```
**********************************************************
   44932 Fri May  8 18:04:17 2015
new/usr/src/uts/common/vm/seg_kmem.c
no need for bad-op segment op functions
The segment drivers have a number of bad-op functions that simply panic.
Keeping the function pointer NULL will accomplish the same thing in most
cases.  In other cases, keeping the function pointer NULL will result in
proper error code being returned.
**********************************************************
______unchanged_portion_omitted_

static void
segkmem_badop()
{
        panic("segkmem_badop");
}

#define SEGKMEM_BADOP(t)        (t(*)())segkmem_badop

/*ARGSUSED*/
static faultcode_t
segkmem_fault(struct hat *hat, struct seg *seg, caddr_t addr, size_t size,
        enum fault_type type, enum seg_rw rw)
{
        pgcnt_t npages;
        spgcnt_t pg;
        page_t *pp;
        struct vnode *vp = seg->s_data;

        ASSERT(RW_READ_HELD(&seg->s_as->a_lock));

        if (seg->s_as != &kas || size > seg->s_size ||
            addr < seg->s_base || addr + size > seg->s_base + seg->s_size)
                panic("segkmem_fault: bad args");

        /*
         * If it is one of segkp pages, call segkp_fault.
         */
        if (segkp_bitmap && seg == &kvseg &&
            BT_TEST(segkp_bitmap, btop((uintptr_t)(addr - seg->s_base))))
                return (segop_fault(hat, segkp, addr, size, type, rw));

        if (rw != S_READ && rw != S_WRITE && rw != S_OTHER)
                return (FC_NOSUPPORT);

        npages = btopr(size);

        switch (type) {
        case F_SOFTLOCK:        /* lock down already-loaded translations */
                for (pg = 0; pg < npages; pg++) {
                        pp = page_lookup(vp, (u_offset_t)(uintptr_t)addr,
                            SE_SHARED);
                        if (pp == NULL) {
                                /*
                                 * Hmm, no page. Does a kernel mapping
                                 * exist for it?
                                 */
                                if (!hat_probe(kas.a_hat, addr)) {
                                        addr -= PAGESIZE;
                                        while (--pg >= 0) {
                                                pp = page_find(vp, (u_offset_t)
                                                    (uintptr_t)addr);
                                                if (pp)
                                                        page_unlock(pp);
                                                addr -= PAGESIZE;
                                        }
```



```
                                        return (FC_NOMAP);
                                }
                        }
                        addr += PAGESIZE;
                }
                if (rw == S_OTHER)
                        hat_reserve(seg->s_as, addr, size);
                return (0);
        case F_SOFTUNLOCK:
                while (npages--) {
                        pp = page_find(vp, (u_offset_t)(uintptr_t)addr);
                        if (pp)
                                page_unlock(pp);
                        addr += PAGESIZE;
                }
                return (0);
        default:
                return (FC_NOSUPPORT);
        }
        /*NOTREACHED*/
}
______unchanged_portion_omitted_

/*
 * This is a dummy segkmem function overloaded to call segkp
 * when segkp is under the heap.
 */
/* ARGSUSED */
static int
segkmem_checkprot(struct seg *seg, caddr_t addr, size_t size, uint_t prot)
{
        ASSERT(RW_LOCK_HELD(&seg->s_as->a_lock));

        if (seg->s_as != &kas)
                panic("segkmem badop");
                segkmem_badop();

        /*
         * If it is one of segkp pages, call into segkp.
         */
        if (segkp_bitmap && seg == &kvseg &&
            BT_TEST(segkp_bitmap, btop((uintptr_t)(addr - seg->s_base))))
                return (segop_checkprot(segkp, addr, size, prot));

        panic("segkmem badop");
        segkmem_badop();
        return (0);
}

/*
 * This is a dummy segkmem function overloaded to call segkp
 * when segkp is under the heap.
 */
/* ARGSUSED */
static int
segkmem_kluster(struct seg *seg, caddr_t addr, ssize_t delta)
{
        ASSERT(RW_LOCK_HELD(&seg->s_as->a_lock));

        if (seg->s_as != &kas)
                panic("segkmem badop");
                segkmem_badop();

        /*
         * If it is one of segkp pages, call into segkp.
         */
```

```
563         if (segkp_bitmap && seg == &kvseg &&
564             BT_TEST(segkp_bitmap, btop((uintptr_t)(addr - seg->s_base))))
565                 return (segop_kluster(segkp, addr, delta));

567         panic("segkmem badop");
575         segkmem_badop();
568         return (0);
569 }
______unchanged_portion_omitted_

728 /*
729  * This is a dummy segkmem function overloaded to call segkp
730  * when segkp is under the heap.
731  */
732 /* ARGSUSED */
733 static int
734 segkmem_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp)
735 {
736         ASSERT(RW_LOCK_HELD(&seg->s_as->a_lock));

738         if (seg->s_as != &kas)
739                 panic("segkmem badop");
747                 segkmem_badop();

741         /*
742          * If it is one of segkp pages, call into segkp.
