

new/usr/src/uts/common/vm/seg.h

1

```
*****
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_____

146 #ifndef _KERNEL

148 /*
149  * Generic segment operations
150  */
151 extern void seg_init(void);
152 extern struct seg *seg_alloc(struct as *as, caddr_t base, size_t size);
153 extern int seg_attach(struct as *as, caddr_t base, size_t size,
154                      struct seg *seg);
155 extern void seg_unmap(struct seg *seg);
156 extern void seg_free(struct seg *seg);

158 /*
159  * functions for pagelock cache support
160  */
161 typedef int (*seg_preclaim_cbfunc_t)(void *, caddr_t, size_t,
162                                     struct page **, enum seg_rw, int);

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

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

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

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

185 extern segadvstat_t segadvstat;

187 /*
188  * Flags for pagelock cache support.
189  * Flags argument is passed as uint_t to pcache routines. upper 16 bits of
190  * the flags argument are reserved for alignment page shift when SEGP_PSHIFT
191  * is set.
192  */
193 #define SEGP_FORCE_WIRED 0x1 /* skip check against seg_pwindow */
194 #define SEGP_AMP 0x2 /* anon map's pcache entry */
195 #define SEGP_PSHIFT 0x4 /* addr pgsz shift for hash function */

197 /*
198  * Return values for seg_pininsert and seg_pininsert_check functions.
199  */
200 #define SEGP_SUCCESS 0 /* seg_pininsert() succeeded */
201 #define SEGP_FAIL 1 /* seg_pininsert() failed */

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

new/usr/src/uts/common/vm/seg.h

2

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

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

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

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

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

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

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

238 #ifdef VMDEBUG

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

243 #endif /* VMDEBUG */

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

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

new/usr/src/uts/common/vm/seg.h

3

```
269 struct lgrp_mem_policy_info *segop_getpolicy(struct seg *, caddr_t);
270 int segop_capable(struct seg *, segcapability_t);
271 int segop_inherit(struct seg *, caddr_t, size_t, uint_t);

273 #endif /* _KERNEL */

275 #ifdef __cplusplus
276 }
_____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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

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

30 /*
31  * University Copyright- Copyright (c) 1982, 1986, 1988
32  * The Regents of the University of California
33  * All Rights Reserved
34  *
35  * University Acknowledgment- Portions of this document are derived from
36  * software developed by the University of California, Berkeley, and its
37  * contributors.
38  */

40 /*
41  * VM - segment of a mapped device.
42  *
43  * This segment driver is used when mapping character special devices.
44  */

46 #include <sys/types.h>
47 #include <sys/t_lock.h>
48 #include <sys/sysmacros.h>
49 #include <sys/vtrace.h>
50 #include <sys/system.h>
51 #include <sys/vmsystem.h>
52 #include <sys/mman.h>
53 #include <sys/errno.h>
54 #include <sys/kmem.h>
55 #include <sys/cmn_err.h>
56 #include <sys/vnode.h>
57 #include <sys/proc.h>
58 #include <sys/conf.h>
59 #include <sys/debug.h>
60 #include <sys/ddidevmap.h>
61 #include <sys/ddi_implfuncs.h>

```

```

62 #include <sys/lgrp.h>

64 #include <vm/page.h>
65 #include <vm/hat.h>
66 #include <vm/as.h>
67 #include <vm/seg.h>
68 #include <vm/seg_dev.h>
69 #include <vm/seg_kp.h>
70 #include <vm/seg_kmem.h>
71 #include <vm/vpage.h>

73 #include <sys/sunddi.h>
74 #include <sys/esunddi.h>
75 #include <sys/fs/snodel.h>

78 #if DEBUG
79 int segdev_debug;
80 #define DEBUGF(level, args) { if (segdev_debug >= (level)) cmn_err args; }
81 #else
82 #define DEBUGF(level, args)
83 #endif

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

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

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

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

97 /*
98  * VA_PA_ALIGNED checks to see if both VA and PA are on pgsz boundary
99  * VA_PA_PGSIZE_ALIGNED check to see if VA is aligned with PA w.r.t. pgsz
100 */
101 #define VA_PA_ALIGNED(uvaddr, paddr, pgsz) \
102     (((uvaddr | paddr) & (pgsz - 1)) == 0)
103 #define VA_PA_PGSIZE_ALIGNED(uvaddr, paddr, pgsz) \
104     (((uvaddr ^ paddr) & (pgsz - 1)) == 0)

