```
**********************************************************
    9943 Fri May  8 18:04:22 2015
new/usr/src/uts/common/vm/seg.h
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
**********************************************************
______unchanged_portion_omitted_
```

```
#ifdef _KERNEL

/*
 * Generic segment operations
 */
extern  void    seg_init(void);
extern  struct  seg *seg_alloc(struct as *as, caddr_t base, size_t size);
extern  int     seg_attach(struct as *as, caddr_t base, size_t size,
                        struct seg *seg);
extern  void    seg_unmap(struct seg *seg);
extern  void    seg_free(struct seg *seg);

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

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

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

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

extern  void    seg_pasync_thread(void);
extern  void    seg_preap(void);
extern  int     seg_p_disable(void);
extern  void    seg_p_enable(void);

extern  segadvstat_t    segadvstat;

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

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

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

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

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

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

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

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

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

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

#ifdef VMDEBUG

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

#endif  /* VMDEBUG */

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

/* segop wrappers */
int segop_dup(struct seg *, struct seg *);
int segop_unmap(struct seg *, caddr_t, size_t);
void segop_free(struct seg *);
faultcode_t segop_fault(struct hat *, struct seg *, caddr_t, size_t, enum fault_
faultcode_t segop_faulta(struct seg *, caddr_t);
int segop_setprot(struct seg *, caddr_t, size_t, uint_t);
int segop_checkprot(struct seg *, caddr_t, size_t, uint_t);
int segop_kluster(struct seg *, caddr_t, ssize_t);
int segop_sync(struct seg *, caddr_t, size_t, int, uint_t);
size_t segop_incore(struct seg *, caddr_t, size_t, char *);
int segop_lockop(struct seg *, caddr_t, size_t, int, int, ulong_t *, size_t );
int segop_getprot(struct seg *, caddr_t, size_t, uint_t *);
u_offset_t segop_getoffset(struct seg *, caddr_t);
int segop_gettype(struct seg *, caddr_t);
int segop_getvp(struct seg *, caddr_t, struct vnode **);
int segop_advise(struct seg *, caddr_t, size_t, uint_t);
void segop_dump(struct seg *);
int segop_pagelock(struct seg *, caddr_t, size_t, struct page ***, enum lock_typ
int segop_setpagesize(struct seg *, caddr_t, size_t, uint_t);
int segop_getmemid(struct seg *, caddr_t, memid_t *);
struct lgrp_mem_policy_info *segop_getpolicy(struct seg *, caddr_t);
int segop_capable(struct seg *, segcapability_t);
int segop_inherit(struct seg *, caddr_t, size_t, uint_t);

#endif  /* _KERNEL */

#ifdef  __cplusplus
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
  114075 Fri May  8 18:04:22 2015
new/usr/src/uts/common/vm/seg_dev.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
**********************************************************
/*
 * 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
 * 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 lgrp_mem_policy_info_t   *segdev_getpolicy(struct seg *, caddr_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,
        .getpolicy      = segdev_getpolicy,
        .capable        = segdev_capable,
        .inherit        = seg_inherit_notsup,
};
```

______***unchanged_portion_omitted_***

****
44900 Fri May 8 18:04:22 2015
new/usr/src/uts/common/vm/seg_kmem.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
****
______***unchanged_portion_omitted_***

```
static struct seg_ops segkmem_ops = {
        .fault          = segkmem_fault,
        .setprot        = segkmem_setprot,
        .checkprot      = segkmem_checkprot,
        .kluster        = segkmem_kluster,
        .dump           = segkmem_dump,
        .pagelock       = segkmem_pagelock,
        .getmemid       = segkmem_getmemid,
        .getpolicy      = segkmem_getpolicy,
        .capable        = segkmem_capable,
        .inherit        = seg_inherit_notsup,
};
```
______***unchanged_portion_omitted_***

```
**********************************************************
   36281 Fri May  8 18:04:23 2015
new/usr/src/uts/common/vm/seg_kp.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
**********************************************************
/*
 * 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 lgrp_mem_policy_info_t   *segkp_getpolicy(struct seg *seg,
    caddr_t addr);
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,
        .getpolicy      = segkp_getpolicy,
        .capable        = segkp_capable,
        .inherit        = seg_inherit_notsup,
};
```
______***unchanged_portion_omitted_***

```
**********************************************************
    9587 Fri May  8 18:04:23 2015
new/usr/src/uts/common/vm/seg_kpm.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
**********************************************************
/*
 * 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,
        .inherit        = seg_inherit_notsup,
//#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_***