743          */
744         if (segkp_bitmap && seg == &kvseg &&
745             BT_TEST(segkp_bitmap, btop((uintptr_t)(addr - seg->s_base))))
746                 return (segop_getmemid(segkp, addr, memidp));

748         panic("segkmem badop");
756         segkmem_badop();
749         return (0);
750 }
______unchanged_portion_omitted_

768 static struct seg_ops segkmem_ops = {
777         .dup            = SEGKMEM_BADOP(int),
778         .unmap          = SEGKMEM_BADOP(int),
779         .free           = SEGKMEM_BADOP(void),
769         .fault          = segkmem_fault,
781         .faulta         = SEGKMEM_BADOP(faultcode_t),
770         .setprot        = segkmem_setprot,
771         .checkprot      = segkmem_checkprot,
772         .kluster        = segkmem_kluster,
785         .sync           = SEGKMEM_BADOP(int),
786         .incore         = SEGKMEM_BADOP(size_t),
787         .lockop         = SEGKMEM_BADOP(int),
788         .getprot        = SEGKMEM_BADOP(int),
789         .getoffset      = SEGKMEM_BADOP(u_offset_t),
790         .gettype        = SEGKMEM_BADOP(int),
791         .getvp          = SEGKMEM_BADOP(int),
792         .advise         = SEGKMEM_BADOP(int),
773         .dump           = segkmem_dump,
774         .pagelock       = segkmem_pagelock,
795         .setpagesize    = SEGKMEM_BADOP(int),
775         .getmemid       = segkmem_getmemid,
776         .getpolicy      = segkmem_getpolicy,
777         .capable        = segkmem_capable,
778         .inherit        = seg_inherit_notsup,
779 };
______unchanged_portion_omitted_
```

```
**********************************************************
   36313 Fri May  8 18:04:17 2015
new/usr/src/uts/common/vm/seg_kp.c
no need for bad-op segment op functions
The segment drivers have a number of bad-op functions that simply panic.
Keeping the function pointer NULL will accomplish the same thing in most
cases.  In other cases, keeping the function pointer NULL will result in
proper error code being returned.
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
  23  */

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

  28 /*
  29  * Portions of this source code were derived from Berkeley 4.3 BSD
  30  * under license from the Regents of the University of California.
  31  */

  33 /*
  34  * segkp is a segment driver that administers the allocation and deallocation
  35  * of pageable variable size chunks of kernel virtual address space. Each
  36  * allocated resource is page-aligned.
  37  *
  38  * The user may specify whether the resource should be initialized to 0,
  39  * include a redzone, or locked in memory.
  40  */

  42 #include <sys/types.h>
  43 #include <sys/t_lock.h>
  44 #include <sys/thread.h>
  45 #include <sys/param.h>
  46 #include <sys/errno.h>
  47 #include <sys/sysmacros.h>
  48 #include <sys/systm.h>
  49 #include <sys/buf.h>
  50 #include <sys/mman.h>
  51 #include <sys/vnode.h>
  52 #include <sys/cmn_err.h>
  53 #include <sys/swap.h>
  54 #include <sys/tuneable.h>
  55 #include <sys/kmem.h>
  56 #include <sys/vmem.h>
  57 #include <sys/cred.h>
```



```
  58 #include <sys/dumphdr.h>
  59 #include <sys/debug.h>
  60 #include <sys/vtrace.h>
  61 #include <sys/stack.h>
  62 #include <sys/atomic.h>
  63 #include <sys/archsystm.h>
  64 #include <sys/lgrp.h>

  66 #include <vm/as.h>
  67 #include <vm/seg.h>
  68 #include <vm/seg_kp.h>
  69 #include <vm/seg_kmem.h>
  70 #include <vm/anon.h>
  71 #include <vm/page.h>
  72 #include <vm/hat.h>
  73 #include <sys/bitmap.h>

  75 /*
  76  * Private seg op routines
  77  */
  78 static void     segkp_badop(void);
  78 static void     segkp_dump(struct seg *seg);
  79 static int      segkp_checkprot(struct seg *seg, caddr_t addr, size_t len,
  80                         uint_t prot);
  81 static int      segkp_kluster(struct seg *seg, caddr_t addr, ssize_t delta);
  82 static int      segkp_pagelock(struct seg *seg, caddr_t addr, size_t len,
  83                         struct page ***page, enum lock_type type,
  84                         enum seg_rw rw);
  85 static void     segkp_insert(struct seg *seg, struct segkp_data *kpd);
  86 static void     segkp_delete(struct seg *seg, struct segkp_data *kpd);
  87 static caddr_t  segkp_get_internal(struct seg *seg, size_t len, uint_t flags,
  88                         struct segkp_data **tkpd, struct anon_map *amp);
  89 static void     segkp_release_internal(struct seg *seg,
  90                         struct segkp_data *kpd, size_t len);
  91 static int      segkp_unlock(struct hat *hat, struct seg *seg, caddr_t vaddr,
  92                         size_t len, struct segkp_data *kpd, uint_t flags);
  93 static int      segkp_load(struct hat *hat, struct seg *seg, caddr_t vaddr,
  94                         size_t len, struct segkp_data *kpd, uint_t flags);
  95 static struct   segkp_data *segkp_find(struct seg *seg, caddr_t vaddr);
  96 static int      segkp_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp);
  97 static lgrp_mem_policy_info_t   *segkp_getpolicy(struct seg *seg,
  98     caddr_t addr);
  99 static int      segkp_capable(struct seg *seg, segcapability_t capability);

 101 /*
 102  * Lock used to protect the hash table(s) and caches.
 103  */
 104 static kmutex_t segkp_lock;

 106 /*
 107  * The segkp caches
 108  */
 109 static struct segkp_cache segkp_cache[SEGKP_MAX_CACHE];

 112 #define SEGKP_BADOP(t)  (t(*)())segkp_badop

 111 /*
 112  * When there are fewer than red_minavail bytes left on the stack,
 113  * segkp_map_red() will map in the redzone (if called).  5000 seems
 114  * to work reasonably well...