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

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

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

113 /*
114  * mutex, vnode and page for the page of zeros we use for the trash mappings.
115  * One trash page is allocated on the first ddi_umem_setup call that uses it
116  * XXX Eventually, we may want to combine this with what segnf does when all
117  * hat layers implement HAT_NOFAULT.
118  *
119  * The trash page is used when the backing store for a userland mapping is
120  * removed but the application semantics do not take kindly to a SIGBUS.
121  * In that scenario, the applications pages are mapped to some dummy page
122  * which returns garbage on read and writes go into a common place.
123  * (Perfect for NO_FAULT semantics)
124  * The device driver is responsible to communicating to the app with some
125  * other mechanism that such remapping has happened and the app should take
126  * corrective action.
127  * We can also use an anonymous memory page as there is no requirement to

```

```

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

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

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

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

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

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

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

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

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

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

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

```

```

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

```

unchanged portion omitted

new/usr/src/uts/common/vm/seg\_kmem.c

1

\*\*\*\*\*

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

```
768 static struct seg_ops segkmem_ops = {
769     .fault      = segkmem_fault,
770     .setprot    = segkmem_setprot,
771     .checkprot  = segkmem_checkprot,
772     .kluster    = segkmem_kluster,
773     .dump       = segkmem_dump,
774     .pagelock   = segkmem_pagelock,
775     .getmemid   = segkmem_getmemid,
776     .getpolicy  = segkmem_getpolicy,
777     .capable    = segkmem_capable,
778     .inherit    = seg_inherit_notsup,
778 };
```

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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
23 */

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

28 /*
29 * Portions of this source code were derived from Berkeley 4.3 BSD
30 * under license from the Regents of the University of California.
31 */

33 /*
34 * segkp is a segment driver that administers the allocation and deallocation
35 * of pageable variable size chunks of kernel virtual address space. Each
36 * allocated resource is page-aligned.
37 *
38 * The user may specify whether the resource should be initialized to 0,
39 * include a redzone, or locked in memory.
40 */

42 #include <sys/types.h>
43 #include <sys/t_lock.h>
44 #include <sys/thread.h>
45 #include <sys/param.h>
46 #include <sys/errno.h>
47 #include <sys/sysmacros.h>
48 #include <sys/system.h>
49 #include <sys/buf.h>
50 #include <sys/mman.h>
51 #include <sys/vnode.h>
52 #include <sys/cmn_err.h>
53 #include <sys/swap.h>
54 #include <sys/tuneable.h>
55 #include <sys/kmem.h>
56 #include <sys/vmem.h>
57 #include <sys/cred.h>
58 #include <sys/dumphdr.h>
59 #include <sys/debug.h>
60 #include <sys/vtrace.h>
61 #include <sys/stack.h>

```

```

62 #include <sys/atomic.h>
63 #include <sys/archsystem.h>
64 #include <sys/lgrp.h>

66 #include <vm/as.h>
67 #include <vm/seg.h>
68 #include <vm/seg_kp.h>
69 #include <vm/seg_kmem.h>
70 #include <vm/anon.h>
71 #include <vm/page.h>
72 #include <vm/hat.h>
73 #include <sys/bitmap.h>

75 /*
76  * Private seg op routines
77 */
78 static void      segkp_dump(struct seg *seg);
79 static int       segkp_checkprot(struct seg *seg, caddr_t addr, size_t len,
80                               uint_t prot);
81 static int       segkp_kluster(struct seg *seg, caddr_t addr, ssize_t delta);
82 static int       segkp_pagelock(struct seg *seg, caddr_t addr, size_t len,
83                               struct page **page, enum lock_type type,
84                               enum seg_rw rw);
85 static void      segkp_insert(struct seg *seg, struct segkp_data *kpd);
86 static void      segkp_delete(struct seg *seg, struct segkp_data *kpd);
87 static caddr_t   segkp_get_internal(struct seg *seg, size_t len, uint_t flags,
88                                   struct segkp_data **tkpd, struct anon_map *amp);
89 static void      segkp_release_internal(struct seg *seg,
90                                       struct segkp_data *kpd, size_t len);
91 static int       segkp_unlock(struct hat *hat, struct seg *seg, caddr_t vaddr,
92                               size_t len, struct segkp_data *kpd, uint_t flags);
93 static int       segkp_load(struct hat *hat, struct seg *seg, caddr_t vaddr,
94                               size_t len, struct segkp_data *kpd, uint_t flags);
95 static struct segkp_data *segkp_find(struct seg *seg, caddr_t vaddr);
96 static int       segkp_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp);
97 static lgrp_mem_policy_info_t *segkp_getpolicy(struct seg *seg,
98                                                caddr_t addr);
99 static int       segkp_capable(struct seg *seg, segcapability_t capability);

101 /*
102  * Lock used to protect the hash table(s) and caches.
103  */
104 static kmutex_t segkp_lock;

106 /*
107  * The segkp caches
108  */
109 static struct segkp_cache segkp_cache[SEGKP_MAX_CACHE];

111 /*
112  * When there are fewer than red_minavail bytes left on the stack,
113  * segkp_map_red() will map in the redzone (if called). 5000 seems
114  * to work reasonably well...
115  */
116 long          red_minavail = 5000;

118 /*
119  * will be set to 1 for 32 bit x86 systems only, in startup.c
120  */
121 int           segkp_fromheap = 0;
122 ulong_t      *segkp_bitmap;

124 /*
125  * If segkp_map_red() is called with the redzone already mapped and
126  * with less than RED_DEEP_THRESHOLD bytes available on the stack,
127  * then the stack situation has become quite serious; if much more stack