```
**********************************************************
   57697 Fri May  8 18:04:23 2015
new/usr/src/uts/common/vm/seg_map.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
**********************************************************
/*
 * 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 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 **);

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,
        .getpolicy      = segmap_getpolicy,
        .capable        = segmap_capable,
        .inherit        = seg_inherit_notsup,
};
______unchanged_portion_omitted_
```

```
************************************************************
   82729 Fri May  8 18:04:23 2015
new/usr/src/uts/common/vm/seg_spt.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
************************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
 */

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

#define SEGSPTADDR      (caddr_t)0x0

/*
 * # pages used for spt
 */
size_t  spt_used;

/*
 * segspt_minfree is the memory left for system after ISM
```

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

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

struct seg_ops segspt_ops = {
        .unmap          = segspt_unmap,
        .free           = segspt_free,
        .getpolicy      = segspt_getpolicy,
        .inherit        = seg_inherit_notsup,
};

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,
        .inherit        = seg_inherit_notsup,
};
```

______***unchanged_portion_omitted_***

```
**********************************************************
   90610 Fri May  8 18:04:24 2015
new/usr/src/uts/common/vm/vm_as.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
**********************************************************
______unchanged_portion_omitted_

/*
 * Cache control operations over the interval [addr, addr + size) in
 * address space "as".
 */
/*ARGSUSED*/
int
as_ctl(struct as *as, caddr_t addr, size_t size, int func, int attr,
    uintptr_t arg, ulong_t *lock_map, size_t pos)
{
        struct seg *seg;        /* working segment */
        caddr_t raddr;          /* rounded down addr */
        caddr_t initraddr;      /* saved initial rounded down addr */
        size_t rsize;           /* rounded up size */
        size_t initrsize;       /* saved initial rounded up size */
        size_t ssize;           /* size of seg */
        int error = 0;                  /* result */
        size_t mlock_size;      /* size of bitmap */
        ulong_t *mlock_map;     /* pointer to bitmap used */
                                /* to represent the locked */
                                /* pages. */
retry:
        if (error == IE_RETRY)
                AS_LOCK_ENTER(as, &as->a_lock, RW_WRITER);
        else
                AS_LOCK_ENTER(as, &as->a_lock, RW_READER);

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

                idx = pos;

                if (arg & MCL_FUTURE) {
                        mutex_enter(&as->a_contents);
                        AS_SETPGLCK(as);
                        mutex_exit(&as->a_contents);
                }
                if ((arg & MCL_CURRENT) == 0) {
                        AS_LOCK_EXIT(as, &as->a_lock);
                        return (0);
                }

                seg = AS_SEGFIRST(as);
                if (seg == NULL) {
                        AS_LOCK_EXIT(as, &as->a_lock);
                        return (0);
                }

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

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

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

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

                                raddr = (caddr_t)((uintptr_t)seg->s_base &
                                    (uintptr_t)PAGEMASK);
                                npages = seg_pages(seg);
                                as_segunlock(seg, raddr, attr, mlock_map,
                                    idx, npages);
                                idx += npages;
                        }
                }

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

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

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

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

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

        /*
         * Get initial segment.
         */
        if ((seg = as_segat(as, raddr)) == NULL) {
                AS_LOCK_EXIT(as, &as->a_lock);
                return (ENOMEM);
        }
        if (func == MC_LOCK) {
                mlock_size = BT_BITOUL(btopr(rsize));
                if ((mlock_map = (ulong_t *)kmem_zalloc(mlock_size *
                    sizeof (ulong_t), KM_NOSLEEP)) == NULL) {
                                AS_LOCK_EXIT(as, &as->a_lock);
                                return (EAGAIN);
                }
        }

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

                /*
                 * Make sure there's no hole, calculate the portion
                 * of the next segment to be operated over.
                 */
                if (raddr >= seg->s_base + seg->s_size) {
                        seg = AS_SEGNEXT(as, seg);
                        if (seg == NULL || raddr != seg->s_base) {
                                if (func == MC_LOCK) {
                                        as_unlockerr(as, attr, mlock_map,
                                            initraddr, initrsize - rsize);
                                        kmem_free(mlock_map,
                                            mlock_size * sizeof (ulong_t));
                                }
                                AS_LOCK_EXIT(as, &as->a_lock);
                                return (ENOMEM);
                        }
                }
                if ((raddr + rsize) > (seg->s_base + seg->s_size))
                        ssize = seg->s_base + seg->s_size - raddr;
                else
                        ssize = rsize;

                /*
                 * Dispatch on specific function.
                 */
                switch (func) {

                /*
                 * Synchronize cached data from mappings with backing
                 * objects.
                 */
                case MC_SYNC:
                        if (error = segop_sync(seg, raddr, ssize,
                            attr, (uint_t)arg)) {
                                AS_LOCK_EXIT(as, &as->a_lock);
                                return (error);
                        }
                        break;

                /*
                 * Lock pages in memory.
                 */
                case MC_LOCK:
                        if (error = segop_lockop(seg, raddr, ssize,
                            attr, func, mlock_map, pos)) {
                                as_unlockerr(as, attr, mlock_map, initraddr,
                                    initrsize - rsize + ssize);
                                kmem_free(mlock_map, mlock_size *
                                    sizeof (ulong_t));
```

```
                                AS_LOCK_EXIT(as, &as->a_lock);
                                goto lockerr;
                        }
                        break;

                /*
                 * Unlock mapped pages.
                 */
                case MC_UNLOCK:
                        (void) segop_lockop(seg, raddr, ssize, attr, func,
                            (ulong_t *)NULL, (size_t)NULL);
                        break;

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

                        /*
                         * Check for regular errors and special retry error
                         */
                        if (error) {
                                if (error == IE_RETRY) {
                                        /*
                                         * Need to acquire writers lock, so
                                         * have to drop readers lock and start
                                         * all over again
                                         */
                                        AS_LOCK_EXIT(as, &as->a_lock);
                                        goto retry;
                                } else if (error == IE_REATTACH) {
                                        /*
                                         * Find segment for current address
                                         * because current segment just got
                                         * split or concatenated
                                         */
                                        seg = as_segat(as, raddr);
                                        if (seg == NULL) {
                                                AS_LOCK_EXIT(as, &as->a_lock);
                                                return (ENOMEM);
                                        }
                                } else {
                                        /*
                                         * Regular error
                                         */
                                        AS_LOCK_EXIT(as, &as->a_lock);
                                        return (error);
                                }
                        }
                        break;

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

