```
**********************************************************
   10192 Tue Nov 24 09:34:52 2015
new/usr/src/uts/common/vm/seg.h
6146 seg_inherit_notsup is redundant
**********************************************************
______unchanged_portion_omitted_

 147 #ifdef _KERNEL

 149 /*
 150  * Generic segment operations
 151  */
 152 extern  void    seg_init(void);
 153 extern  struct  seg *seg_alloc(struct as *as, caddr_t base, size_t size);
 154 extern  int     seg_attach(struct as *as, caddr_t base, size_t size,
 155                         struct seg *seg);
 156 extern  void    seg_unmap(struct seg *seg);
 157 extern  void    seg_free(struct seg *seg);

 159 /*
 160  * functions for pagelock cache support
 161  */
 162 typedef int (*seg_preclaim_cbfunc_t)(void *, caddr_t, size_t,
 163     struct page **, enum seg_rw, int);

 165 extern  struct  page **seg_plookup(struct seg *seg, struct anon_map *amp,
 166     caddr_t addr, size_t len, enum seg_rw rw, uint_t flags);
 167 extern  void    seg_pinactive(struct seg *seg, struct anon_map *amp,
 168     caddr_t addr, size_t len, struct page **pp, enum seg_rw rw,
 169     uint_t flags, seg_preclaim_cbfunc_t callback);

 171 extern  void    seg_ppurge(struct seg *seg, struct anon_map *amp,
 172     uint_t flags);
 173 extern  void    seg_ppurge_wiredpp(struct page **pp);

 175 extern  int     seg_pinsert_check(struct seg *seg, struct anon_map *amp,
 176     caddr_t addr, size_t len, uint_t flags);
 177 extern  int     seg_pinsert(struct seg *seg, struct anon_map *amp,
 178     caddr_t addr, size_t len, size_t wlen, struct page **pp, enum seg_rw rw,
 179     uint_t flags, seg_preclaim_cbfunc_t callback);

 181 extern  void    seg_pasync_thread(void);
 182 extern  void    seg_preap(void);
 183 extern  int     seg_p_disable(void);
 184 extern  void    seg_p_enable(void);

 186 extern  segadvstat_t    segadvstat;

 188 /*
 189  * Flags for pagelock cache support.
 190  * Flags argument is passed as uint_t to pcache routines.  upper 16 bits of
 191  * the flags argument are reserved for alignment page shift when SEGP_PSHIFT
 192  * is set.
 193  */
 194 #define SEGP_FORCE_WIRED        0x1     /* skip check against seg_pwindow */
 195 #define SEGP_AMP                0x2     /* anon map's pcache entry */
 196 #define SEGP_PSHIFT             0x4     /* addr pgsz shift for hash function */

 198 /*
 199  * Return values for seg_pinsert and seg_pinsert_check functions.
 200  */
 201 #define SEGP_SUCCESS            0       /* seg_pinsert() succeeded */
 202 #define SEGP_FAIL               1       /* seg_pinsert() failed */

 204 /* Page status bits for segop_incore */
 205 #define SEG_PAGE_INCORE         0x01    /* VA has a page backing it */
```



```
 206 #define SEG_PAGE_LOCKED         0x02    /* VA has a page that is locked */
 207 #define SEG_PAGE_HASCOW         0x04    /* VA has a page with a copy-on-write */
 208 #define SEG_PAGE_SOFTLOCK       0x08    /* VA has a page with softlock held */
 209 #define SEG_PAGE_VNODEBACKED    0x10    /* Segment is backed by a vnode */
 210 #define SEG_PAGE_ANON           0x20    /* VA has an anonymous page */
 211 #define SEG_PAGE_VNODE          0x40    /* VA has a vnode page backing it */

 213 #define seg_page(seg, addr) \
 214         (((uintptr_t)((addr) - (seg)->s_base)) >> PAGESHIFT)

 216 #define seg_pages(seg) \
 217         (((uintptr_t)((seg)->s_size + PAGEOFFSET)) >> PAGESHIFT)

 219 #define IE_NOMEM        -1      /* internal to seg layer */
 220 #define IE_RETRY        -2      /* internal to seg layer */
 221 #define IE_REATTACH     -3      /* internal to seg layer */

 223 /* Values for segop_inherit */
 224 #define SEGP_INH_ZERO   0x01

 226 int seg_inherit_notsup(struct seg *, caddr_t, size_t, uint_t);

 226 /* Delay/retry factors for seg_p_mem_config_pre_del */
 227 #define SEGP_PREDEL_DELAY_FACTOR        4
 228 /*
 229  * As a workaround to being unable to purge the pagelock
 230  * cache during a DR delete memory operation, we use
 231  * a stall threshold that is twice the maximum seen
 232  * during testing.  This workaround will be removed
 233  * when a suitable fix is found.
 234  */
 235 #define SEGP_STALL_SECONDS      25
 236 #define SEGP_STALL_THRESHOLD \
 237         (SEGP_STALL_SECONDS * SEGP_PREDEL_DELAY_FACTOR)

 239 #ifdef VMDEBUG

 241 uint_t  seg_page(struct seg *, caddr_t);
 242 uint_t  seg_pages(struct seg *);

 244 #endif  /* VMDEBUG */

 246 boolean_t       seg_can_change_zones(struct seg *);
 247 size_t          seg_swresv(struct seg *);

 249 /* segop wrappers */
 250 extern int segop_dup(struct seg *, struct seg *);
 251 extern int segop_unmap(struct seg *, caddr_t, size_t);
 252 extern void segop_free(struct seg *);
 253 extern faultcode_t segop_fault(struct hat *, struct seg *, caddr_t, size_t,
 254     enum fault_type, enum seg_rw);
 255 extern faultcode_t segop_faulta(struct seg *, caddr_t);
 256 extern int segop_setprot(struct seg *, caddr_t, size_t, uint_t);
 257 extern int segop_checkprot(struct seg *, caddr_t, size_t, uint_t);
 258 extern int segop_kluster(struct seg *, caddr_t, ssize_t);
 259 extern size_t segop_swapout(struct seg *);
 260 extern int segop_sync(struct seg *, caddr_t, size_t, int, uint_t);
 261 extern size_t segop_incore(struct seg *, caddr_t, size_t, char *);
 262 extern int segop_lockop(struct seg *, caddr_t, size_t, int, int, ulong_t *,
 263     size_t);
 264 extern int segop_getprot(struct seg *, caddr_t, size_t, uint_t *);
 265 extern u_offset_t segop_getoffset(struct seg *, caddr_t);
 266 extern int segop_gettype(struct seg *, caddr_t);
 267 extern int segop_getvp(struct seg *, caddr_t, struct vnode **);
 268 extern int segop_advise(struct seg *, caddr_t, size_t, uint_t);
 269 extern void segop_dump(struct seg *);
```

```
 270 extern int segop_pagelock(struct seg *, caddr_t, size_t, struct page ***,
 271     enum lock_type, enum seg_rw);
 272 extern int segop_setpagesize(struct seg *, caddr_t, size_t, uint_t);
 273 extern int segop_getmemid(struct seg *, caddr_t, memid_t *);
 274 extern struct lgrp_mem_policy_info *segop_getpolicy(struct seg *, caddr_t);
 275 extern int segop_capable(struct seg *, segcapability_t);
 276 extern int segop_inherit(struct seg *, caddr_t, size_t, uint_t);

 278 #endif  /* _KERNEL */

 280 #ifdef  __cplusplus
 281 }
```