```

```
128 * is consumed, we have the potential of scrogging the next thread/LWP
129 * structure. To help debug the "can't happen" panics which may
130 * result from this condition, we record hrestime and the calling thread
131 * in red_deep_hires and red_deep_thread respectively.
132 */
133 #define RED_DEEP_THRESHOLD      2000

135 hrtime_t      red_deep_hires;
136 kthread_t     *red_deep_thread;

138 uint32_t      red_nmapped;
139 uint32_t      red_closest = UINT_MAX;
140 uint32_t      red_ndoubles;

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

145 static struct seg_ops segkp_ops = {
146     .fault      = segkp_fault,
147     .checkprot  = segkp_checkprot,
148     .kluster    = segkp_kluster,
149     .dump       = segkp_dump,
150     .pagelock   = segkp_pagelock,
151     .getmemid   = segkp_getmemid,
152     .getpolicy  = segkp_getpolicy,
153     .capable    = segkp_capable,
154     .inherit    = seg_inherit_notsup,
154 };
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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */
27 /*
28 * Kernel Physical Mapping (kpm) segment driver (segkpm).
29 *
30 * This driver delivers along with the hat_kpm* interfaces an alternative
31 * mechanism for kernel mappings within the 64-bit Solaris operating system,
32 * which allows the mapping of all physical memory into the kernel address
33 * space at once. This is feasible in 64 bit kernels, e.g. for Ultrasparc II
34 * and beyond processors, since the available VA range is much larger than
35 * possible physical memory. Momentarily all physical memory is supported,
36 * that is represented by the list of memory segments (memsegs).
37 *
38 * Segkpm mappings have also very low overhead and large pages are used
39 * (when possible) to minimize the TLB and TSB footprint. It is also
40 * extensible for other than Sparc architectures (e.g. AMD64). Main
41 * advantage is the avoidance of the TLB-shutdown X-calls, which are
42 * normally needed when a kernel (global) mapping has to be removed.
43 *
44 * First example of a kernel facility that uses the segkpm mapping scheme
45 * is seg_map, where it is used as an alternative to hat_memload().
46 * See also hat layer for more information about the hat_kpm* routines.
47 * The kpm facility can be turned off at boot time (e.g. /etc/system).
48 */
50 #include <sys/types.h>
51 #include <sys/param.h>
52 #include <sys/sysmacros.h>
53 #include <sys/unistd.h>
54 #include <sys/vnode.h>
55 #include <sys/cmn_err.h>
56 #include <sys/debug.h>
57 #include <sys/thread.h>
58 #include <sys/cpuvar.h>
59 #include <sys/bitmap.h>
60 #include <sys/atomic.h>
61 #include <sys/lgrp.h>

