```
**********************************************************
  113260 Tue Nov 24 09:35:08 2015
new/usr/src/uts/common/vm/seg_dev.c
6152 use NULL dump segop as a shorthand for no-op
**********************************************************
```

```
/*
 * 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 *);
 181 static int      segdev_pagelock(struct seg *, caddr_t, size_t,
 182                     struct page ***, enum lock_type, enum seg_rw);
 183 static int      segdev_getmemid(struct seg *, caddr_t, memid_t *);

 185 /*
 186  * XXX  this struct is used by rootnex_map_fault to identify
 187  *      the segment it has been passed. So if you make it
 188  *      "static" you'll need to fix rootnex_map_fault.
 189  */
 190 struct seg_ops segdev_ops = {
 191         .dup            = segdev_dup,
 192         .unmap          = segdev_unmap,
```



```
 193         .free           = segdev_free,
 194         .fault          = segdev_fault,
 195         .faulta         = segdev_faulta,
 196         .setprot        = segdev_setprot,
 197         .checkprot      = segdev_checkprot,
 198         .kluster        = (int (*)())segdev_badop,
 199         .sync           = segdev_sync,
 200         .incore         = segdev_incore,
 201         .lockop         = segdev_lockop,
 202         .getprot        = segdev_getprot,
 203         .getoffset      = segdev_getoffset,
 204         .gettype        = segdev_gettype,
 205         .getvp          = segdev_getvp,
 206         .advise         = segdev_advise,
 208         .dump           = segdev_dump,
 207         .pagelock       = segdev_pagelock,
 208         .getmemid       = segdev_getmemid,
 209 };
_____unchanged_portion_omitted_

2375 /*
2376  * segdev pages are not dumped, so we just return
2377  */
2378 /*ARGSUSED*/
2379 static void
2380 segdev_dump(struct seg *seg)
2381 {}

2373 /*
2374  * ddi_segmap_setup:    Used by drivers who wish specify mapping attributes
2375  *                      for a segment.  Called from a drivers segmap(9E)
2376  *                      routine.
2377  */
2378 /*ARGSUSED*/
2379 int
2380 ddi_segmap_setup(dev_t dev, off_t offset, struct as *as, caddr_t *addrp,
2381     off_t len, uint_t prot, uint_t maxprot, uint_t flags, cred_t *cred,
2382     ddi_device_acc_attr_t *accattrp, uint_t rnumber)
2383 {
2384         struct segdev_crargs dev_a;
2385         int (*mapfunc)(dev_t dev, off_t off, int prot);
2386         uint_t hat_attr;
2387         pfn_t pfn;
2388         int     error, i;

2390         TRACE_0(TR_FAC_DEVMAP, TR_DEVMAP_SEGMAP_SETUP,
2391             "ddi_segmap_setup:start");

2393         if ((mapfunc = devopsp[getmajor(dev)]->devo_cb_ops->cb_mmap) == nodev)
2394                 return (ENODEV);

2396         /*
2397          * Character devices that support the d_mmap
2398          * interface can only be mmap'ed shared.
2399          */
2400         if ((flags & MAP_TYPE) != MAP_SHARED)
2401                 return (EINVAL);

2403         /*
2404          * Check that this region is indeed mappable on this platform.
2405          * Use the mapping function.
2406          */
2407         if (ddi_device_mapping_check(dev, accattrp, rnumber, &hat_attr) == -1)
2408                 return (ENXIO);

2410         /*
```

```
2411          * Check to ensure that the entire range is
2412          * legal and we are not trying to map in
2413          * more than the device will let us.
2414          */
2415         for (i = 0; i < len; i += PAGESIZE) {
2416                 if (i == 0) {
2417                         /*
2418                          * Save the pfn at offset here. This pfn will be
2419                          * used later to get user address.