 115  */
 116 long            red_minavail = 5000;

 118 /*
 119  * will be set to 1 for 32 bit x86 systems only, in startup.c
 120  */
```

```
121 int     segkp_fromheap = 0;
122 ulong_t *segkp_bitmap;

124 /*
125  * If segkp_map_red() is called with the redzone already mapped and
126  * with less than RED_DEEP_THRESHOLD bytes available on the stack,
127  * then the stack situation has become quite serious;  if much more stack
128  * is consumed, we have the potential of scrogging the next thread/LWP
129  * structure.  To help debug the "can't happen" panics which may
130  * result from this condition, we record hrestime and the calling thread
131  * in red_deep_hires and red_deep_thread respectively.
132  */
133 #define RED_DEEP_THRESHOLD      2000

135 hrtime_t        red_deep_hires;
136 kthread_t       *red_deep_thread;

138 uint32_t        red_nmapped;
139 uint32_t        red_closest = UINT_MAX;
140 uint32_t        red_ndoubles;

142 pgcnt_t anon_segkp_pages_locked;        /* See vm/anon.h */
143 pgcnt_t anon_segkp_pages_resv;          /* anon reserved by seg_kp */

145 static struct   seg_ops segkp_ops = {
149         .dup            = SEGKP_BADOP(int),
150         .unmap          = SEGKP_BADOP(int),
151         .free           = SEGKP_BADOP(void),
146         .fault          = segkp_fault,
153         .faulta         = SEGKP_BADOP(faultcode_t),
154         .setprot        = SEGKP_BADOP(int),
147         .checkprot      = segkp_checkprot,
148         .kluster        = segkp_kluster,
157         .sync           = SEGKP_BADOP(int),
158         .incore         = SEGKP_BADOP(size_t),
159         .lockop         = SEGKP_BADOP(int),
160         .getprot        = SEGKP_BADOP(int),
161         .getoffset      = SEGKP_BADOP(u_offset_t),
162         .gettype        = SEGKP_BADOP(int),
163         .getvp          = SEGKP_BADOP(int),
164         .advise         = SEGKP_BADOP(int),
149         .dump           = segkp_dump,
150         .pagelock       = segkp_pagelock,
167         .setpagesize    = SEGKP_BADOP(int),
151         .getmemid       = segkp_getmemid,
152         .getpolicy      = segkp_getpolicy,
153         .capable        = segkp_capable,
154         .inherit        = seg_inherit_notsup,
155 };


175 static void
176 segkp_badop(void)
177 {
178         panic("segkp_badop");
179         /*NOTREACHED*/
180 }

158 static void segkpinit_mem_config(struct seg *);

160 static uint32_t segkp_indel;

162 /*
163  * Allocate the segment specific private data struct and fill it in
164  * with the per kp segment mutex, anon ptr. array and hash table.
165  */
```



```
166 int
167 segkp_create(struct seg *seg)
168 {
169         struct segkp_segdata *kpsd;
170         size_t  np;

172         ASSERT(seg != NULL && seg->s_as == &kas);
173         ASSERT(RW_WRITE_HELD(&seg->s_as->a_lock));

175         if (seg->s_size & PAGEOFFSET) {
176                 panic("Bad segkp size");
177                 /*NOTREACHED*/
178         }

180         kpsd = kmem_zalloc(sizeof (struct segkp_segdata), KM_SLEEP);

182         /*
183          * Allocate the virtual memory for segkp and initialize it
184          */
185         if (segkp_fromheap) {
186                 np = btop(kvseg.s_size);
187                 segkp_bitmap = kmem_zalloc(BT_SIZEOFMAP(np), KM_SLEEP);
188                 kpsd->kpsd_arena = vmem_create("segkp", NULL, 0, PAGESIZE,
189                     vmem_alloc, vmem_free, heap_arena, 5 * PAGESIZE, VM_SLEEP);
190         } else {
191                 segkp_bitmap = NULL;
192                 np = btop(seg->s_size);
193                 kpsd->kpsd_arena = vmem_create("segkp", seg->s_base,
194                     seg->s_size, PAGESIZE, NULL, NULL, NULL, 5 * PAGESIZE,
195                     VM_SLEEP);
196         }

198         kpsd->kpsd_anon = anon_create(np, ANON_SLEEP | ANON_ALLOC_FORCE);

200         kpsd->kpsd_hash = kmem_zalloc(SEGKP_HASHSZ * sizeof (struct segkp *),
201             KM_SLEEP);
202         seg->s_data = (void *)kpsd;
203         seg->s_ops = &segkp_ops;
204         segkpinit_mem_config(seg);
205         return (0);
206 }
```

unchanged_portion_omitted_

```
**********************************************************
    9619 Fri May  8 18:04:17 2015
new/usr/src/uts/common/vm/seg_kpm.c
no need for bad-op segment op functions
The segment drivers have a number of bad-op functions that simply panic.
Keeping the function pointer NULL will accomplish the same thing in most
cases.  In other cases, keeping the function pointer NULL will result in
proper error code being returned.
**********************************************************
```

```
   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, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */
  22 /*
  23  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */

  27 /*
  28  * Kernel Physical Mapping (kpm) segment driver (segkpm).