unchanged_portion_omitted_

```
**********************************************************
  114075 Tue Nov 24 09:34:52 2015
new/usr/src/uts/common/vm/seg_dev.c
6146 seg_inherit_notsup is redundant
**********************************************************
```

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

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

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * VM - segment of a mapped device.
 *
 * This segment driver is used when mapping character special devices.
 */

#include <sys/types.h>
#include <sys/t_lock.h>
#include <sys/sysmacros.h>
#include <sys/vtrace.h>
#include <sys/systm.h>
#include <sys/vmsystm.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/ddidevmap.h>
#include <sys/ddi_implfuncs.h>
```



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

#include <vm/page.h>
#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_dev.h>
#include <vm/seg_kp.h>
#include <vm/seg_kmem.h>
#include <vm/vpage.h>

#include <sys/sunddi.h>
#include <sys/esunddi.h>
#include <sys/fs/snode.h>


#if DEBUG
int segdev_debug;
#define DEBUGF(level, args) { if (segdev_debug >= (level)) cmn_err args; }
#else
#define DEBUGF(level, args)
#endif

/* Default timeout for devmap context management */
#define CTX_TIMEOUT_VALUE 0

#define HOLD_DHP_LOCK(dhp)  if (dhp->dh_flags & DEVMAP_ALLOW_REMAP) \
                        { mutex_enter(&dhp->dh_lock); }

#define RELE_DHP_LOCK(dhp) if (dhp->dh_flags & DEVMAP_ALLOW_REMAP) \
                        { mutex_exit(&dhp->dh_lock); }

#define round_down_p2(a, s)     ((a) & ~((s) - 1))
#define round_up_p2(a, s)       (((a) + (s) - 1) & ~((s) - 1))

/*
 * VA_PA_ALIGNED checks to see if both VA and PA are on pgsize boundary
 * VA_PA_PGSIZE_ALIGNED check to see if VA is aligned with PA w.r.t. pgsize
 */
#define VA_PA_ALIGNED(uvaddr, paddr, pgsize)            \
        (((uvaddr | paddr) & (pgsize - 1)) == 0)
#define VA_PA_PGSIZE_ALIGNED(uvaddr, paddr, pgsize)     \
        (((uvaddr ^ paddr) & (pgsize - 1)) == 0)

#define vpgtob(n)       ((n) * sizeof (struct vpage))   /* For brevity */

#define VTOCVP(vp)      (VTOS(vp)->s_commonvp)  /* we "know" it's an snode */

static struct devmap_ctx *devmapctx_list = NULL;
static struct devmap_softlock *devmap_slist = NULL;

/*
 * mutex, vnode and page for the page of zeros we use for the trash mappings.
 * One trash page is allocated on the first ddi_umem_setup call that uses it
 * XXX Eventually, we may want to combine this with what segnf does when all
 * hat layers implement HAT_NOFAULT.
 *
 * The trash page is used when the backing store for a userland mapping is
 * removed but the application semantics do not take kindly to a SIGBUS.
 * In that scenario, the applications pages are mapped to some dummy page
 * which returns garbage on read and writes go into a common place.
 * (Perfect for NO_FAULT semantics)
 * The device driver is responsible to communicating to the app with some
 * other mechanism that such remapping has happened and the app should take
 * corrective action.
 * We can also use an anonymous memory page as there is no requirement to
```

```
 128  * keep the page locked, however this complicates the fault code. RFE.
 129  */
 130 static struct vnode trashvp;
 131 static struct page *trashpp;

 133 /* Non-pageable kernel memory is allocated from the umem_np_arena. */
 134 static vmem_t *umem_np_arena;

 136 /* Set the cookie to a value we know will never be a valid umem_cookie */
 137 #define DEVMAP_DEVMEM_COOKIE    ((ddi_umem_cookie_t)0x1)

 139 /*
 140  * Macros to check if type of devmap handle
 141  */
 142 #define cookie_is_devmem(c)     \
 143         ((c) == (struct ddi_umem_cookie *)DEVMAP_DEVMEM_COOKIE)

 145 #define cookie_is_pmem(c)       \
 146         ((c) == (struct ddi_umem_cookie *)DEVMAP_PMEM_COOKIE)

 148 #define cookie_is_kpmem(c)      (!cookie_is_devmem(c) && !cookie_is_pmem(c) &&\
 149         ((c)->type == KMEM_PAGEABLE))

 151 #define dhp_is_devmem(dhp)      \
 152         (cookie_is_devmem((struct ddi_umem_cookie *)((dhp)->dh_cookie)))

 154 #define dhp_is_pmem(dhp)        \
 155         (cookie_is_pmem((struct ddi_umem_cookie *)((dhp)->dh_cookie)))

 157 #define dhp_is_kpmem(dhp)       \
 158         (cookie_is_kpmem((struct ddi_umem_cookie *)((dhp)->dh_cookie)))

 160 /*
 161  * Private seg op routines.
 162  */
 163 static int      segdev_dup(struct seg *, struct seg *);
 164 static int      segdev_unmap(struct seg *, caddr_t, size_t);
 165 static void     segdev_free(struct seg *);
 166 static faultcode_t segdev_fault(struct hat *, struct seg *, caddr_t, size_t,
 167                     enum fault_type, enum seg_rw);
 168 static faultcode_t segdev_faulta(struct seg *, caddr_t);
 169 static int      segdev_setprot(struct seg *, caddr_t, size_t, uint_t);
 170 static int      segdev_checkprot(struct seg *, caddr_t, size_t, uint_t);
 171 static void     segdev_badop(void);
 172 static int      segdev_sync(struct seg *, caddr_t, size_t, int, uint_t);
 173 static size_t   segdev_incore(struct seg *, caddr_t, size_t, char *);
 174 static int      segdev_lockop(struct seg *, caddr_t, size_t, int, int,
 175                     ulong_t *, size_t);
 176 static int      segdev_getprot(struct seg *, caddr_t, size_t, uint_t *);
 177 static u_offset_t       segdev_getoffset(struct seg *, caddr_t);
 178 static int      segdev_gettype(struct seg *, caddr_t);
 179 static int      segdev_getvp(struct seg *, caddr_t, struct vnode **);
 180 static int      segdev_advise(struct seg *, caddr_t, size_t, uint_t);
 181 static void     segdev_dump(struct seg *);
 182 static int      segdev_pagelock(struct seg *, caddr_t, size_t,
 183                     struct page ***, enum lock_type, enum seg_rw);
 184 static int      segdev_setpagesize(struct seg *, caddr_t, size_t, uint_t);
 185 static int      segdev_getmemid(struct seg *, caddr_t, memid_t *);
 186 static lgrp_mem_policy_info_t   *segdev_getpolicy(struct seg *, caddr_t);
 187 static int      segdev_capable(struct seg *, segcapability_t);

 189 /*
 190  * XXX  this struct is used by rootnex_map_fault to identify
 191  *      the segment it has been passed. So if you make it
 192  *      "static" you'll need to fix rootnex_map_fault.
 193  */
```



```
 194 struct seg_ops segdev_ops = {
 195         .dup            = segdev_dup,
 196         .unmap          = segdev_unmap,
 197         .free           = segdev_free,
 198         .fault          = segdev_fault,
 199         .faulta         = segdev_faulta,
 200         .setprot        = segdev_setprot,
 201         .checkprot      = segdev_checkprot,
 202         .kluster        = (int (*)())segdev_badop,
 203         .sync           = segdev_sync,
 204         .incore         = segdev_incore,
 205         .lockop         = segdev_lockop,
 206         .getprot        = segdev_getprot,
 207         .getoffset      = segdev_getoffset,
 208         .gettype        = segdev_gettype,
 209         .getvp          = segdev_getvp,
 210         .advise         = segdev_advise,
 211         .dump           = segdev_dump,
 212         .pagelock       = segdev_pagelock,
 213         .setpagesize    = segdev_setpagesize,
 214         .getmemid       = segdev_getmemid,
 215         .getpolicy      = segdev_getpolicy,
 216         .capable        = segdev_capable,
 217         .inherit        = seg_inherit_notsup,
 217 };
```

