atomic_or_8();
 300 extern void atomic_or_uchar();
 301 extern void atomic_or_16();
 302 extern void atomic_or_ushort();
 303 extern void atomic_or_32();
 304 extern void atomic_or_uint();
 305 extern void atomic_or_ulong();
 306 #if defined(_INT64_TYPE)
 307 extern void atomic_or_64();
 308 #endif /* defined(_INT64_TYPE) */
 309 extern void atomic_and_8();
 310 extern void atomic_and_uchar();
 311 extern void atomic_and_16();
 312 extern void atomic_and_ushort();
 313 extern void atomic_and_32();
 314 extern void atomic_and_uint();
 315 extern void atomic_and_ulong();
 316 #if defined(_INT64_TYPE)
 317 extern void atomic_and_64();
 318 #endif /* defined(_INT64_TYPE) */
 319 extern uint8_t atomic_inc_8_nv();
 320 extern uchar_t atomic_inc_uchar_nv();
 321 extern uint16_t atomic_inc_16_nv();
 322 extern ushort_t atomic_inc_ushort_nv();
 323 extern uint32_t atomic_inc_32_nv();
```



```
 324 extern uint_t atomic_inc_uint_nv();
 325 extern ulong_t atomic_inc_ulong_nv();
 326 #if defined(_INT64_TYPE)
 327 extern uint64_t atomic_inc_64_nv();
 328 #endif /* defined(_INT64_TYPE) */
 329 extern uint8_t atomic_dec_8_nv();
 330 extern uchar_t atomic_dec_uchar_nv();
 331 extern uint16_t atomic_dec_16_nv();
 332 extern ushort_t atomic_dec_ushort_nv();
 333 extern uint32_t atomic_dec_32_nv();
 334 extern uint_t atomic_dec_uint_nv();
 335 extern ulong_t atomic_dec_ulong_nv();
 336 #if defined(_INT64_TYPE)
 337 extern uint64_t atomic_dec_64_nv();
 338 #endif /* defined(_INT64_TYPE) */
 339 extern uint8_t atomic_add_8_nv();
 340 extern uchar_t atomic_add_char_nv();
 341 extern uint16_t atomic_add_16_nv();
 342 extern ushort_t atomic_add_short_nv();
 343 extern uint32_t atomic_add_32_nv();
 344 extern uint_t atomic_add_int_nv();
 345 extern void *atomic_add_ptr_nv();
 346 extern ulong_t atomic_add_long_nv();
 347 #if defined(_INT64_TYPE)
 348 extern uint64_t atomic_add_64_nv();
 349 #endif /* defined(_INT64_TYPE) */
 350 extern uint8_t atomic_or_8_nv();
 351 extern uchar_t atomic_or_uchar_nv();
 352 extern uint16_t atomic_or_16_nv();
 353 extern ushort_t atomic_or_ushort_nv();
 354 extern uint32_t atomic_or_32_nv();
 355 extern uint_t atomic_or_uint_nv();
 356 extern ulong_t atomic_or_ulong_nv();
 357 #if defined(_INT64_TYPE)
 358 extern uint64_t atomic_or_64_nv();
 359 #endif /* defined(_INT64_TYPE) */
 360 extern uint8_t atomic_and_8_nv();
 361 extern uchar_t atomic_and_uchar_nv();
 362 extern uint16_t atomic_and_16_nv();
 363 extern ushort_t atomic_and_ushort_nv();
 364 extern uint32_t atomic_and_32_nv();
 365 extern uint_t atomic_and_uint_nv();
 366 extern ulong_t atomic_and_ulong_nv();
 367 #if defined(_INT64_TYPE)
 368 extern uint64_t atomic_and_64_nv();
 369 #endif /* defined(_INT64_TYPE) */
 370 extern uint8_t atomic_cas_8();
 371 extern uchar_t atomic_cas_uchar();
 372 extern uint16_t atomic_cas_16();
 373 extern ushort_t atomic_cas_ushort();
 374 extern uint32_t atomic_cas_32();
 375 extern uint_t atomic_cas_uint();
 376 extern void *atomic_cas_ptr();
 377 extern ulong_t atomic_cas_ulong();
 378 #if defined(_INT64_TYPE)
 379 extern uint64_t atomic_cas_64();
 380 #endif /* defined(_INT64_TYPE) */
 381 extern uint8_t atomic_swap_8();
 382 extern uchar_t atomic_swap_uchar();
 383 extern uint16_t atomic_swap_16();
 384 extern ushort_t atomic_swap_ushort();
 385 extern uint32_t atomic_swap_32();
 386 extern uint_t atomic_swap_uint();
 387 extern void *atomic_swap_ptr();
 388 extern ulong_t atomic_swap_ulong();
 389 #if defined(_INT64_TYPE)
```