  29  *
  30  * This driver delivers along with the hat_kpm* interfaces an alternative
  31  * mechanism for kernel mappings within the 64-bit Solaris operating system,
  32  * which allows the mapping of all physical memory into the kernel address
  33  * space at once. This is feasible in 64 bit kernels, e.g. for Ultrasparc II
  34  * and beyond processors, since the available VA range is much larger than
  35  * possible physical memory. Momentarily all physical memory is supported,
  36  * that is represented by the list of memory segments (memsegs).
  37  *
  38  * Segkpm mappings have also very low overhead and large pages are used
  39  * (when possible) to minimize the TLB and TSB footprint. It is also
  40  * extentable for other than Sparc architectures (e.g. AMD64). Main
  41  * advantage is the avoidance of the TLB-shootdown X-calls, which are
  42  * normally needed when a kernel (global) mapping has to be removed.
  43  *
  44  * First example of a kernel facility that uses the segkpm mapping scheme
  45  * is seg_map, where it is used as an alternative to hat_memload().
  46  * See also hat layer for more information about the hat_kpm* routines.
  47  * The kpm facilty can be turned off at boot time (e.g. /etc/system).
  48  */

  50 #include <sys/types.h>
  51 #include <sys/param.h>
  52 #include <sys/sysmacros.h>
  53 #include <sys/systm.h>
  54 #include <sys/vnode.h>
  55 #include <sys/cmn_err.h>
  56 #include <sys/debug.h>
  57 #include <sys/thread.h>
```



```
  58 #include <sys/cpuvar.h>
  59 #include <sys/bitmap.h>
  60 #include <sys/atomic.h>
  61 #include <sys/lgrp.h>

  63 #include <vm/seg_kmem.h>
  64 #include <vm/seg_kpm.h>
  65 #include <vm/hat.h>
  66 #include <vm/as.h>
  67 #include <vm/seg.h>
  68 #include <vm/page.h>

  70 /*
  71  * Global kpm controls.
  72  * See also platform and mmu specific controls.
  73  *
  74  * kpm_enable -- global on/off switch for segkpm.
  75  * . Set by default on 64bit platforms that have kpm support.
  76  * . Will be disabled from platform layer if not supported.
  77  * . Can be disabled via /etc/system.
  78  *
  79  * kpm_smallpages -- use only regular/system pagesize for kpm mappings.
  80  * . Can be useful for critical debugging of kpm clients.
  81  * . Set to zero by default for platforms that support kpm large pages.
  82  *   The use of kpm large pages reduces the footprint of kpm meta data
  83  *   and has all the other advantages of using large pages (e.g TLB
  84  *   miss reduction).
  85  * . Set by default for platforms that don't support kpm large pages or
  86  *   where large pages cannot be used for other reasons (e.g. there are
  87  *   only few full associative TLB entries available for large pages).
  88  *
  89  * segmap_kpm -- separate on/off switch for segmap using segkpm:
  90  * . Set by default.
  91  * . Will be disabled when kpm_enable is zero.
  92  * . Will be disabled when MAXBSIZE != PAGESIZE.
  93  * . Can be disabled via /etc/system.
  94  *
  95  */
  96 int kpm_enable = 1;
  97 int kpm_smallpages = 0;
  98 int segmap_kpm = 1;

 100 /*
 101  * Private seg op routines.
 102  */
 103 faultcode_t segkpm_fault(struct hat *hat, struct seg *seg, caddr_t addr,
 104                         size_t len, enum fault_type type, enum seg_rw rw);
 105 static void     segkpm_dump(struct seg *);
 106 static int      segkpm_pagelock(struct seg *seg, caddr_t addr, size_t len,
 107                         struct page ***page, enum lock_type type,
 108                         enum seg_rw rw);
 106 static void     segkpm_badop(void);
 107 static int      segkpm_notsup(void);
 109 static int      segkpm_capable(struct seg *, segcapability_t);

 110 #define SEGKPM_BADOP(t) (t(*)())segkpm_badop
 111 #define SEGKPM_NOTSUP   (int(*)())segkpm_notsup

 111 static struct seg_ops segkpm_ops = {
 114         .dup            = SEGKPM_BADOP(int),
 115         .unmap          = SEGKPM_BADOP(int),
 116         .free           = SEGKPM_BADOP(void),
 112         .fault          = segkpm_fault,
 118         .faulta         = SEGKPM_BADOP(int),
 119         .setprot        = SEGKPM_BADOP(int),
 120         .checkprot      = SEGKPM_BADOP(int),
```

```
121         .kluster        = SEGKPM_BADOP(int),
122         .sync           = SEGKPM_BADOP(int),
123         .incore         = SEGKPM_BADOP(size_t),
124         .lockop         = SEGKPM_BADOP(int),
125         .getprot        = SEGKPM_BADOP(int),
126         .getoffset      = SEGKPM_BADOP(u_offset_t),
127         .gettype        = SEGKPM_BADOP(int),
128         .getvp          = SEGKPM_BADOP(int),
129         .advise         = SEGKPM_BADOP(int),
113         .dump           = segkpm_dump,
114         .pagelock       = segkpm_pagelock,
131         .pagelock       = SEGKPM_NOTSUP,
132         .setpagesize    = SEGKPM_BADOP(int),
133         .getmemid       = SEGKPM_BADOP(int),
134         .getpolicy      = SEGKPM_BADOP(lgrp_mem_policy_info_t *),
115         .capable        = segkpm_capable,
116         .inherit        = seg_inherit_notsup,
117 //#ifndef SEGKPM_SUPPORT
118 #if 0
119 #error FIXME: define nop
120         .dup            = nop,
121         .unmap          = nop,
122         .free           = nop,
123         .faulta         = nop,
124         .setprot        = nop,
125         .checkprot      = nop,
126         .kluster        = nop,
127         .sync           = nop,
128         .incore         = nop,
129         .lockop         = nop,
130         .getprot        = nop,
131         .getoffset      = nop,
132         .gettype        = nop,
133         .getvp          = nop,
134         .advise         = nop,
135         .setpagesize    = nop,
136         .getmemid       = nop,
137         .getpolicy      = nop,
138 #endif
139 #endif /* ! codereview */
140 };

142 /*
143  * kpm_pgsz and kpm_pgshft are set by platform layer.
