****
113669 Fri May 8 18:04:37 2015
new/usr/src/uts/common/vm/seg_dev.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
****

```c
/*
 * 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>
```

```c
#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
 * keep the page locked, however this complicates the fault code. RFE.
 */
static struct vnode trashvp;
static struct page *trashpp;

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

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

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

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

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

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

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

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

/*
 * Private seg op routines.
 */
static int      segdev_dup(struct seg *, struct seg *);
static int      segdev_unmap(struct seg *, caddr_t, size_t);
static void     segdev_free(struct seg *);
static faultcode_t segdev_fault(struct hat *, struct seg *, caddr_t, size_t,
                    enum fault_type, enum seg_rw);
static faultcode_t segdev_faulta(struct seg *, caddr_t);
static int      segdev_setprot(struct seg *, caddr_t, size_t, uint_t);
static int      segdev_checkprot(struct seg *, caddr_t, size_t, uint_t);
static void     segdev_badop(void);
static int      segdev_sync(struct seg *, caddr_t, size_t, int, uint_t);
static size_t   segdev_incore(struct seg *, caddr_t, size_t, char *);
static int      segdev_lockop(struct seg *, caddr_t, size_t, int, int,
                    ulong_t *, size_t);
static int      segdev_getprot(struct seg *, caddr_t, size_t, uint_t *);
static u_offset_t       segdev_getoffset(struct seg *, caddr_t);
static int      segdev_gettype(struct seg *, caddr_t);
static int      segdev_getvp(struct seg *, caddr_t, struct vnode **);
static int      segdev_advise(struct seg *, caddr_t, size_t, uint_t);
static void     segdev_dump(struct seg *);
static int      segdev_pagelock(struct seg *, caddr_t, size_t,
                    struct page ***, enum lock_type, enum seg_rw);
static int      segdev_setpagesize(struct seg *, caddr_t, size_t, uint_t);
static int      segdev_getmemid(struct seg *, caddr_t, memid_t *);
static int      segdev_capable(struct seg *, segcapability_t);

/*
 * XXX  this struct is used by rootnex_map_fault to identify
 *      the segment it has been passed. So if you make it
```

```
 *      "static" you'll need to fix rootnex_map_fault.
 */
struct seg_ops segdev_ops = {
        .dup            = segdev_dup,
        .unmap          = segdev_unmap,
        .free           = segdev_free,
        .fault          = segdev_fault,
        .faulta         = segdev_faulta,
        .setprot        = segdev_setprot,
        .checkprot      = segdev_checkprot,
        .kluster        = (int (*)())segdev_badop,
        .sync           = segdev_sync,
        .incore         = segdev_incore,
        .lockop         = segdev_lockop,
        .getprot        = segdev_getprot,
        .getoffset      = segdev_getoffset,
        .gettype        = segdev_gettype,
        .getvp          = segdev_getvp,
        .advise         = segdev_advise,
        .dump           = segdev_dump,
        .pagelock       = segdev_pagelock,
        .setpagesize    = segdev_setpagesize,
        .getmemid       = segdev_getmemid,
        .capable        = segdev_capable,
};
```
______***unchanged_portion_omitted_***

```
static int
segdev_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp)
{
        struct segdev_data *sdp = (struct segdev_data *)seg->s_data;

        /*
         * It looks as if it is always mapped shared
         */
        TRACE_0(TR_FAC_DEVMAP, TR_DEVMAP_GETMEMID,
            "segdev_getmemid:start");
        memidp->val[0] = (uintptr_t)VTOCVP(sdp->vp);
        memidp->val[1] = sdp->offset + (uintptr_t)(addr - seg->s_base);
        return (0);
}

/*ARGSUSED*/
static int
segdev_capable(struct seg *seg, segcapability_t capability)
{
        return (0);
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   35855 Fri May  8 18:04:38 2015
new/usr/src/uts/common/vm/seg_kp.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
**********************************************************
/*
 * 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) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/* Copyright (c) 1984, 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.
 */

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

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

```
#include <sys/debug.h>
#include <sys/vtrace.h>
#include <sys/stack.h>
#include <sys/atomic.h>
#include <sys/archsystm.h>
#include <sys/lgrp.h>