```
 390 extern uint64_t atomic_swap_64();
 391 #endif /* defined(_INT64_TYPE) */


 394 extern int atomic_set_long_excl();
 395 extern int atomic_clear_long_excl();

 397 extern void membar_enter();
 398 extern void membar_exit();
 399 extern void membar_producer();
 400 extern void membar_consumer();

 402 #endif

 404 #if defined(_KERNEL)

 406 #if defined(_LP64) || defined(_ILP32)
 407 #define atomic_add_ip           atomic_add_long
 408 #define atomic_add_ip_nv        atomic_add_long_nv
 409 #define casip                   atomic_cas_ulong
 410 #endif

 412 #if defined(__sparc)
 413 extern uint8_t ldstub(uint8_t *);
 414 #endif

 418 /*
 419  * Legacy kernel interfaces; they will go away (eventually).
 420  */
 421 extern uint8_t cas8(uint8_t *, uint8_t, uint8_t);
 422 extern uint32_t cas32(uint32_t *, uint32_t, uint32_t);
 423 extern uint64_t cas64(uint64_t *, uint64_t, uint64_t);
 424 extern ulong_t caslong(ulong_t *, ulong_t, ulong_t);
 425 extern void *casptr(void *, void *, void *);
 426 extern void atomic_and_long(ulong_t *, ulong_t);
 427 extern void atomic_or_long(ulong_t *, ulong_t);
 428 #if defined(__sparc)
 429 extern uint32_t swapl(uint32_t *, uint32_t);
 430 #endif

 416 #endif  /* _KERNEL */

 418 #ifdef  __cplusplus
 419 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   12278 Mon Jul 28 07:44:06 2014
new/usr/src/uts/sun4v/cpu/mach_cpu_module.c
5043 remove deprecated atomic functions' prototypes
**********************************************************
______unchanged_portion_omitted_

 130 void
 131 kdi_flush_caches(void)
 132 {}

 134 /*ARGSUSED*/
 135 int
 136 kzero(void *addr, size_t count)
 137 { return (0); }

 139 /*ARGSUSED*/
 140 void
 141 uzero(void *addr, size_t count)
 142 {}

 144 /*ARGSUSED*/
 145 void
 146 bzero(void *addr, size_t count)
 147 {}

 149 /*ARGSUSED*/
 150 void
 151 cpu_inv_tsb(caddr_t tsb_base, uint_t tsb_bytes)
 152 {}

 154 /*
 155  *  Atomic Function Stubs
 156  */

 158 uint8_t
 159 cas8(uint8_t *target, uint8_t value1, uint8_t value2)
 160 { return (0); }

 162 /* ARGSUSED */
 163 uint32_t
 164 cas32(uint32_t *target, uint32_t value1, uint32_t value2)
 165 { return (0); }

 167 /* ARGSUSED */
 168 uint64_t
 169 cas64(uint64_t *target, uint64_t value1, uint64_t value2)
 170 { return (0); }

 172 /* ARGSUSED */
 173 ulong_t
 174 caslong(ulong_t *target, ulong_t value1, ulong_t value2)
 175 { return (0); }

 177 /* ARGSUSED */
 178 void *
 179 casptr(void *ptr1, void *ptr2, void *ptr3)
 180 { return (0); }

 182 /* ARGSUSED */
 183 void
 184 atomic_and_long(ulong_t *target, ulong_t value)
 185 {}

 187 /* ARGSUSED */
 188 void
```



```
 189 atomic_or_long(ulong_t *target, ulong_t value)
 190 {}

 158 /* ARGSUSED */
 159 void
 160 atomic_inc_8(volatile uint8_t *target)
 161 {}

 163 /* ARGSUSED */
 164 void
 165 atomic_inc_uchar(volatile uchar_t *target)
 166 {}

 168 /* ARGSUSED */
 169 void
 170 atomic_inc_16(volatile uint16_t *target)
 171 {}

 173 /* ARGSUSED */
 174 void
 175 atomic_inc_ushort(volatile ushort_t *target)
 176 {}

 178 /* ARGSUSED */
 179 void
 180 atomic_inc_32(volatile uint32_t *target)
 181 {}

 183 /* ARGSUSED */
 184 void
 185 atomic_inc_uint(volatile uint_t *target)
 186 {}

 188 /* ARGSUSED */
 189 void
 190 atomic_inc_ulong(volatile ulong_t *target)
 191 {}