144  */
145 size_t          kpm_pgsz;       /* kpm page size */
146 uint_t          kpm_pgshft;     /* kpm page shift */
147 u_offset_t      kpm_pgoff;      /* kpm page offset mask */
148 uint_t          kpmp2pshft;     /* kpm page to page shift */
149 pgcnt_t         kpmpnpgs;       /* how many pages per kpm page */


152 #ifdef  SEGKPM_SUPPORT

154 int
155 segkpm_create(struct seg *seg, void *argsp)
156 {
157         struct segkpm_data *skd;
158         struct segkpm_crargs *b = (struct segkpm_crargs *)argsp;
159         ushort_t *p;
160         int i, j;

162         ASSERT(seg->s_as && RW_WRITE_HELD(&seg->s_as->a_lock));
163         ASSERT(btokpmp(seg->s_size) >= 1 &&
164             kpmpageoff((uintptr_t)seg->s_base) == 0 &&
165             kpmpageoff((uintptr_t)seg->s_base + seg->s_size) == 0);
```



```
167         skd = kmem_zalloc(sizeof (struct segkpm_data), KM_SLEEP);

169         seg->s_data = (void *)skd;
170         seg->s_ops = &segkpm_ops;
171         skd->skd_prot = b->prot;

173         /*
174          * (1) Segkpm virtual addresses are based on physical adresses.
175          * From this and in opposite to other segment drivers it is
176          * often required to allocate a page first to be able to
177          * calculate the final segkpm virtual address.
178          * (2) Page  allocation is done by calling page_create_va(),
179          * one important input argument is a virtual address (also
180          * expressed by the "va" in the function name). This function
181          * is highly optimized to select the right page for an optimal
182          * processor and platform support (e.g. virtual addressed
183          * caches (VAC), physical addressed caches, NUMA).
184          *
185          * Because of (1) the approach is to generate a faked virtual
186          * address for calling page_create_va(). In order to exploit
187          * the abilities of (2), especially to utilize the cache
188          * hierarchy (3) and to avoid VAC alias conflicts (4) the
189          * selection has to be done carefully. For each virtual color
190          * a separate counter is provided (4). The count values are
191          * used for the utilization of all cache lines (3) and are
192          * corresponding to the cache bins.
193          */
194         skd->skd_nvcolors = b->nvcolors;

196         p = skd->skd_va_select =
197             kmem_zalloc(NCPU * b->nvcolors * sizeof (ushort_t), KM_SLEEP);

199         for (i = 0; i < NCPU; i++)
200                 for (j = 0; j < b->nvcolors; j++, p++)
201                         *p = j;

203         return (0);
204 }

206 /*
207  * This routine is called via a machine specific fault handling
208  * routine.
209  */
210 /* ARGSUSED */
211 faultcode_t
212 segkpm_fault(struct hat *hat, struct seg *seg, caddr_t addr, size_t len,
213         enum fault_type type, enum seg_rw rw)
214 {
215         ASSERT(seg->s_as && AS_LOCK_HELD(seg->s_as, &seg->s_as->a_lock));

217         switch (type) {
218         case F_INVAL:
219                 return (hat_kpm_fault(hat, addr));
220         case F_SOFTLOCK:
221         case F_SOFTUNLOCK:
222                 return (0);
223         default:
224                 return (FC_NOSUPPORT);
225         }
226         /*NOTREACHED*/
227 }

229 #define addr_to_vcolor(addr, vcolors) \
230         ((int)(((uintptr_t)(addr) & ((vcolors << PAGESHIFT) - 1)) >> PAGESHIFT))
```

```
232 /*
233  * Create a virtual address that can be used for invocations of
234  * page_create_va. Goal is to utilize the cache hierarchy (round
235  * robin bins) and to select the right color for virtual indexed
236  * caches. It isn't exact since we also increment the bin counter
237  * when the caller uses VOP_GETPAGE and gets a hit in the page
238  * cache, but we keep the bins turning for cache distribution
239  * (see also segkpm_create block comment).
240  */
241 caddr_t
242 segkpm_create_va(u_offset_t off)
243 {
244         int vcolor;
245         ushort_t *p;
246         struct segkpm_data *skd = (struct segkpm_data *)segkpm->s_data;
247         int nvcolors = skd->skd_nvcolors;
248         caddr_t va;
249
250         vcolor = (nvcolors > 1) ? addr_to_vcolor(off, nvcolors) : 0;
251         p = &skd->skd_va_select[(CPU->cpu_id * nvcolors) + vcolor];
252         va = (caddr_t)ptob(*p);
253
254         atomic_add_16(p, nvcolors);
255
256         return (va);
257 }
258
259 /*
260  * Unload mapping if the instance has an active kpm mapping.