#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_kp.h>
#include <vm/seg_kmem.h>
#include <vm/anon.h>
#include <vm/page.h>
#include <vm/hat.h>
#include <sys/bitmap.h>

/*
 * Private seg op routines
 */
static void     segkp_dump(struct seg *seg);
static int      segkp_checkprot(struct seg *seg, caddr_t addr, size_t len,
                        uint_t prot);
static int      segkp_kluster(struct seg *seg, caddr_t addr, ssize_t delta);
static int      segkp_pagelock(struct seg *seg, caddr_t addr, size_t len,
                        struct page ***page, enum lock_type type,
                        enum seg_rw rw);
static void     segkp_insert(struct seg *seg, struct segkp_data *kpd);
static void     segkp_delete(struct seg *seg, struct segkp_data *kpd);
static caddr_t  segkp_get_internal(struct seg *seg, size_t len, uint_t flags,
                        struct segkp_data **tkpd, struct anon_map *amp);
static void     segkp_release_internal(struct seg *seg,
                        struct segkp_data *kpd, size_t len);
static int      segkp_unlock(struct hat *hat, struct seg *seg, caddr_t vaddr,
                        size_t len, struct segkp_data *kpd, uint_t flags);
static int      segkp_load(struct hat *hat, struct seg *seg, caddr_t vaddr,
                        size_t len, struct segkp_data *kpd, uint_t flags);
static struct   segkp_data *segkp_find(struct seg *seg, caddr_t vaddr);
static int      segkp_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp);
static int      segkp_capable(struct seg *seg, segcapability_t capability);

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

/*
 * The segkp caches
 */
static struct segkp_cache segkp_cache[SEGKP_MAX_CACHE];

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

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

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

hrtime_t        red_deep_hires;
kthread_t       *red_deep_thread;

uint32_t        red_nmapped;
uint32_t        red_closest = UINT_MAX;
uint32_t        red_ndoubles;

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

static struct   seg_ops segkp_ops = {
        .fault          = segkp_fault,
        .checkprot      = segkp_checkprot,
        .kluster        = segkp_kluster,
        .dump           = segkp_dump,
        .pagelock       = segkp_pagelock,
        .getmemid       = segkp_getmemid,
        .capable        = segkp_capable,
};
```
______***unchanged_portion_omitted_***

```
/*ARGSUSED*/
static int
segkp_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp)
{
        return (ENODEV);
}

/*ARGSUSED*/
static int
segkp_capable(struct seg *seg, segcapability_t capability)
{
        return (0);
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    9347 Fri May  8 18:04:38 2015
new/usr/src/uts/common/vm/seg_kpm.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
**********************************************************
/*
 * 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 int      segkpm_pagelock(struct seg *seg, caddr_t addr, size_t len,
                        struct page ***page, enum lock_type type,
                        enum seg_rw rw);
static int      segkpm_capable(struct seg *, segcapability_t);

static struct seg_ops segkpm_ops = {
        .fault          = segkpm_fault,
        .dump           = segkpm_dump,
        .pagelock       = segkpm_pagelock,
        .capable        = segkpm_capable,
//#ifndef SEGKPM_SUPPORT
#if 0
#error FIXME: define nop
        .dup            = nop,
        .unmap          = nop,
        .free           = nop,
        .faulta         = nop,
        .setprot        = nop,
        .checkprot      = nop,
        .kluster        = nop,
        .sync           = nop,
        .incore         = nop,
        .lockop         = nop,
        .getprot        = nop,
        .getoffset      = nop,
        .gettype        = nop,
        .getvp          = nop,
        .advise         = nop,
        .setpagesize    = nop,
        .getmemid       = nop,
        .getpolicy      = nop,
#endif
};
```

______***unchanged_portion_omitted_***

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

/*
 * We claim to have no special capabilities.
 */
/*ARGSUSED*/
static int
segkpm_capable(struct seg *seg, segcapability_t capability)
{
        return (0);
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   57265 Fri May  8 18:04:38 2015
new/usr/src/uts/common/vm/seg_map.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
**********************************************************
```

```
/*
 * 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 int	segmap_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp);
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 **);

static struct seg_ops segmap_ops = {
	.free		= segmap_free,
	.fault		= segmap_fault,
	.faulta		= segmap_faulta,
	.checkprot	= segmap_checkprot,
	.kluster	= segmap_kluster,
	.getprot	= segmap_getprot,
	.getoffset	= segmap_getoffset,
	.gettype	= segmap_gettype,
	.getvp		= segmap_getvp,
	.dump		= segmap_dump,
	.pagelock	= segmap_pagelock,
	.getmemid	= segmap_getmemid,
	.capable	= segmap_capable,
};
```