_____***unchanged_portion_omitted_***

****
45431 Tue Nov 24 09:34:52 2015
new/usr/src/uts/common/vm/seg_kmem.c
6146 seg_inherit_notsup is redundant
****

_____***unchanged_portion_omitted_***

```
 776 static struct seg_ops segkmem_ops = {
 777         .dup            = SEGKMEM_BADOP(int),
 778         .unmap          = SEGKMEM_BADOP(int),
 779         .free           = SEGKMEM_BADOP(void),
 780         .fault          = segkmem_fault,
 781         .faulta         = SEGKMEM_BADOP(faultcode_t),
 782         .setprot        = segkmem_setprot,
 783         .checkprot      = segkmem_checkprot,
 784         .kluster        = segkmem_kluster,
 785         .swapout        = SEGKMEM_BADOP(size_t),
 786         .sync           = SEGKMEM_BADOP(int),
 787         .incore         = SEGKMEM_BADOP(size_t),
 788         .lockop         = SEGKMEM_BADOP(int),
 789         .getprot        = SEGKMEM_BADOP(int),
 790         .getoffset      = SEGKMEM_BADOP(u_offset_t),
 791         .gettype        = SEGKMEM_BADOP(int),
 792         .getvp          = SEGKMEM_BADOP(int),
 793         .advise         = SEGKMEM_BADOP(int),
 794         .dump           = segkmem_dump,
 795         .pagelock       = segkmem_pagelock,
 796         .setpagesize    = SEGKMEM_BADOP(int),
 797         .getmemid       = segkmem_getmemid,
 798         .getpolicy      = segkmem_getpolicy,
 799         .capable        = segkmem_capable,
 800         .inherit        = seg_inherit_notsup,
 800 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   37135 Tue Nov 24 09:34:52 2015
new/usr/src/uts/common/vm/seg_kp.c
6146 seg_inherit_notsup is redundant
**********************************************************
```

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

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

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

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

 114 /*
 115  * When there are fewer than red_minavail bytes left on the stack,
 116  * segkp_map_red() will map in the redzone (if called).  5000 seems
 117  * to work reasonably well...
 118  */
 119 long            red_minavail = 5000;

 121 /*
 122  * will be set to 1 for 32 bit x86 systems only, in startup.c
 123  */
 124 int     segkp_fromheap = 0;
 125 ulong_t *segkp_bitmap;

 127 /*
```

```
128  * If segkp_map_red() is called with the redzone already mapped and
129  * with less than RED_DEEP_THRESHOLD bytes available on the stack,
130  * then the stack situation has become quite serious;  if much more stack
131  * is consumed, we have the potential of scrogging the next thread/LWP
132  * structure.  To help debug the "can't happen" panics which may
133  * result from this condition, we record hrestime and the calling thread
134  * in red_deep_hires and red_deep_thread respectively.
135  */
136 #define RED_DEEP_THRESHOLD      2000

138 hrtime_t        red_deep_hires;
139 kthread_t       *red_deep_thread;

141 uint32_t        red_nmapped;
142 uint32_t        red_closest = UINT_MAX;
143 uint32_t        red_ndoubles;

145 pgcnt_t anon_segkp_pages_locked;        /* See vm/anon.h */
146 pgcnt_t anon_segkp_pages_resv;          /* anon reserved by seg_kp */

148 static struct   seg_ops segkp_ops = {
149         .dup            = SEGKP_BADOP(int),
150         .unmap          = SEGKP_BADOP(int),
151         .free           = SEGKP_BADOP(void),
152         .fault          = segkp_fault,
153         .faulta         = SEGKP_BADOP(faultcode_t),
154         .setprot        = SEGKP_BADOP(int),
155         .checkprot      = segkp_checkprot,
156         .kluster        = segkp_kluster,
157         .swapout        = SEGKP_BADOP(size_t),
158         .sync           = SEGKP_BADOP(int),
159         .incore         = SEGKP_BADOP(size_t),
160         .lockop         = SEGKP_BADOP(int),
161         .getprot        = SEGKP_BADOP(int),
162         .getoffset      = SEGKP_BADOP(u_offset_t),
163         .gettype        = SEGKP_BADOP(int),
164         .getvp          = SEGKP_BADOP(int),
165         .advise         = SEGKP_BADOP(int),
166         .dump           = segkp_dump,
167         .pagelock       = segkp_pagelock,
168         .setpagesize    = SEGKP_BADOP(int),
169         .getmemid       = segkp_getmemid,
170         .getpolicy      = segkp_getpolicy,
171         .capable        = segkp_capable,
172         .inherit        = seg_inherit_notsup,
172 };
_____unchanged_portion_omitted_
```