2420                          */
2421                         if ((pfn = (pfn_t)cdev_mmap(mapfunc, dev, offset,
2422                             maxprot)) == PFN_INVALID)
2423                                 return (ENXIO);
2424                 } else {
2425                         if (cdev_mmap(mapfunc, dev, offset + i, maxprot) ==
2426                             PFN_INVALID)
2427                                 return (ENXIO);
2428                 }
2429         }

2431         as_rangelock(as);
2432         /* Pick an address w/o worrying about any vac alignment constraints. */
2433         error = choose_addr(as, addrp, len, ptob(pfn), ADDR_NOVACALIGN, flags);
2434         if (error != 0) {
2435                 as_rangeunlock(as);
2436                 return (error);
2437         }

2439         dev_a.mapfunc = mapfunc;
2440         dev_a.dev = dev;
2441         dev_a.offset = (offset_t)offset;
2442         dev_a.type = flags & MAP_TYPE;
2443         dev_a.prot = (uchar_t)prot;
2444         dev_a.maxprot = (uchar_t)maxprot;
2445         dev_a.hat_attr = hat_attr;
2446         dev_a.hat_flags = 0;
2447         dev_a.devmap_data = NULL;

2449         error = as_map(as, *addrp, len, segdev_create, &dev_a);
2450         as_rangeunlock(as);
2451         return (error);

2453 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9423 Tue Nov 24 09:35:08 2015
new/usr/src/uts/common/vm/seg_kpm.c
6152 use NULL dump segop as a shorthand for no-op
**********************************************************
```

```
/*
 * 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);

#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),
        .gettype        = SEGKPM_BADOP(int),
```

```
 127         .getvp          = SEGKPM_BADOP(int),
 128         .advise         = SEGKPM_BADOP(int),
 130         .dump           = segkpm_dump,
 129         .pagelock       = SEGKPM_NOTSUP,
 130         .setpagesize    = SEGKPM_BADOP(int),
 131         .getmemid       = SEGKPM_BADOP(int),
 132         .getpolicy      = SEGKPM_BADOP(lgrp_mem_policy_info_t *),
 133 };
______unchanged_portion_omitted_

 322 /*
 323  * segkpm pages are not dumped, so we just return
 324  */
 325 /*ARGSUSED*/
 326 static void
 327 segkpm_dump(struct seg *seg)
 328 {}
```

```
**********************************************************
   83531 Tue Nov 24 09:35:08 2015
new/usr/src/uts/common/vm/seg_spt.c
6152 use NULL dump segop as a shorthand for no-op
**********************************************************
______unchanged_portion_omitted_

 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);
 139 static int segspt_shmpagelock(struct seg *, caddr_t, size_t,
 140                         struct page ***, enum lock_type, enum seg_rw);
 141 static int segspt_shmgetmemid(struct seg *, caddr_t, memid_t *);
 142 static lgrp_mem_policy_info_t *segspt_shmgetpolicy(struct seg *, caddr_t);

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

2853 /*
2854  * We need to wait for pending IO to complete to a DISM segment in order for
2855  * pages to get kicked out of the seg_pcache.  120 seconds should be more
```



```
2856  * than enough time to wait.
2857  */
2858 static clock_t spt_pcache_wait = 120;

2860 /*ARGSUSED*/
2861 static int
2862 segspt_shmadvise(struct seg *seg, caddr_t addr, size_t len, uint_t behav)
2863 {
2864         struct shm_data *shmd = (struct shm_data *)seg->s_data;
2865         struct spt_data *sptd = (struct spt_data *)shmd->shm_sptseg->s_data;
2866         struct anon_map *amp;
2867         pgcnt_t pg_idx;
2868         ushort_t gen;
2869         clock_t end_lbolt;
2870         int writer;
2871         page_t **ppa;

2873         ASSERT(seg->s_as && AS_LOCK_HELD(seg->s_as, &seg->s_as->a_lock));

2875         if (behav == MADV_FREE) {
2876                 if ((sptd->spt_flags & SHM_PAGEABLE) == 0)
2877                         return (0);

2879                 amp = sptd->spt_amp;
2880                 pg_idx = seg_page(seg, addr);

2882                 mutex_enter(&sptd->spt_lock);
2883                 if ((ppa = sptd->spt_ppa) == NULL) {
2884                         mutex_exit(&sptd->spt_lock);
2885                         ANON_LOCK_ENTER(&amp->a_rwlock, RW_READER);
2886                         anon_disclaim(amp, pg_idx, len);
2887                         ANON_LOCK_EXIT(&amp->a_rwlock);
2888                         return (0);
2889                 }

2891                 sptd->spt_flags |= DISM_PPA_CHANGED;
2892                 gen = sptd->spt_gen;

2894                 mutex_exit(&sptd->spt_lock);

2896                 /*
2897                  * Purge all DISM cached pages
2898                  */
2899                 seg_ppurge_wiredpp(ppa);

2901                 /*
2902                  * Drop the AS_LOCK so that other threads can grab it
2903                  * in the as_pageunlock path and hopefully get the segment
2904                  * kicked out of the seg_pcache.  We bump the shm_softlockcnt
2905                  * to keep this segment resident.