 193 /* ARGSUSED */
 194 void
 195 atomic_inc_64(volatile uint64_t *target)
 196 {}

 198 /* ARGSUSED */
 199 void
 200 atomic_dec_8(volatile uint8_t *target)
 201 {}

 203 /* ARGSUSED */
 204 void
 205 atomic_dec_uchar(volatile uchar_t *target)
 206 {}

 208 /* ARGSUSED */
 209 void
 210 atomic_dec_16(volatile uint16_t *target)
 211 {}

 213 /* ARGSUSED */
 214 void
 215 atomic_dec_ushort(volatile ushort_t *target)
 216 {}

 218 /* ARGSUSED */
 219 void
 220 atomic_dec_32(volatile uint32_t *target)
```

```
221 {}

223 /* ARGSUSED */
224 void
225 atomic_dec_uint(volatile uint_t *target)
226 {}

228 /* ARGSUSED */
229 void
230 atomic_dec_ulong(volatile ulong_t *target)
231 {}

233 /* ARGSUSED */
234 void
235 atomic_dec_64(volatile uint64_t *target)
236 {}

238 /* ARGSUSED */
239 void
240 atomic_add_8(volatile uint8_t *target, int8_t value)
241 {}

243 /* ARGSUSED */
244 void
245 atomic_add_char(volatile uchar_t *target, signed char value)
246 {}

248 /* ARGSUSED */
249 void
250 atomic_add_16(volatile uint16_t *target, int16_t delta)
251 {}

253 /* ARGSUSED */
254 void
255 atomic_add_ushort(volatile ushort_t *target, short value)
256 {}

258 /* ARGSUSED */
259 void
260 atomic_add_32(volatile uint32_t *target, int32_t delta)
261 {}

263 /* ARGSUSED */
264 void
265 atomic_add_ptr(volatile void *target, ssize_t value)
266 {}

268 /* ARGSUSED */
269 void
270 atomic_add_long(volatile ulong_t *target, long delta)
271 {}

273 /* ARGSUSED */
274 void
275 atomic_add_64(volatile uint64_t *target, int64_t delta)
276 {}

278 /* ARGSUSED */
279 void
280 atomic_or_8(volatile uint8_t *target, uint8_t bits)
281 {}

283 /* ARGSUSED */
284 void
285 atomic_or_uchar(volatile uchar_t *target, uchar_t bits)
286 {}
```



```
288 /* ARGSUSED */
289 void
290 atomic_or_16(volatile uint16_t *target, uint16_t bits)
291 {}

293 /* ARGSUSED */
294 void
295 atomic_or_ushort(volatile ushort_t *target, ushort_t bits)
296 {}

298 /* ARGSUSED */
299 void
300 atomic_or_32(volatile uint32_t *target, uint32_t bits)
301 {}

303 /* ARGSUSED */
304 void
305 atomic_or_uint(volatile uint_t *target, uint_t bits)
306 {}

308 /* ARGSUSED */
309 void
310 atomic_or_ulong(volatile ulong_t *target, ulong_t bits)
311 {}

313 /* ARGSUSED */
314 void
315 atomic_or_64(volatile uint64_t *target, uint64_t bits)
316 {}

318 /* ARGSUSED */
319 void
320 atomic_and_8(volatile uint8_t *target, uint8_t bits)
321 {}

323 /* ARGSUSED */
324 void
325 atomic_and_uchar(volatile uchar_t *target, uchar_t bits)
326 {}

328 /* ARGSUSED */
329 void
330 atomic_and_16(volatile uint16_t *target, uint16_t bits)
331 {}

333 /* ARGSUSED */
334 void
335 atomic_and_ushort(volatile ushort_t *target, ushort_t bits)
336 {}

338 /* ARGSUSED */
339 void
340 atomic_and_32(volatile uint32_t *target, uint32_t bits)
341 {}

343 /* ARGSUSED */
344 void
345 atomic_and_uint(volatile uint_t *target, uint_t bits)
346 {}

348 /* ARGSUSED */
349 void
350 atomic_and_ulong(volatile ulong_t *target, ulong_t bits)
351 {}
```

```
353 /* ARGSUSED */
354 void
355 atomic_and_64(volatile uint64_t *target, uint64_t bits)
356 {}

358 /* ARGSUSED */
359 uint8_t
360 atomic_inc_8_nv(volatile uint8_t *target)
361 { return (0); }

363 /* ARGSUSED */
364 uchar_t
365 atomic_inc_uchar_nv(volatile uchar_t *target)
366 { return (0); }

368 /* ARGSUSED */
369 uint16_t
370 atomic_inc_16_nv(volatile uint16_t *target)
371 { return (0); }