```
**********************************************************
    9840 Tue Nov 24 09:34:52 2015
new/usr/src/uts/common/vm/seg_kpm.c
6146 seg_inherit_notsup is redundant
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#include <sys/types.h>
#include <sys/param.h>
#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/vnode.h>
#include <sys/cmn_err.h>
#include <sys/debug.h>
#include <sys/thread.h>
#include <sys/cpuvar.h>
#include <sys/bitmap.h>
#include <sys/atomic.h>
#include <sys/lgrp.h>
```



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

/*
 * Global kpm controls.
 * See also platform and mmu specific controls.
 *
 * kpm_enable -- global on/off switch for segkpm.
 * . Set by default on 64bit platforms that have kpm support.
 * . Will be disabled from platform layer if not supported.
 * . Can be disabled via /etc/system.
 *
 * kpm_smallpages -- use only regular/system pagesize for kpm mappings.
 * . Can be useful for critical debugging of kpm clients.
 * . Set to zero by default for platforms that support kpm large pages.
 *   The use of kpm large pages reduces the footprint of kpm meta data
 *   and has all the other advantages of using large pages (e.g TLB
 *   miss reduction).
 * . Set by default for platforms that don't support kpm large pages or
 *   where large pages cannot be used for other reasons (e.g. there are
 *   only few full associative TLB entries available for large pages).
 *
 * segmap_kpm -- separate on/off switch for segmap using segkpm:
 * . Set by default.
 * . Will be disabled when kpm_enable is zero.
 * . Will be disabled when MAXBSIZE != PAGESIZE.
 * . Can be disabled via /etc/system.
 *
 */
int kpm_enable = 1;
int kpm_smallpages = 0;
int segmap_kpm = 1;

/*
 * Private seg op routines.
 */
faultcode_t segkpm_fault(struct hat *hat, struct seg *seg, caddr_t addr,
                         size_t len, enum fault_type type, enum seg_rw rw);
static void     segkpm_dump(struct seg *);
static void     segkpm_badop(void);
static int      segkpm_notsup(void);
static int      segkpm_capable(struct seg *, segcapability_t);

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

static struct seg_ops segkpm_ops = {
        .dup            = SEGKPM_BADOP(int),
        .unmap          = SEGKPM_BADOP(int),
        .free           = SEGKPM_BADOP(void),
        .fault          = segkpm_fault,
        .faulta         = SEGKPM_BADOP(int),
        .setprot        = SEGKPM_BADOP(int),
        .checkprot      = SEGKPM_BADOP(int),
        .kluster        = SEGKPM_BADOP(int),
        .swapout        = SEGKPM_BADOP(size_t),
        .sync           = SEGKPM_BADOP(int),
        .incore         = SEGKPM_BADOP(size_t),
        .lockop         = SEGKPM_BADOP(int),
        .getprot        = SEGKPM_BADOP(int),
        .getoffset      = SEGKPM_BADOP(u_offset_t),
```

```
 128         .gettype        = SEGKPM_BADOP(int),
 129         .getvp          = SEGKPM_BADOP(int),
 130         .advise         = SEGKPM_BADOP(int),
 131         .dump           = segkpm_dump,
 132         .pagelock       = SEGKPM_NOTSUP,
 133         .setpagesize    = SEGKPM_BADOP(int),
 134         .getmemid       = SEGKPM_BADOP(int),
 135         .getpolicy      = SEGKPM_BADOP(lgrp_mem_policy_info_t *),
 136         .capable        = segkpm_capable,
 137         .inherit        = seg_inherit_notsup,
 137 };
```

