```
**********************************************************
    9266 Tue Nov 24 09:35:11 2015
new/usr/src/uts/common/vm/seg_kpm.c
6153 use NULL pagelock segop as a shorthand for ENOTSUP
**********************************************************
/*
 * 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_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),
        .getvp          = SEGKPM_BADOP(int),
        .advise         = SEGKPM_BADOP(int),
        .pagelock       = SEGKPM_NOTSUP,
        .setpagesize    = SEGKPM_BADOP(int),
        .getmemid       = SEGKPM_BADOP(int),
        .getpolicy      = SEGKPM_BADOP(lgrp_mem_policy_info_t *),
};
```
______***unchanged_portion_omitted_***

```
/* ARGSUSED */
caddr_t segkpm_create_va(u_offset_t off) { return (NULL); }

/* ARGSUSED */
void segkpm_mapout_validkpme(struct kpme *kpme) {}

static void
segkpm_badop() {}

#endif  /* SEGKPM_SUPPORT */

static int
segkpm_notsup()
{
        return (ENOTSUP);
}
```

```
**********************************************************
   54634 Tue Nov 24 09:35:11 2015
new/usr/src/uts/common/vm/vm_seg.c
6153 use NULL pagelock segop as a shorthand for ENOTSUP
**********************************************************
______unchanged_portion_omitted_

int
segop_pagelock(struct seg *seg, caddr_t addr, size_t len, struct page ***page,
    enum lock_type type, enum seg_rw rw)
{
        if (seg->s_ops->pagelock == NULL)
                return (ENOTSUP);

#endif /* ! codereview */
        return (seg->s_ops->pagelock(seg, addr, len, page, type, rw));
}

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

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

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

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

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

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

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

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

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

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