261  */
262 void
263 segkpm_mapout_validkpme(struct kpme *kpme)
264 {
265         caddr_t vaddr;
266         page_t *pp;
267
268 retry:
269         if ((pp = kpme->kpe_page) == NULL) {
270                 return;
271         }
272
273         if (page_lock(pp, SE_SHARED, (kmutex_t *)NULL, P_RECLAIM) == 0)
274                 goto retry;
275
276         /*
277          * Check if segkpm mapping is not unloaded in the meantime
278          */
279         if (kpme->kpe_page == NULL) {
280                 page_unlock(pp);
281                 return;
282         }
283
284         vaddr = hat_kpm_page2va(pp, 1);
285         hat_kpm_mapout(pp, kpme, vaddr);
286         page_unlock(pp);
287 }

137 static void
138 segkpm_badop()
139 {
140         panic("segkpm_badop");
141 }

289 #else   /* SEGKPM_SUPPORT */

291 /* segkpm stubs */
```



```
293 /*ARGSUSED*/
294 int segkpm_create(struct seg *seg, void *argsp)
295 {
296         return (0);
297 }
148 int segkpm_create(struct seg *seg, void *argsp) { return (0); }

299 /* ARGSUSED */
300 faultcode_t
301 segkpm_fault(struct hat *hat, struct seg *seg, caddr_t addr, size_t len,
302         enum fault_type type, enum seg_rw rw)
303 {
304         return (0);
155         return ((faultcode_t)0);
305 }

307 /* ARGSUSED */
308 caddr_t segkpm_create_va(u_offset_t off)
309 {
310         return (NULL);
311 }
159 caddr_t segkpm_create_va(u_offset_t off) { return (NULL); }

313 /* ARGSUSED */
314 void segkpm_mapout_validkpme(struct kpme *kpme)
315 {
316 }
162 void segkpm_mapout_validkpme(struct kpme *kpme) {}

164 static void
165 segkpm_badop() {}

318 #endif  /* SEGKPM_SUPPORT */

320 /* ARGSUSED */
321 #endif /* ! codereview */
322 static int
323 segkpm_pagelock(struct seg *seg, caddr_t addr, size_t len,
324         struct page ***page, enum lock_type type, enum seg_rw rw)
169 segkpm_notsup()
325 {
326         return (ENOTSUP);
327 }

329 /*
330  * segkpm pages are not dumped, so we just return
331  */
332 /*ARGSUSED*/
333 static void
334 segkpm_dump(struct seg *seg)
335 {
336 }
180 {}

338 /*
339  * We claim to have no special capabilities.
340  */
341 /*ARGSUSED*/
342 static int
343 segkpm_capable(struct seg *seg, segcapability_t capability)
344 {
345         return (0);
346 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   57729 Fri May  8 18:04:17 2015
new/usr/src/uts/common/vm/seg_map.c
no need for bad-op segment op functions
The segment drivers have a number of bad-op functions that simply panic.
Keeping the function pointer NULL will accomplish the same thing in most
cases.  In other cases, keeping the function pointer NULL will result in
proper error code being returned.
**********************************************************
```

```
/*
 * 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.
 */

/*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T     */
/*        All Rights Reserved   */

/*
 * Portions of this source code were derived from Berkeley 4.3 BSD
 * under license from the Regents of the University of California.
 */

/*
 * VM - generic vnode mapping segment.
 *
 * The segmap driver is used only by the kernel to get faster (than seg_vn)
 * mappings [lower routine overhead; more persistent cache] to random
 * vnode/offsets.  Note than the kernel may (and does) use seg_vn as well.
 */

#include <sys/types.h>
#include <sys/t_lock.h>
#include <sys/param.h>
#include <sys/sysmacros.h>
#include <sys/buf.h>
#include <sys/systm.h>
#include <sys/vnode.h>
#include <sys/mman.h>
#include <sys/errno.h>
#include <sys/cred.h>
#include <sys/kmem.h>
#include <sys/vtrace.h>
#include <sys/cmn_err.h>
#include <sys/debug.h>
#include <sys/thread.h>
#include <sys/dumphdr.h>
```



```
#include <sys/bitmap.h>
#include <sys/lgrp.h>

#include <vm/seg_kmem.h>
#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_kpm.h>
#include <vm/seg_map.h>
#include <vm/page.h>
#include <vm/pvn.h>
#include <vm/rm.h>

/*
 * Private seg op routines.