_____unchanged_portion_omitted_

```
**********************************************************
   58130 Tue Nov 24 09:34:53 2015
new/usr/src/uts/common/vm/seg_map.c
6146 seg_inherit_notsup is redundant
**********************************************************
```

```
/*
 * 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,
        .swapout        = SEGMAP_BADOP(size_t),
        .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,
        .getpolicy      = segmap_getpolicy,
        .capable        = segmap_capable,
        .inherit        = seg_inherit_notsup,
```

```
 127 };
______unchanged_portion_omitted_
```

```
**********************************************************
   84116 Tue Nov 24 09:34:53 2015
new/usr/src/uts/common/vm/seg_spt.c
6146 seg_inherit_notsup is redundant
**********************************************************
______unchanged_portion_omitted_

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

  88 struct seg_ops segspt_ops = {
  89         .dup            = SEGSPT_BADOP(int),
  90         .unmap          = segspt_unmap,
  91         .free           = segspt_free,
  92         .fault          = SEGSPT_BADOP(int),
  93         .faulta         = SEGSPT_BADOP(faultcode_t),
  94         .setprot        = SEGSPT_BADOP(int),
  95         .checkprot      = SEGSPT_BADOP(int),
  96         .kluster        = SEGSPT_BADOP(int),
  97         .swapout        = SEGSPT_BADOP(size_t),
  98         .sync           = SEGSPT_BADOP(int),
  99         .incore         = SEGSPT_BADOP(size_t),
 100         .lockop         = SEGSPT_BADOP(int),
 101         .getprot        = SEGSPT_BADOP(int),
 102         .getoffset      = SEGSPT_BADOP(u_offset_t),
 103         .gettype        = SEGSPT_BADOP(int),
 104         .getvp          = SEGSPT_BADOP(int),
 105         .advise         = SEGSPT_BADOP(int),
 106         .dump           = SEGSPT_BADOP(void),
 107         .pagelock       = SEGSPT_BADOP(int),
 108         .setpagesize    = SEGSPT_BADOP(int),
 109         .getmemid       = SEGSPT_BADOP(int),
 110         .getpolicy      = segspt_getpolicy,
 111         .capable        = SEGSPT_BADOP(int),
 112         .inherit        = seg_inherit_notsup,
 112 };

 114 static int segspt_shmdup(struct seg *seg, struct seg *newseg);
 115 static int segspt_shmunmap(struct seg *seg, caddr_t raddr, size_t ssize);
 116 static void segspt_shmfree(struct seg *seg);
 117 static faultcode_t segspt_shmfault(struct hat *hat, struct seg *seg,
 118                 caddr_t addr, size_t len, enum fault_type type, enum seg_rw rw);
 119 static faultcode_t segspt_shmfaulta(struct seg *seg, caddr_t addr);
 120 static int segspt_shmsetprot(register struct seg *seg, register caddr_t addr,
 121                         register size_t len, register uint_t prot);
 122 static int segspt_shmcheckprot(struct seg *seg, caddr_t addr, size_t size,
 123                         uint_t prot);
 124 static int      segspt_shmkluster(struct seg *seg, caddr_t addr, ssize_t delta);
 125 static size_t   segspt_shmswapout(struct seg *seg);
 126 static size_t segspt_shmincore(struct seg *seg, caddr_t addr, size_t len,
 127                         register char *vec);
 128 static int segspt_shmsync(struct seg *seg, register caddr_t addr, size_t len,
 129                         int attr, uint_t flags);
 130 static int segspt_shmlockop(struct seg *seg, caddr_t addr, size_t len,
 131                         int attr, int op, ulong_t *lockmap, size_t pos);
 132 static int segspt_shmgetprot(struct seg *seg, caddr_t addr, size_t len,
 133                         uint_t *protv);
 134 static u_offset_t segspt_shmgetoffset(struct seg *seg, caddr_t addr);
 135 static int segspt_shmgettype(struct seg *seg, caddr_t addr);
 136 static int segspt_shmgetvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
 137 static int segspt_shmadvise(struct seg *seg, caddr_t addr, size_t len,
 138                         uint_t behav);
 139 static void segspt_shmdump(struct seg *seg);
 140 static int segspt_shmpagelock(struct seg *, caddr_t, size_t,
 141                         struct page ***, enum lock_type, enum seg_rw);
 142 static int segspt_shmsetpgsz(struct seg *, caddr_t, size_t, uint_t);
 143 static int segspt_shmgetmemid(struct seg *, caddr_t, memid_t *);
```



```
 144 static lgrp_mem_policy_info_t *segspt_shmgetpolicy(struct seg *, caddr_t);
 145 static int segspt_shmcapable(struct seg *, segcapability_t);

 147 struct seg_ops segspt_shmops = {
 148         .dup            = segspt_shmdup,
 149         .unmap          = segspt_shmunmap,
 150         .free           = segspt_shmfree,
 151         .fault          = segspt_shmfault,
 152         .faulta         = segspt_shmfaulta,
 153         .setprot        = segspt_shmsetprot,
 154         .checkprot      = segspt_shmcheckprot,
 155         .kluster        = segspt_shmkluster,
 156         .swapout        = segspt_shmswapout,
 157         .sync           = segspt_shmsync,
 158         .incore         = segspt_shmincore,
 159         .lockop         = segspt_shmlockop,
 160         .getprot        = segspt_shmgetprot,
 161         .getoffset      = segspt_shmgetoffset,
 162         .gettype        = segspt_shmgettype,
 163         .getvp          = segspt_shmgetvp,
 164         .advise         = segspt_shmadvise,
 165         .dump           = segspt_shmdump,
 166         .pagelock       = segspt_shmpagelock,
 167         .setpagesize    = segspt_shmsetpgsz,
 168         .getmemid       = segspt_shmgetmemid,
 169         .getpolicy      = segspt_shmgetpolicy,
 170         .capable        = segspt_shmcapable,
 172         .inherit        = seg_inherit_notsup,
 171 };
______unchanged_portion_omitted_
```

```
**********************************************************
   94617 Tue Nov 24 09:34:53 2015
new/usr/src/uts/common/vm/vm_as.c
6146 seg_inherit_notsup is redundant
**********************************************************
______unchanged_portion_omitted_

2307 /*
2308  * Cache control operations over the interval [addr, addr + size) in
2309  * address space "as".
2310  */
2311 /*ARGSUSED*/
2312 int
2313 as_ctl(struct as *as, caddr_t addr, size_t size, int func, int attr,
2314     uintptr_t arg, ulong_t *lock_map, size_t pos)
2315 {
2316         struct seg *seg;        /* working segment */
2317         caddr_t raddr;          /* rounded down addr */
2318         caddr_t initraddr;      /* saved initial rounded down addr */
2319         size_t rsize;           /* rounded up size */
2320         size_t initrsize;       /* saved initial rounded up size */