```
		/*
		 * Can’t happen.
		 */
		default:
			panic("as_ctl: bad operation %d", func);
			/*NOTREACHED*/
		}

		rsize -= ssize;
		raddr += ssize;
	}

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

	/*
	 * If the lower levels returned EDEADLK for a segment lockop,
	 * it means that we should retry the operation.  Let’s wait
	 * a bit also to let the deadlock causing condition clear.
	 * This is part of a gross hack to work around a design flaw
	 * in the ufs/sds logging code and should go away when the
	 * logging code is re-designed to fix the problem. See bug
	 * 4125102 for details of the problem.
	 */
	if (error == EDEADLK) {
		delay(deadlk_wait);
		error = 0;
		goto retry;
	}
	return (error);
}
```
______***unchanged_portion_omitted_***

****
55115 Fri May 8 18:04:24 2015
new/usr/src/uts/common/vm/vm_seg.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
****
______***unchanged_portion_omitted_***

```
/*
 * General not supported function for segop_inherit
 */
/* ARGSUSED */
int
seg_inherit_notsup(struct seg *seg, caddr_t addr, size_t len, uint_t op)
{
        return (ENOTSUP);
}

/*
 * segop wrappers
 */
int
segop_dup(struct seg *seg, struct seg *new)
{
        VERIFY3P(seg->s_ops->dup, !=, NULL);

        return (seg->s_ops->dup(seg, new));
}
```
______***unchanged_portion_omitted_***

****
16964 Fri May 8 18:04:24 2015
new/usr/src/uts/i86xpv/vm/seg_mf.c
seg_inherit_notsup is redundant since segop_inherit checks for NULL properly
****
______***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,
        .getpolicy      = segmf_getpolicy,
        .capable        = segmf_capable,
        .inherit        = seg_inherit_notsup,
};
```
______***unchanged_portion_omitted_***