______***unchanged_portion_omitted_***

```
static int
segmap_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp)
{
	struct segmap_data *smd = (struct segmap_data *)seg->s_data;

	memidp->val[0] = (uintptr_t)smd->smd_sm->sm_vp;
	memidp->val[1] = smd->smd_sm->sm_off + (uintptr_t)(addr - seg->s_base);
	return (0);
}
```

```
/*ARGSUSED*/
static int
segmap_capable(struct seg *seg, segcapability_t capability)
{
        return (0);
}
```
______***unchanged_portion_omitted_***

****
   82532 Fri May 8 18:04:39 2015
new/usr/src/uts/common/vm/seg_spt.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
****
______*unchanged_portion_omitted_*

```
static int segspt_shmdup(struct seg *seg, struct seg *newseg);
static int segspt_shmunmap(struct seg *seg, caddr_t raddr, size_t ssize);
static void segspt_shmfree(struct seg *seg);
static faultcode_t segspt_shmfault(struct hat *hat, struct seg *seg,
                caddr_t addr, size_t len, enum fault_type type, enum seg_rw rw);
static faultcode_t segspt_shmfaulta(struct seg *seg, caddr_t addr);
static int segspt_shmsetprot(register struct seg *seg, register caddr_t addr,
                        register size_t len, register uint_t prot);
static int segspt_shmcheckprot(struct seg *seg, caddr_t addr, size_t size,
                        uint_t prot);
static int      segspt_shmkluster(struct seg *seg, caddr_t addr, ssize_t delta);
static size_t segspt_shmincore(struct seg *seg, caddr_t addr, size_t len,
                        register char *vec);
static int segspt_shmsync(struct seg *seg, register caddr_t addr, size_t len,
                        int attr, uint_t flags);
static int segspt_shmlockop(struct seg *seg, caddr_t addr, size_t len,
                        int attr, int op, ulong_t *lockmap, size_t pos);
static int segspt_shmgetprot(struct seg *seg, caddr_t addr, size_t len,
                        uint_t *protv);
static u_offset_t segspt_shmgetoffset(struct seg *seg, caddr_t addr);
static int segspt_shmgettype(struct seg *seg, caddr_t addr);
static int segspt_shmgetvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
static int segspt_shmadvise(struct seg *seg, caddr_t addr, size_t len,
                        uint_t behav);
static void segspt_shmdump(struct seg *seg);
static int segspt_shmpagelock(struct seg *, caddr_t, size_t,
                        struct page ***, enum lock_type, enum seg_rw);
static int segspt_shmsetpgsz(struct seg *, caddr_t, size_t, uint_t);
static int segspt_shmgetmemid(struct seg *, caddr_t, memid_t *);
static lgrp_mem_policy_info_t *segspt_shmgetpolicy(struct seg *, caddr_t);
static int segspt_shmcapable(struct seg *, segcapability_t);

struct seg_ops segspt_shmops = {
        .dup            = segspt_shmdup,
        .unmap          = segspt_shmunmap,
        .free           = segspt_shmfree,
        .fault          = segspt_shmfault,
        .faulta         = segspt_shmfaulta,
        .setprot        = segspt_shmsetprot,
        .checkprot      = segspt_shmcheckprot,
        .kluster        = segspt_shmkluster,
        .sync           = segspt_shmsync,
        .incore         = segspt_shmincore,
        .lockop         = segspt_shmlockop,
        .getprot        = segspt_shmgetprot,
        .getoffset      = segspt_shmgetoffset,
        .gettype        = segspt_shmgettype,
        .getvp          = segspt_shmgetvp,
        .advise         = segspt_shmadvise,
        .dump           = segspt_shmdump,
        .pagelock       = segspt_shmpagelock,
        .setpagesize    = segspt_shmsetpgsz,
        .getmemid       = segspt_shmgetmemid,
        .getpolicy      = segspt_shmgetpolicy,
        .capable        = segspt_shmcapable,
};
```
______*unchanged_portion_omitted_*

```
/*
 * Get memory allocation policy info for specified address in given segment
 */
static lgrp_mem_policy_info_t *
segspt_shmgetpolicy(struct seg *seg, caddr_t addr)
{
        struct anon_map         *amp;