2321         size_t ssize;           /* size of seg */
2322         int error = 0;                  /* result */
2323         size_t mlock_size;      /* size of bitmap */
2324         ulong_t *mlock_map;     /* pointer to bitmap used */
2325                                 /* to represent the locked */
2326                                 /* pages. */
2327 retry:
2328         if (error == IE_RETRY)
2329                 AS_LOCK_ENTER(as, &as->a_lock, RW_WRITER);
2330         else
2331                 AS_LOCK_ENTER(as, &as->a_lock, RW_READER);

2333         /*
2334          * If these are address space lock/unlock operations, loop over
2335          * all segments in the address space, as appropriate.
2336          */
2337         if (func == MC_LOCKAS) {
2338                 size_t npages, idx;
2339                 size_t rlen = 0;        /* rounded as length */

2341                 idx = pos;

2343                 if (arg & MCL_FUTURE) {
2344                         mutex_enter(&as->a_contents);
2345                         AS_SETPGLCK(as);
2346                         mutex_exit(&as->a_contents);
2347                 }
2348                 if ((arg & MCL_CURRENT) == 0) {
2349                         AS_LOCK_EXIT(as, &as->a_lock);
2350                         return (0);
2351                 }

2353                 seg = AS_SEGFIRST(as);
2354                 if (seg == NULL) {
2355                         AS_LOCK_EXIT(as, &as->a_lock);
2356                         return (0);
2357                 }

2359                 do {
2360                         raddr = (caddr_t)((uintptr_t)seg->s_base &
2361                             (uintptr_t)PAGEMASK);
2362                         rlen += (((uintptr_t)(seg->s_base + seg->s_size) +
2363                             PAGEOFFSET) & PAGEMASK) - (uintptr_t)raddr;
2364                 } while ((seg = AS_SEGNEXT(as, seg)) != NULL);
```



```
2366                 mlock_size = BT_BITOUL(btopr(rlen));
2367                 if ((mlock_map = (ulong_t *)kmem_zalloc(mlock_size *
2368                     sizeof (ulong_t), KM_NOSLEEP)) == NULL) {
2369                         AS_LOCK_EXIT(as, &as->a_lock);
2370                         return (EAGAIN);
2371                 }

2373                 for (seg = AS_SEGFIRST(as); seg; seg = AS_SEGNEXT(as, seg)) {
2374                         error = segop_lockop(seg, seg->s_base,
2375                             seg->s_size, attr, MC_LOCK, mlock_map, pos);
2376                         if (error != 0)
2377                                 break;
2378                         pos += seg_pages(seg);
2379                 }

2381                 if (error) {
2382                         for (seg = AS_SEGFIRST(as); seg != NULL;
2383                             seg = AS_SEGNEXT(as, seg)) {

2385                                 raddr = (caddr_t)((uintptr_t)seg->s_base &
2386                                     (uintptr_t)PAGEMASK);
2387                                 npages = seg_pages(seg);
2388                                 as_segunlock(seg, raddr, attr, mlock_map,
2389                                     idx, npages);
2390                                 idx += npages;
2391                         }
2392                 }

2394                 kmem_free(mlock_map, mlock_size * sizeof (ulong_t));
2395                 AS_LOCK_EXIT(as, &as->a_lock);
2396                 goto lockerr;
2397         } else if (func == MC_UNLOCKAS) {
2398                 mutex_enter(&as->a_contents);
2399                 AS_CLRPGLCK(as);
2400                 mutex_exit(&as->a_contents);

2402                 for (seg = AS_SEGFIRST(as); seg; seg = AS_SEGNEXT(as, seg)) {
2403                         error = segop_lockop(seg, seg->s_base,
2404                             seg->s_size, attr, MC_UNLOCK, NULL, 0);
2405                         if (error != 0)
2406                                 break;
2407                 }

2409                 AS_LOCK_EXIT(as, &as->a_lock);
2410                 goto lockerr;
2411         }

2413         /*
2414          * Normalize addresses and sizes.
2415          */
2416         initraddr = raddr = (caddr_t)((uintptr_t)addr & (uintptr_t)PAGEMASK);
2417         initrsize = rsize = (((size_t)(addr + size) + PAGEOFFSET) & PAGEMASK) -
2418             (size_t)raddr;

2420         if (raddr + rsize < raddr) {            /* check for wraparound */
2421                 AS_LOCK_EXIT(as, &as->a_lock);
2422                 return (ENOMEM);
2423         }

2425         /*
2426          * Get initial segment.
2427          */
2428         if ((seg = as_segat(as, raddr)) == NULL) {
2429                 AS_LOCK_EXIT(as, &as->a_lock);
2430                 return (ENOMEM);
2431         }
```

```
2433         if (func == MC_LOCK) {
2434                 mlock_size = BT_BITOUL(btopr(rsize));
2435                 if ((mlock_map = (ulong_t *)kmem_zalloc(mlock_size *
2436                     sizeof (ulong_t), KM_NOSLEEP)) == NULL) {
2437                                 AS_LOCK_EXIT(as, &as->a_lock);
2438                                 return (EAGAIN);
2439                 }
2440         }

2442         /*
2443          * Loop over all segments.  If a hole in the address range is
2444          * discovered, then fail.  For each segment, perform the appropriate
2445          * control operation.
2446          */
2447         while (rsize != 0) {

2449                 /*
2450                  * Make sure there's no hole, calculate the portion
2451                  * of the next segment to be operated over.
2452                  */
2453                 if (raddr >= seg->s_base + seg->s_size) {
2454                         seg = AS_SEGNEXT(as, seg);
2455                         if (seg == NULL || raddr != seg->s_base) {
2456                                 if (func == MC_LOCK) {
2457                                         as_unlockerr(as, attr, mlock_map,
2458                                             initraddr, initrsize - rsize);
2459                                         kmem_free(mlock_map,
2460                                             mlock_size * sizeof (ulong_t));
2461                                 }
2462                                 AS_LOCK_EXIT(as, &as->a_lock);
2463                                 return (ENOMEM);
2464                         }
2465                 }
2466                 if ((raddr + rsize) > (seg->s_base + seg->s_size))
2467                         ssize = seg->s_base + seg->s_size - raddr;