373 /* ARGSUSED */
374 ushort_t
375 atomic_inc_ushort_nv(volatile ushort_t *target)
376 { return (0); }

378 /* ARGSUSED */
379 uint32_t
380 atomic_inc_32_nv(volatile uint32_t *target)
381 { return (0); }

383 /* ARGSUSED */
384 uint_t
385 atomic_inc_uint_nv(volatile uint_t *target)
386 { return (0); }

388 /* ARGSUSED */
389 ulong_t
390 atomic_inc_ulong_nv(volatile ulong_t *target)
391 { return (0); }

393 /* ARGSUSED */
394 uint64_t
395 atomic_inc_64_nv(volatile uint64_t *target)
396 { return (0); }

398 /* ARGSUSED */
399 uint8_t
400 atomic_dec_8_nv(volatile uint8_t *target)
401 { return (0); }

403 /* ARGSUSED */
404 uchar_t
405 atomic_dec_uchar_nv(volatile uchar_t *target)
406 { return (0); }

408 /* ARGSUSED */
409 uint16_t
410 atomic_dec_16_nv(volatile uint16_t *target)
411 { return (0); }

413 /* ARGSUSED */
414 ushort_t
415 atomic_dec_ushort_nv(volatile ushort_t *target)
416 { return (0); }

418 /* ARGSUSED */
```



```
419 uint32_t
420 atomic_dec_32_nv(volatile uint32_t *target)
421 { return (0); }

423 /* ARGSUSED */
424 uint_t
425 atomic_dec_uint_nv(volatile uint_t *target)
426 { return (0); }

428 /* ARGSUSED */
429 ulong_t
430 atomic_dec_ulong_nv(volatile ulong_t *target)
431 { return (0); }

433 /* ARGSUSED */
434 uint64_t
435 atomic_dec_64_nv(volatile uint64_t *target)
436 { return (0); }

438 /* ARGSUSED */
439 uint8_t
440 atomic_add_8_nv(volatile uint8_t *target, int8_t value)
441 { return (0); }

443 /* ARGSUSED */
444 uchar_t
445 atomic_add_char_nv(volatile uchar_t *target, signed char value)
446 { return (0); }

448 /* ARGSUSED */
449 uint16_t
450 atomic_add_16_nv(volatile uint16_t *target, int16_t delta)
451 { return (0); }

453 /* ARGSUSED */
454 ushort_t
455 atomic_add_short_nv(volatile ushort_t *target, short value)
456 { return (0); }

458 /* ARGSUSED */
459 uint32_t
460 atomic_add_32_nv(volatile uint32_t *target, int32_t delta)
461 { return (0); }

463 /* ARGSUSED */
464 uint_t
465 atomic_add_int_nv(volatile uint_t *target, int delta)
466 { return (0); }

468 /* ARGSUSED */
469 void *
470 atomic_add_ptr_nv(volatile void *target, ssize_t value)
471 { return (NULL); }

473 /* ARGSUSED */
474 ulong_t
475 atomic_add_long_nv(volatile ulong_t *target, long delta)
476 { return (0); }

478 /* ARGSUSED */
479 uint64_t
480 atomic_add_64_nv(volatile uint64_t *target, int64_t delta)
481 { return (0); }

483 /* ARGSUSED */
484 uint8_t
```

```
485 atomic_or_8_nv(volatile uint8_t *target, uint8_t value)
486 { return (0); }

488 /* ARGSUSED */
489 uchar_t
490 atomic_or_uchar_nv(volatile uchar_t *target, uchar_t value)
491 { return (0); }

493 /* ARGSUSED */
494 uint16_t
495 atomic_or_16_nv(volatile uint16_t *target, uint16_t value)
496 { return (0); }

498 /* ARGSUSED */
499 ushort_t
500 atomic_or_ushort_nv(volatile ushort_t *target, ushort_t value)
501 { return (0); }

503 /* ARGSUSED */
504 uint32_t
505 atomic_or_32_nv(volatile uint32_t *target, uint32_t value)
506 { return (0); }

508 /* ARGSUSED */
509 uint_t
510 atomic_or_uint_nv(volatile uint_t *target, uint_t value)
511 { return (0); }

513 /* ARGSUSED */
514 ulong_t
515 atomic_or_ulong_nv(volatile ulong_t *target, ulong_t value)
516 { return (0); }

518 /* ARGSUSED */
519 uint64_t
520 atomic_or_64_nv(volatile uint64_t *target, uint64_t value)
521 { return (0); }