        ulong_t                 anon_index;
        lgrp_mem_policy_info_t  *policy_info;
        struct shm_data         *shm_data;

        ASSERT(seg != NULL);

        /*
         * Get anon_map from segshm
         *
         * Assume that no lock needs to be held on anon_map, since
         * it should be protected by its reference count which must be
         * nonzero for an existing segment
         * Need to grab readers lock on policy tree though
         */
        shm_data = (struct shm_data *)seg->s_data;
        if (shm_data == NULL)
                return (NULL);
        amp = shm_data->shm_amp;
        ASSERT(amp->refcnt != 0);

        /*
         * Get policy info
         *
         * Assume starting anon index of 0
         */
        anon_index = seg_page(seg, addr);
        policy_info = lgrp_shm_policy_get(amp, anon_index, NULL, 0);

        return (policy_info);
}

/*ARGSUSED*/
static int
segspt_shmcapable(struct seg *seg, segcapability_t capability)
{
        return (0);
}
```
______*unchanged_portion_omitted_*

```
**********************************************************
  280458 Fri May  8 18:04:39 2015
new/usr/src/uts/common/vm/seg_vn.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
**********************************************************
/*
 * 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) 1986, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2015, Joyent, Inc. All rights reserved.
 * Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
 */

/*      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 - shared or copy-on-write from a vnode/anonymous memory.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/t_lock.h>
#include <sys/errno.h>
#include <sys/systm.h>
#include <sys/mman.h>
#include <sys/debug.h>
#include <sys/cred.h>
#include <sys/vmsystm.h>
#include <sys/tuneable.h>
#include <sys/bitmap.h>
#include <sys/swap.h>
#include <sys/kmem.h>
#include <sys/sysmacros.h>
#include <sys/vtrace.h>
```

```
#include <sys/cmn_err.h>
#include <sys/callb.h>
#include <sys/vm.h>
#include <sys/dumphdr.h>
#include <sys/lgrp.h>

#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_vn.h>
#include <vm/pvn.h>
#include <vm/anon.h>
#include <vm/page.h>
#include <vm/vpage.h>
#include <sys/proc.h>
#include <sys/task.h>
#include <sys/project.h>
#include <sys/zone.h>
#include <sys/shm_impl.h>

/*
 * segvn_fault needs a temporary page list array.  To avoid calling kmem all
 * the time, it creates a small (PVN_GETPAGE_NUM entry) array and uses it if
 * it can.  In the rare case when this page list is not large enough, it
 * goes and gets a large enough array from kmem.
 *
 * This small page list array covers either 8 pages or 64kB worth of pages -
 * whichever is smaller.
 */
#define PVN_MAX_GETPAGE_SZ      0x10000
#define PVN_MAX_GETPAGE_NUM     0x8

#if PVN_MAX_GETPAGE_SZ > PVN_MAX_GETPAGE_NUM * PAGESIZE
#define PVN_GETPAGE_SZ  ptob(PVN_MAX_GETPAGE_NUM)
#define PVN_GETPAGE_NUM PVN_MAX_GETPAGE_NUM
#else
#define PVN_GETPAGE_SZ  PVN_MAX_GETPAGE_SZ
#define PVN_GETPAGE_NUM btop(PVN_MAX_GETPAGE_SZ)
#endif

/*
 * Private seg op routines.
 */
static int      segvn_dup(struct seg *seg, struct seg *newseg);
static int      segvn_unmap(struct seg *seg, caddr_t addr, size_t len);
static void     segvn_free(struct seg *seg);
static faultcode_t segvn_fault(struct hat *hat, struct seg *seg,
                    caddr_t addr, size_t len, enum fault_type type,
                    enum seg_rw rw);
static faultcode_t segvn_faulta(struct seg *seg, caddr_t addr);
static int      segvn_setprot(struct seg *seg, caddr_t addr,
                    size_t len, uint_t prot);
static int      segvn_checkprot(struct seg *seg, caddr_t addr,
                    size_t len, uint_t prot);
static int      segvn_kluster(struct seg *seg, caddr_t addr, ssize_t delta);
static int      segvn_sync(struct seg *seg, caddr_t addr, size_t len,