2468                 else
2469                         ssize = rsize;

2471                 /*
2472                  * Dispatch on specific function.
2473                  */
2474                 switch (func) {

2476                 /*
2477                  * Synchronize cached data from mappings with backing
2478                  * objects.
2479                  */
2480                 case MC_SYNC:
2481                         if (error = segop_sync(seg, raddr, ssize,
2482                             attr, (uint_t)arg)) {
2483                                 AS_LOCK_EXIT(as, &as->a_lock);
2484                                 return (error);
2485                         }
2486                         break;

2488                 /*
2489                  * Lock pages in memory.
2490                  */
2491                 case MC_LOCK:
2492                         if (error = segop_lockop(seg, raddr, ssize,
2493                             attr, func, mlock_map, pos)) {
2494                                 as_unlockerr(as, attr, mlock_map, initraddr,
2495                                     initrsize - rsize + ssize);
2496                                 kmem_free(mlock_map, mlock_size *
2497                                     sizeof (ulong_t));
```



```
2498                                 AS_LOCK_EXIT(as, &as->a_lock);
2499                                 goto lockerr;
2500                         }
2501                         break;

2503                 /*
2504                  * Unlock mapped pages.
2505                  */
2506                 case MC_UNLOCK:
2507                         (void) segop_lockop(seg, raddr, ssize, attr, func,
2508                             (ulong_t *)NULL, (size_t)NULL);
2509                         break;

2511                 /*
2512                  * Store VM advise for mapped pages in segment layer.
2513                  */
2514                 case MC_ADVISE:
2515                         error = segop_advise(seg, raddr, ssize, (uint_t)arg);

2517                         /*
2518                          * Check for regular errors and special retry error
2519                          */
2520                         if (error) {
2521                                 if (error == IE_RETRY) {
2522                                         /*
2523                                          * Need to acquire writers lock, so
2524                                          * have to drop readers lock and start
2525                                          * all over again
2526                                          */
2527                                         AS_LOCK_EXIT(as, &as->a_lock);
2528                                         goto retry;
2529                                 } else if (error == IE_REATTACH) {
2530                                         /*
2531                                          * Find segment for current address
2532                                          * because current segment just got
2533                                          * split or concatenated
2534                                          */
2535                                         seg = as_segat(as, raddr);
2536                                         if (seg == NULL) {
2537                                                 AS_LOCK_EXIT(as, &as->a_lock);
2538                                                 return (ENOMEM);
2539                                         }
2540                                 } else {
2541                                         /*
2542                                          * Regular error
2543                                          */
2544                                         AS_LOCK_EXIT(as, &as->a_lock);
2545                                         return (error);
2546                                 }
2547                         }
2548                         break;

2550                 case MC_INHERIT_ZERO:
2551                         error = segop_inherit(seg, raddr, ssize, SEGP_INH_ZERO);
2551                         if (seg->s_ops->inherit == NULL) {
2552                                 error = ENOTSUP;
2553                         } else {
2554                                 error = segop_inherit(seg, raddr, ssize,
2555                                     SEGP_INH_ZERO);
2556                         }
2552                         if (error != 0) {
2553                                 AS_LOCK_EXIT(as, &as->a_lock);
2554                                 return (error);
2555                         }
2556                         break;
```

```
2558                 /*
2559                  * Can't happen.
2560                  */
2561                 default:
2562                         panic("as_ctl: bad operation %d", func);
2563                         /*NOTREACHED*/
2564                 }

2566                 rsize -= ssize;
2567                 raddr += ssize;
2568         }

2570         if (func == MC_LOCK)
2571                 kmem_free(mlock_map, mlock_size * sizeof (ulong_t));
2572         AS_LOCK_EXIT(as, &as->a_lock);
2573         return (0);
2574 lockerr:

2576         /*
2577          * If the lower levels returned EDEADLK for a segment lockop,
2578          * it means that we should retry the operation.  Let's wait
2579          * a bit also to let the deadlock causing condition clear.
2580          * This is part of a gross hack to work around a design flaw
2581          * in the ufs/sds logging code and should go away when the
2582          * logging code is re-designed to fix the problem. See bug
2583          * 4125102 for details of the problem.
2584          */
2585         if (error == EDEADLK) {
2586                 delay(deadlk_wait);
2587                 error = 0;
2588                 goto retry;
2589         }
2590         return (error);
2591 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   54373 Tue Nov 24 09:34:53 2015
new/usr/src/uts/common/vm/vm_seg.c
6146 seg_inherit_notsup is redundant
**********************************************************
______unchanged_portion_omitted_

1857 /*
1858  * General not supported function for segop_inherit
1859  */
1860 /* ARGSUSED */
1861 int
1862 seg_inherit_notsup(struct seg *seg, caddr_t addr, size_t len, uint_t op)
1863 {
1864         return (ENOTSUP);
1865 }

1867 /*
1858  * segop wrappers
1859  */
1860 int
1861 segop_dup(struct seg *seg, struct seg *new)
1862 {
1863         return (seg->s_ops->dup(seg, new));
1864 }
______unchanged_portion_omitted_
```