523 /* ARGSUSED */
524 uint8_t
525 atomic_and_8_nv(volatile uint8_t *target, uint8_t value)
526 { return (0); }

528 /* ARGSUSED */
529 uchar_t
530 atomic_and_uchar_nv(volatile uchar_t *target, uchar_t value)
531 { return (0); }

533 /* ARGSUSED */
534 uint16_t
535 atomic_and_16_nv(volatile uint16_t *target, uint16_t value)
536 { return (0); }

538 /* ARGSUSED */
539 ushort_t
540 atomic_and_ushort_nv(volatile ushort_t *target, ushort_t value)
541 { return (0); }

543 /* ARGSUSED */
544 uint32_t
545 atomic_and_32_nv(volatile uint32_t *target, uint32_t value)
546 { return (0); }

548 /* ARGSUSED */
549 uint_t
550 atomic_and_uint_nv(volatile uint_t *target, uint_t value)
```



```
551 { return (0); }

553 /* ARGSUSED */
554 ulong_t
555 atomic_and_ulong_nv(volatile ulong_t *target, ulong_t value)
556 { return (0); }

558 /* ARGSUSED */
559 uint64_t
560 atomic_and_64_nv(volatile uint64_t *target, uint64_t value)
561 { return (0); }

563 /* ARGSUSED */
564 uint8_t
565 atomic_cas_8(volatile uint8_t *target, uint8_t cmp, uint8_t new)
566 { return (0); }

568 /* ARGSUSED */
569 uchar_t
570 atomic_cas_uchar(volatile uchar_t *target, uchar_t cmp, uchar_t new)
571 { return (0); }

573 /* ARGSUSED */
574 uint16_t
575 atomic_cas_16(volatile uint16_t *target, uint16_t cmp, uint16_t new)
576 { return (0); }

578 /* ARGSUSED */
579 ushort_t
580 atomic_cas_ushort(volatile ushort_t *target, ushort_t cmp, ushort_t new)
581 { return (0); }

583 /* ARGSUSED */
584 uint32_t
585 atomic_cas_32(volatile uint32_t *target, uint32_t cmp, uint32_t new)
586 { return (0); }

588 /* ARGSUSED */
589 uint_t
590 atomic_cas_uint(volatile uint_t *target, uint_t cmp, uint_t new)
591 { return (0); }

593 /* ARGSUSED */
594 ulong_t
595 atomic_cas_ulong(volatile ulong_t *target, ulong_t cmp, ulong_t new)
596 { return (0); }

598 /* ARGSUSED */
599 uint64_t
600 atomic_cas_uint64(volatile uint64_t *target, ulong_t cmp, uint64_t new)
601 { return (0); }

603 /* ARGSUSED */
604 void *
605 atomic_cas_ptr(volatile void *target, void *cmp, void *new)
606 { return (NULL); }

608 /* ARGSUSED */
609 uint8_t
610 atomic_swap_8(volatile uint8_t *target, uint8_t new)
611 { return (0); }

613 /* ARGSUSED */
614 uchar_t
615 atomic_swap_char(volatile uchar_t *target, uchar_t new)
616 { return (0); }
```

```
618 /* ARGSUSED */
619 uint16_t
620 atomic_swap_16(volatile uint16_t *target, uint16_t new)
621 { return (0); }

623 /* ARGSUSED */
624 ushort_t
625 atomic_swap_ushort(volatile ushort_t *target, ushort_t new)
626 { return (0); }

628 /* ARGSUSED */
629 uint32_t
630 atomic_swap_32(volatile uint32_t *target, uint32_t new)
631 { return (0); }

633 /* ARGSUSED */
634 uint_t
635 atomic_swap_uint(volatile uint_t *target, uint_t new)
636 { return (0); }

638 /* ARGSUSED */
639 uint64_t
640 atomic_swap_64(volatile uint64_t *target, uint64_t new)
641 { return (0); }

643 /* ARGSUSED */
644 void *
645 atomic_swap_ptr(volatile void *target, void *new)
646 { return (NULL); }

648 /* ARGSUSED */
649 ulong_t
650 atomic_swap_ulong(volatile ulong_t *target, ulong_t new)
651 { return (0); }

653 /* ARGSUSED */
654 int
655 atomic_set_long_excl(volatile ulong_t *target, uint_t value)
656 { return (0); }

658 /* ARGSUSED */
659 int
660 atomic_clear_long_excl(volatile ulong_t *target, uint_t value)
661 { return (0); }

663 void
664 fp_zero(void)
665 {}

667 uint64_t
668 gettick_npt(void)
669 { return (0); }

671 uint64_t
672 getstick_npt(void)
673 { return (0); }
```