2906                  */
2907                 writer = AS_WRITE_HELD(seg->s_as, &seg->s_as->a_lock);
2908                 atomic_inc_ulong((ulong_t *)(&(shmd->shm_softlockcnt)));
2909                 AS_LOCK_EXIT(seg->s_as, &seg->s_as->a_lock);

2911                 mutex_enter(&sptd->spt_lock);

2913                 end_lbolt = ddi_get_lbolt() + (hz * spt_pcache_wait);

2915                 /*
2916                  * Try to wait for pages to get kicked out of the seg_pcache.
2917                  */
2918                 while (sptd->spt_gen == gen &&
2919                     (sptd->spt_flags & DISM_PPA_CHANGED) &&
2920                     ddi_get_lbolt() < end_lbolt) {
2921                         if (!cv_timedwait_sig(&sptd->spt_cv,
```

```
2922                            &sptd->spt_lock, end_lbolt)) {
2923                                break;
2924                        }
2925                }

2927                mutex_exit(&sptd->spt_lock);

2929                /* Regrab the AS_LOCK and release our hold on the segment */
2930                AS_LOCK_ENTER(seg->s_as, &seg->s_as->a_lock,
2931                    writer ? RW_WRITER : RW_READER);
2932                atomic_dec_ulong((ulong_t *)(&(shmd->shm_softlockcnt)));
2933                if (shmd->shm_softlockcnt <= 0) {
2934                        if (AS_ISUNMAPWAIT(seg->s_as)) {
2935                                mutex_enter(&seg->s_as->a_contents);
2936                                if (AS_ISUNMAPWAIT(seg->s_as)) {
2937                                        AS_CLRUNMAPWAIT(seg->s_as);
2938                                        cv_broadcast(&seg->s_as->a_cv);
2939                                }
2940                                mutex_exit(&seg->s_as->a_contents);
2941                        }
2942                }

2944                ANON_LOCK_ENTER(&amp->a_rwlock, RW_READER);
2945                anon_disclaim(amp, pg_idx, len);
2946                ANON_LOCK_EXIT(&amp->a_rwlock);
2947        } else if (lgrp_optimizations() && (behav == MADV_ACCESS_LWP ||
2948            behav == MADV_ACCESS_MANY || behav == MADV_ACCESS_DEFAULT)) {
2949                int                     already_set;
2950                ulong_t                 anon_index;
2951                lgrp_mem_policy_t       policy;
2952                caddr_t                 shm_addr;
2953                size_t                  share_size;
2954                size_t                  size;
2955                struct seg              *sptseg = shmd->shm_sptseg;
2956                caddr_t                 sptseg_addr;

2958                /*
2959                 * Align address and length to page size of underlying segment
2960                 */
2961                share_size = page_get_pagesize(shmd->shm_sptseg->s_szc);
2962                shm_addr = (caddr_t)P2ALIGN((uintptr_t)(addr), share_size);
2963                size = P2ROUNDUP((uintptr_t)(((addr + len) - shm_addr)),
2964                    share_size);

2966                amp = shmd->shm_amp;
2967                anon_index = seg_page(seg, shm_addr);

2969                /*
2970                 * And now we may have to adjust size downward if we have
2971                 * exceeded the realsize of the segment or initial anon
2972                 * allocations.
2973                 */
2974                sptseg_addr = sptseg->s_base + ptob(anon_index);
2975                if ((sptseg_addr + size) >
2976                    (sptseg->s_base + sptd->spt_realsize))
2977                        size = (sptseg->s_base + sptd->spt_realsize) -
2978                            sptseg_addr;

2980                /*
2981                 * Set memory allocation policy for this segment
2982                 */
2983                policy = lgrp_madv_to_policy(behav, len, MAP_SHARED);
2984                already_set = lgrp_shm_policy_set(policy, amp, anon_index,
2985                    NULL, 0, len);

2987                /*
```



```
2988                 * If random memory allocation policy set already,
2989                 * don't bother reapplying it.