****
16964 Tue Nov 24 09:34:53 2015
new/usr/src/uts/i86xpv/vm/seg_mf.c
6146 seg_inherit_notsup is redundant
****
_____***unchanged_portion_omitted_***

```
 760 static struct seg_ops segmf_ops = {
 761         .dup            = segmf_dup,
 762         .unmap          = segmf_unmap,
 763         .free           = segmf_free,
 764         .fault          = segmf_fault,
 765         .faulta         = segmf_faulta,
 766         .setprot        = segmf_setprot,
 767         .checkprot      = segmf_checkprot,
 768         .kluster        = segmf_kluster,
 769         .sync           = segmf_sync,
 770         .incore         = segmf_incore,
 771         .lockop         = segmf_lockop,
 772         .getprot        = segmf_getprot,
 773         .getoffset      = segmf_getoffset,
 774         .gettype        = segmf_gettype,
 775         .getvp          = segmf_getvp,
 776         .advise         = segmf_advise,
 777         .dump           = segmf_dump,
 778         .pagelock       = segmf_pagelock,
 779         .setpagesize    = segmf_setpagesize,
 780         .getmemid       = segmf_getmemid,
 781         .getpolicy      = segmf_getpolicy,
 782         .capable        = segmf_capable,
 783         .inherit        = seg_inherit_notsup,
 783 };
```

_____***unchanged_portion_omitted_***