                    int attr, uint_t flags);
static size_t   segvn_incore(struct seg *seg, caddr_t addr, size_t len,
                    char *vec);
static int      segvn_lockop(struct seg *seg, caddr_t addr, size_t len,
                    int attr, int op, ulong_t *lockmap, size_t pos);
static int      segvn_getprot(struct seg *seg, caddr_t addr, size_t len,
                    uint_t *protv);
static u_offset_t       segvn_getoffset(struct seg *seg, caddr_t addr);
static int      segvn_gettype(struct seg *seg, caddr_t addr);
static int      segvn_getvp(struct seg *seg, caddr_t addr,
```

```c
                    struct vnode **vpp);
static int      segvn_advise(struct seg *seg, caddr_t addr, size_t len,
                    uint_t behav);
static void     segvn_dump(struct seg *seg);
static int      segvn_pagelock(struct seg *seg, caddr_t addr, size_t len,
                    struct page ***ppp, enum lock_type type, enum seg_rw rw);
static int      segvn_setpagesize(struct seg *seg, caddr_t addr, size_t len,
                    uint_t szc);
static int      segvn_getmemid(struct seg *seg, caddr_t addr,
                    memid_t *memidp);
static lgrp_mem_policy_info_t   *segvn_getpolicy(struct seg *, caddr_t);
static int      segvn_capable(struct seg *seg, segcapability_t capable);
static int      segvn_inherit(struct seg *, caddr_t, size_t, uint_t);

struct  seg_ops segvn_ops = {
        .dup            = segvn_dup,
        .unmap          = segvn_unmap,
        .free           = segvn_free,
        .fault          = segvn_fault,
        .faulta         = segvn_faulta,
        .setprot        = segvn_setprot,
        .checkprot      = segvn_checkprot,
        .kluster        = segvn_kluster,
        .sync           = segvn_sync,
        .incore         = segvn_incore,
        .lockop         = segvn_lockop,
        .getprot        = segvn_getprot,
        .getoffset      = segvn_getoffset,
        .gettype        = segvn_gettype,
        .getvp          = segvn_getvp,
        .advise         = segvn_advise,
        .dump           = segvn_dump,
        .pagelock       = segvn_pagelock,
        .setpagesize    = segvn_setpagesize,
        .getmemid       = segvn_getmemid,
        .getpolicy      = segvn_getpolicy,
        .capable        = segvn_capable,
        .inherit        = segvn_inherit,
};
```
______***unchanged_portion_omitted_***

```c
/*
 * Get memory allocation policy info for specified address in given segment
 */
static lgrp_mem_policy_info_t *
segvn_getpolicy(struct seg *seg, caddr_t addr)
{
        struct anon_map         *amp;
        ulong_t                 anon_index;
        lgrp_mem_policy_info_t  *policy_info;
        struct segvn_data       *svn_data;
        u_offset_t              vn_off;
        vnode_t                 *vp;

        ASSERT(seg != NULL);

        svn_data = (struct segvn_data *)seg->s_data;
        if (svn_data == NULL)
                return (NULL);

        /*
         * Get policy info for private or shared memory
         */
        if (svn_data->type != MAP_SHARED) {
                if (svn_data->tr_state != SEGVN_TR_ON) {
                        policy_info = &svn_data->policy_info;
```

```c
                } else {
                        policy_info = &svn_data->tr_policy_info;
                        ASSERT(policy_info->mem_policy ==
                            LGRP_MEM_POLICY_NEXT_SEG);
                }
        } else {
                amp = svn_data->amp;
                anon_index = svn_data->anon_index + seg_page(seg, addr);
                vp = svn_data->vp;
                vn_off = svn_data->offset + (uintptr_t)(addr - seg->s_base);
                policy_info = lgrp_shm_policy_get(amp, anon_index, vp, vn_off);
        }

        return (policy_info);
}

/*ARGSUSED*/
static int
segvn_capable(struct seg *seg, segcapability_t capability)
{
        return (0);
}
```
______***unchanged_portion_omitted_***