2990                 */
2991                if (already_set && !LGRP_MEM_POLICY_REAPPLICABLE(policy))
2992                        return (0);

2994                /*
2995                 * Mark any existing pages in the given range for
2996                 * migration, flushing the I/O page cache, and using
2997                 * underlying segment to calculate anon index and get
2998                 * anonmap and vnode pointer from
2999                 */
3000                if (shmd->shm_softlockcnt > 0)
3001                        segspt_purge(seg);

3003                page_mark_migrate(seg, shm_addr, size, amp, 0, NULL, 0, 0);
3004        }

3006        return (0);
3009 }

3011 /*ARGSUSED*/
3012 void
3013 segspt_shmdump(struct seg *seg)
3014 {
3015         /* no-op for ISM segment */
3007 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   54578 Tue Nov 24 09:35:09 2015
new/usr/src/uts/common/vm/vm_seg.c
6152 use NULL dump segop as a shorthand for no-op
**********************************************************
______unchanged_portion_omitted_

1964 void
1965 segop_dump(struct seg *seg)
1966 {
1967         if (seg->s_ops->dump == NULL)
1968                 return;

1970 #endif /* ! codereview */
1971         seg->s_ops->dump(seg);
1972 }

1974 int
1975 segop_pagelock(struct seg *seg, caddr_t addr, size_t len, struct page ***page,
1976     enum lock_type type, enum seg_rw rw)
1977 {
1978         return (seg->s_ops->pagelock(seg, addr, len, page, type, rw));
1979 }

1981 int
1982 segop_setpagesize(struct seg *seg, caddr_t addr, size_t len, uint_t szc)
1983 {
1984         if (seg->s_ops->setpagesize == NULL)
1985                 return (ENOTSUP);

1987         return (seg->s_ops->setpagesize(seg, addr, len, szc));
1988 }

1990 int
1991 segop_getmemid(struct seg *seg, caddr_t addr, memid_t *mp)
1992 {
1993         if (seg->s_ops->getmemid == NULL)
1994                 return (ENODEV);

1996         return (seg->s_ops->getmemid(seg, addr, mp));
1997 }

1999 struct lgrp_mem_policy_info *
2000 segop_getpolicy(struct seg *seg, caddr_t addr)
2001 {
2002         if (seg->s_ops->getpolicy == NULL)
2003                 return (NULL);

2005         return (seg->s_ops->getpolicy(seg, addr));
2006 }

2008 int
2009 segop_capable(struct seg *seg, segcapability_t cap)
2010 {
2011         if (seg->s_ops->capable == NULL)
2012                 return (0);

2014         return (seg->s_ops->capable(seg, cap));
2015 }

2017 int
2018 segop_inherit(struct seg *seg, caddr_t addr, size_t len, uint_t op)
2019 {
2020         if (seg->s_ops->inherit == NULL)
2021                 return (ENOTSUP);
```



```
2023         return (seg->s_ops->inherit(seg, addr, len, op));
2024 }
```

```
**********************************************************
   16457 Tue Nov 24 09:35:09 2015
new/usr/src/uts/i86xpv/vm/seg_mf.c
6152 use NULL dump segop as a shorthand for no-op
**********************************************************
______unchanged_portion_omitted_

 472 /*ARGSUSED*/
 473 static void
 474 segmf_dump(struct seg *seg)
 475 {}

 477 /*ARGSUSED*/
 473 static int
 474 segmf_pagelock(struct seg *seg, caddr_t addr, size_t len,
 475     struct page ***ppp, enum lock_type type, enum seg_rw rw)
 476 {
 477         return (ENOTSUP);
 478 }
______unchanged_portion_omitted_

 734 static struct seg_ops segmf_ops = {
 735         .dup            = segmf_dup,
 736         .unmap          = segmf_unmap,
 737         .free           = segmf_free,
 738         .fault          = segmf_fault,
 739         .faulta         = segmf_faulta,
 740         .setprot        = segmf_setprot,
 741         .checkprot      = segmf_checkprot,
 742         .kluster        = segmf_kluster,
 743         .sync           = segmf_sync,
 744         .incore         = segmf_incore,
 745         .lockop         = segmf_lockop,
 746         .getprot        = segmf_getprot,
 747         .getoffset      = segmf_getoffset,
 748         .gettype        = segmf_gettype,
 749         .getvp          = segmf_getvp,
 750         .advise         = segmf_advise,
 756         .dump           = segmf_dump,
 751         .pagelock       = segmf_pagelock,
 752         .getmemid       = segmf_getmemid,
 753 };
______unchanged_portion_omitted_
```

```
**********************************************************
   11601 Tue Nov 24 09:35:09 2015
new/usr/src/uts/sparc/v9/vm/seg_nf.c
6152 use NULL dump segop as a shorthand for no-op
**********************************************************
```

```
/*
 * 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);


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,
};
```

_____**unchanged_portion_omitted_**

```
/*
 * segnf pages are not dumped, so we just return
 */
/* ARGSUSED */
static void
segnf_dump(struct seg *seg)
{}

/*ARGSUSED*/
static int
segnf_pagelock(struct seg *seg, caddr_t addr, size_t len,
    struct page ***ppp, enum lock_type type, enum seg_rw rw)
{
        return (ENOTSUP);
}
```

_____**unchanged_portion_omitted_**