 */
static void     segmap_free(struct seg *seg);
faultcode_t segmap_fault(struct hat *hat, struct seg *seg, caddr_t addr,
                        size_t len, enum fault_type type, enum seg_rw rw);
static faultcode_t segmap_faulta(struct seg *seg, caddr_t addr);
static int      segmap_checkprot(struct seg *seg, caddr_t addr, size_t len,
                        uint_t prot);
static int      segmap_kluster(struct seg *seg, caddr_t addr, ssize_t);
static int      segmap_getprot(struct seg *seg, caddr_t addr, size_t len,
                        uint_t *protv);
static u_offset_t       segmap_getoffset(struct seg *seg, caddr_t addr);
static int      segmap_gettype(struct seg *seg, caddr_t addr);
static int      segmap_getvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
static void     segmap_dump(struct seg *seg);
static int      segmap_pagelock(struct seg *seg, caddr_t addr, size_t len,
                        struct page ***ppp, enum lock_type type,
                        enum seg_rw rw);
static void     segmap_badop(void);
static int      segmap_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp);
static lgrp_mem_policy_info_t   *segmap_getpolicy(struct seg *seg,
    caddr_t addr);
static int      segmap_capable(struct seg *seg, segcapability_t capability);

/* segkpm support */
static caddr_t  segmap_pagecreate_kpm(struct seg *, vnode_t *, u_offset_t,
                        struct smap *, enum seg_rw);
struct smap     *get_smap_kpm(caddr_t, page_t **);

#define SEGMAP_BADOP(t) (t(*)())segmap_badop

static struct seg_ops segmap_ops = {
        .dup            = SEGMAP_BADOP(int),
        .unmap          = SEGMAP_BADOP(int),
        .free           = segmap_free,
        .fault          = segmap_fault,
        .faulta         = segmap_faulta,
        .setprot        = SEGMAP_BADOP(int),
        .checkprot      = segmap_checkprot,
        .kluster        = segmap_kluster,
        .sync           = SEGMAP_BADOP(int),
        .incore         = SEGMAP_BADOP(size_t),
        .lockop         = SEGMAP_BADOP(int),
        .getprot        = segmap_getprot,
        .getoffset      = segmap_getoffset,
        .gettype        = segmap_gettype,
        .getvp          = segmap_getvp,
        .advise         = SEGMAP_BADOP(int),
        .dump           = segmap_dump,
        .pagelock       = segmap_pagelock,
        .setpagesize    = SEGMAP_BADOP(int),
        .getmemid       = segmap_getmemid,
```

```
 113         .getpolicy      = segmap_getpolicy,
 114         .capable        = segmap_capable,
 115         .inherit        = seg_inherit_notsup,
 116 };
______unchanged_portion_omitted_

 886 /*
 887  * Check to see if it makes sense to do kluster/read ahead to
 888  * addr + delta relative to the mapping at addr.  We assume here
 889  * that delta is a signed PAGESIZE'd multiple (which can be negative).
 890  *
 891  * For segmap we always "approve" of this action from our standpoint.
 892  */
 893 /*ARGSUSED*/
 894 static int
 895 segmap_kluster(struct seg *seg, caddr_t addr, ssize_t delta)
 896 {
 897         return (0);
 909 }

 911 static void
 912 segmap_badop()
 913 {
 914         panic("segmap_badop");
 915         /*NOTREACHED*/
 898 }
______unchanged_portion_omitted_
```

```
**********************************************************
   82793 Fri May  8 18:04:18 2015
new/usr/src/uts/common/vm/seg_spt.c
no need for bad-op segment op functions
The segment drivers have a number of bad-op functions that simply panic.
Keeping the function pointer NULL will accomplish the same thing in most
cases.  In other cases, keeping the function pointer NULL will result in
proper error code being returned.
**********************************************************
```

```
/*
 * 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 (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
 */

#include <sys/param.h>
#include <sys/user.h>
#include <sys/mman.h>
#include <sys/kmem.h>
#include <sys/sysmacros.h>
#include <sys/cmn_err.h>
#include <sys/systm.h>
#include <sys/tuneable.h>
#include <vm/hat.h>
#include <vm/seg.h>
#include <vm/as.h>
#include <vm/anon.h>
#include <vm/page.h>
#include <sys/buf.h>
#include <sys/swap.h>
#include <sys/atomic.h>
#include <vm/seg_spt.h>
#include <sys/debug.h>
#include <sys/vtrace.h>
#include <sys/shm.h>
#include <sys/shm_impl.h>
#include <sys/lgrp.h>
#include <sys/vmsystm.h>
#include <sys/policy.h>
#include <sys/project.h>
#include <sys/tnf_probe.h>
#include <sys/zone.h>

#define SEGSPTADDR      (caddr_t)0x0

/*
 * # pages used for spt
 */
```



```
size_t  spt_used;

/*
 * segspt_minfree is the memory left for system after ISM
 * locked its pages; it is set up to 5% of availrmem in
 * sptcreate when ISM is created.  ISM should not use more
 * than ~90% of availrmem; if it does, then the performance
 * of the system may decrease. Machines with large memories may
 * be able to use up more memory for ISM so we set the default
 * segspt_minfree to 5% (which gives ISM max 95% of availrmem.
 * If somebody wants even more memory for ISM (risking hanging
 * the system) they can patch the segspt_minfree to smaller number.