****
   55121 Fri May 8 18:04:39 2015
new/usr/src/uts/common/vm/vm_seg.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
****

______unchanged_portion_omitted_

```
int
segop_capable(struct seg *seg, segcapability_t cap)
{
        if (seg->s_ops->capable == NULL)
                return (0);
        VERIFY3P(seg->s_ops->capable, !=, NULL);

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

______unchanged_portion_omitted_

****
   16692 Fri May 8 18:04:40 2015
new/usr/src/uts/i86xpv/vm/seg_mf.c
use NULL capable segop as a shorthand for no-capabilities
Instead of forcing every segment driver to implement a dummy "return 0"
function, handle NULL capable segop function pointer as "no copabilities
supported" shorthand.
****
______***unchanged_portion_omitted_***

```
/*ARGSUSED*/
static int
segmf_capable(struct seg *seg, segcapability_t capability)
{
        return (0);
}

/*
 * Add a set of contiguous foreign MFNs to the segment. soft-locking them.  The
 * pre-faulting is necessary due to live migration; in particular we must
 * return an error in response to IOCTL_PRIVCMD_MMAPBATCH rather than faulting
 * later on a bad MFN.  Whilst this isn't necessary for the other MMAP
 * ioctl()s, we lock them too, as they should be transitory.
 */
int
segmf_add_mfns(struct seg *seg, caddr_t addr, mfn_t mfn,
    pgcnt_t pgcnt, domid_t domid)
{
        struct segmf_data *data = seg->s_data;
        pgcnt_t base;
        faultcode_t fc;
        pgcnt_t i;
        int error = 0;

        if (seg->s_ops != &segmf_ops)
                return (EINVAL);

        /*
         * Don't mess with dom0.
         *
         * Only allow the domid to be set once for the segment.
         * After that attempts to add mappings to this segment for
         * other domains explicitly fails.
         */

        if (domid == 0 || domid == DOMID_SELF)
                return (EACCES);

        mutex_enter(&data->lock);

        if (data->domid == 0)
                data->domid = domid;

        if (data->domid != domid) {
                error = EINVAL;
                goto out;
        }

        base = seg_page(seg, addr);

        for (i = 0; i < pgcnt; i++) {
                data->map[base + i].t_type = SEGMF_MAP_MFN;
                data->map[base + i].u.m.m_mfn = mfn++;
        }

        fc = segmf_fault_range(seg->s_as->a_hat, seg, addr,
```

```
            pgcnt * MMU_PAGESIZE, F_SOFTLOCK, S_OTHER);

        if (fc != 0) {
                error = fc_decode(fc);
                for (i = 0; i < pgcnt; i++) {
                        data->map[base + i].t_type = SEGMF_MAP_EMPTY;
                }
        }

out:
        mutex_exit(&data->lock);
        return (error);
}
```

______***unchanged_portion_omitted_***

```
static struct seg_ops segmf_ops = {
        .dup            = segmf_dup,
        .unmap          = segmf_unmap,
        .free           = segmf_free,
        .fault          = segmf_fault,
        .faulta         = segmf_faulta,
        .setprot        = segmf_setprot,
        .checkprot      = segmf_checkprot,
        .kluster        = segmf_kluster,
        .sync           = segmf_sync,
        .incore         = segmf_incore,
        .lockop         = segmf_lockop,
        .getprot        = segmf_getprot,
        .getoffset      = segmf_getoffset,
        .gettype        = segmf_gettype,
        .getvp          = segmf_getvp,
        .advise         = segmf_advise,
        .dump           = segmf_dump,
        .pagelock       = segmf_pagelock,
        .setpagesize    = segmf_setpagesize,
        .getmemid       = segmf_getmemid,
        .capable        = segmf_capable,
};
```

______***unchanged_portion_omitted_***