 */
pgcnt_t segspt_minfree = 0;

static int segspt_create(struct seg *seg, caddr_t argsp);
static int segspt_unmap(struct seg *seg, caddr_t raddr, size_t ssize);
static void segspt_free(struct seg *seg);
static void segspt_free_pages(struct seg *seg, caddr_t addr, size_t len);
static lgrp_mem_policy_info_t *segspt_getpolicy(struct seg *seg, caddr_t addr);

static void
segspt_badop()
{
        panic("segspt_badop called");
        /*NOTREACHED*/
}

#define SEGSPT_BADOP(t) (t(*)())segspt_badop

struct seg_ops segspt_ops = {
        .dup            = SEGSPT_BADOP(int),
        .unmap          = segspt_unmap,
        .free           = segspt_free,
        .fault          = SEGSPT_BADOP(int),
        .faulta         = SEGSPT_BADOP(faultcode_t),
        .setprot        = SEGSPT_BADOP(int),
        .checkprot      = SEGSPT_BADOP(int),
        .kluster        = SEGSPT_BADOP(int),
        .sync           = SEGSPT_BADOP(int),
        .incore         = SEGSPT_BADOP(size_t),
        .lockop         = SEGSPT_BADOP(int),
        .getprot        = SEGSPT_BADOP(int),
        .getoffset      = SEGSPT_BADOP(u_offset_t),
        .gettype        = SEGSPT_BADOP(int),
        .getvp          = SEGSPT_BADOP(int),
        .advise         = SEGSPT_BADOP(int),
        .dump           = SEGSPT_BADOP(void),
        .pagelock       = SEGSPT_BADOP(int),
        .setpagesize    = SEGSPT_BADOP(int),
        .getmemid       = SEGSPT_BADOP(int),
        .getpolicy      = segspt_getpolicy,
        .capable        = SEGSPT_BADOP(int),
        .inherit        = seg_inherit_notsup,
};
______unchanged_portion_omitted_
```

```
**********************************************************
   12330 Fri May  8 18:04:18 2015
new/usr/src/uts/sparc/v9/vm/seg_nf.c
no need for bad-op segment op functions
The segment drivers have a number of bad-op functions that simply panic.
Keeping the function pointer NULL will accomplish the same thing in most
cases.  In other cases, keeping the function pointer NULL will result in
proper error code being returned.
**********************************************************
```

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

/* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
/* All Rights Reserved */

/*
 * Portions of this source code were derived from Berkeley 4.3 BSD
 * under license from the Regents of the University of California.
 */

/*
 * VM - segment for non-faulting loads.
 */

#include <sys/types.h>
#include <sys/t_lock.h>
#include <sys/param.h>
#include <sys/mman.h>
#include <sys/errno.h>
#include <sys/kmem.h>
#include <sys/cmn_err.h>
#include <sys/vnode.h>
#include <sys/proc.h>
#include <sys/conf.h>
#include <sys/debug.h>
#include <sys/archsystm.h>
#include <sys/lgrp.h>

#include <vm/page.h>
#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/vpage.h>
```



```
/*
 * Private seg op routines.
 */
static int      segnf_dup(struct seg *seg, struct seg *newseg);
static int      segnf_unmap(struct seg *seg, caddr_t addr, size_t len);
static void     segnf_free(struct seg *seg);
static faultcode_t segnf_nomap(void);
static int      segnf_setprot(struct seg *seg, caddr_t addr,
                    size_t len, uint_t prot);
static int      segnf_checkprot(struct seg *seg, caddr_t addr,
                    size_t len, uint_t prot);
static void     segnf_badop(void);
static int      segnf_nop(void);
static int      segnf_getprot(struct seg *seg, caddr_t addr,
                    size_t len, uint_t *protv);
static u_offset_t segnf_getoffset(struct seg *seg, caddr_t addr);
static int      segnf_gettype(struct seg *seg, caddr_t addr);
static int      segnf_getvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
static void     segnf_dump(struct seg *seg);
static int      segnf_pagelock(struct seg *seg, caddr_t addr, size_t len,
                    struct page ***ppp, enum lock_type type, enum seg_rw rw);
static int      segnf_setpagesize(struct seg *seg, caddr_t addr, size_t len,
                    uint_t szc);
static int      segnf_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp);
static lgrp_mem_policy_info_t   *segnf_getpolicy(struct seg *seg,
    caddr_t addr);


struct seg_ops segnf_ops = {
        .dup            = segnf_dup,
        .unmap          = segnf_unmap,
        .free           = segnf_free,
        .fault          = (faultcode_t (*)(struct hat *, struct seg *, caddr_t,
            size_t, enum fault_type, enum seg_rw))segnf_nomap,
        .faulta         = (faultcode_t (*)(struct seg *, caddr_t)) segnf_nomap,
        .setprot        = segnf_setprot,
        .checkprot      = segnf_checkprot,
        .kluster        = (int (*)())segnf_badop,
        .sync           = (int (*)(struct seg *, caddr_t, size_t, int, uint_t))
                segnf_nop,
        .incore         = (size_t (*)(struct seg *, caddr_t, size_t, char *))
                segnf_nop,
        .lockop         = (int (*)(struct seg *, caddr_t, size_t, int, int,
            ulong_t *, size_t))segnf_nop,
        .getprot        = segnf_getprot,
        .getoffset      = segnf_getoffset,
        .gettype        = segnf_gettype,
        .getvp          = segnf_getvp,
        .advise         = (int (*)(struct seg *, caddr_t, size_t, uint_t))
                segnf_nop,
        .dump           = segnf_dump,
        .pagelock       = segnf_pagelock,
        .setpagesize    = segnf_setpagesize,
        .getmemid       = segnf_getmemid,
        .getpolicy      = segnf_getpolicy,
};
```

_____*unchanged_portion_omitted_*

```
/* ARGSUSED */
static int
segnf_checkprot(struct seg *seg, caddr_t addr, size_t len, uint_t prot)
{
        uint_t sprot;
        ASSERT(seg->s_as && AS_LOCK_HELD(seg->s_as, &seg->s_as->a_lock));

        sprot = seg->s_as == &kas ?  PROT_READ : PROT_READ|PROT_USER;
```

```
 399         return ((prot & sprot) == prot ? 0 : EACCES);
 402 }

 404 static void
 405 segnf_badop(void)
 406 {
 407         panic("segnf_badop");
 408         /*NOTREACHED*/
 400 }
```

_____***unchanged_portion_omitted_***