```

```

63 #include <vm/seg_kmem.h>
64 #include <vm/seg_kpm.h>
65 #include <vm/hat.h>
66 #include <vm/as.h>
67 #include <vm/seg.h>
68 #include <vm/page.h>
69
70 /*
71 * Global kpm controls.
72 * See also platform and mmu specific controls.
73 *
74 * kpm_enable -- global on/off switch for segkpm.
75 * . Set by default on 64bit platforms that have kpm support.
76 * . Will be disabled from platform layer if not supported.
77 * . Can be disabled via /etc/system.
78 *
79 * kpm_smallpages -- use only regular/system pagesize for kpm mappings.
80 * . Can be useful for critical debugging of kpm clients.
81 * . Set to zero by default for platforms that support kpm large pages.
82 * . The use of kpm large pages reduces the footprint of kpm meta data
83 * and has all the other advantages of using large pages (e.g TLB
84 * miss reduction).
85 * . Set by default for platforms that don't support kpm large pages or
86 * where large pages cannot be used for other reasons (e.g. there are
87 * only few full associative TLB entries available for large pages).
88 *
89 * segmap_kpm -- separate on/off switch for segmap using segkpm:
90 * . Set by default.
91 * . Will be disabled when kpm_enable is zero.
92 * . Will be disabled when MAXBSIZE != PAGESIZE.
93 * . Can be disabled via /etc/system.
94 *
95 */
96 int kpm_enable = 1;
97 int kpm_smallpages = 0;
98 int segmap_kpm = 1;
99
100 /*
101 * Private seg op routines.
102 */
103 faultcode_t segkpm_fault(struct hat *hat, struct seg *seg, caddr_t addr,
104                          size_t len, enum fault_type type, enum seg_rw rw);
105 static void segkpm_dump(struct seg *);
106 static int segkpm_pagelock(struct seg *seg, caddr_t addr, size_t len,
107                            struct page ***page, enum lock_type type,
108                            enum seg_rw rw);
109 static int segkpm_capable(struct seg *, segcapability_t);
110
111 static struct seg_ops segkpm_ops = {
112     .fault = segkpm_fault,
113     .dump = segkpm_dump,
114     .pagelock = segkpm_pagelock,
115     .capable = segkpm_capable,
116     .inherit = seg_inherit_notsup,
117 };
118 #ifndef SEGKPM_SUPPORT
119 #if 0
120 #error FIXME: define nop
121     .dup = nop,
122     .unmap = nop,
123     .free = nop,
124     .faulta = nop,
125     .setprot = nop,
126     .checkprot = nop,
127     .kluster = nop,
128     .sync = nop,

```



new/usr/src/uts/common/vm/seg\_kpm.c

3

```
127     .incore      = nop,  
128     .lockop     = nop,  
129     .getprot    = nop,  
130     .getoffset  = nop,  
131     .gettype    = nop,  
132     .getvp      = nop,  
133     .advise     = nop,  
134     .setpagesize = nop,  
135     .getmemid   = nop,  
136     .getpolicy  = nop,  
137 #endif  
138 };  
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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T      */
27 /*      All Rights Reserved      */

29 /*
30 * Portions of this source code were derived from Berkeley 4.3 BSD
31 * under license from the Regents of the University of California.
32 */

34 /*
35 * VM - generic vnode mapping segment.
36 *
37 * The segmap driver is used only by the kernel to get faster (than seg_vn)
38 * mappings [lower routine overhead; more persistent cache] to random
39 * vnode/offsets. Note than the kernel may (and does) use seg_vn as well.
40 */

42 #include <sys/types.h>
43 #include <sys/t_lock.h>
44 #include <sys/param.h>
45 #include <sys/sysmacros.h>
46 #include <sys/buf.h>
47 #include <sys/system.h>
48 #include <sys/vnode.h>
49 #include <sys/mman.h>
50 #include <sys/errno.h>
51 #include <sys/cred.h>
52 #include <sys/kmem.h>
53 #include <sys/vtrace.h>
54 #include <sys/cmn_err.h>
55 #include <sys/debug.h>
56 #include <sys/thread.h>
57 #include <sys/dumphdr.h>
58 #include <sys/bitmap.h>
59 #include <sys/lgrp.h>

61 #include <vm/seg_kmem.h>

```

```

62 #include <vm/hat.h>
63 #include <vm/as.h>
64 #include <vm/seg.h>
65 #include <vm/seg_kpm.h>
66 #include <vm/seg_map.h>
67 #include <vm/page.h>
68 #include <vm/pvn.h>
69 #include <vm/rm.h>

71 /*
72  * Private seg op routines.
73 */
74 static void      segmap_free(struct seg *seg);
75 faultcode_t segmap_fault(struct hat *hat, struct seg *seg, caddr_t addr,
76                          size_t len, enum fault_type type, enum seg_rw rw);
77 static faultcode_t segmap_faulta(struct seg *seg, caddr_t addr);
78 static int      segmap_checkprot(struct seg *seg, caddr_t addr, size_t len,
79                                  uint_t prot);
80 static int      segmap_kluster(struct seg *seg, caddr_t addr, ssize_t);
81 static int      segmap_getprot(struct seg *seg, caddr_t addr, size_t len,
82                                uint_t *protv);
83 static u_offset_t segmap_getoffset(struct seg *seg, caddr_t addr);
84 static int      segmap_gettype(struct seg *seg, caddr_t addr);
85 static int      segmap_getvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
86 static void      segmap_dump(struct seg *seg);
87 static int      segmap_pagelock(struct seg *seg, caddr_t addr, size_t len,
88                                  struct page ***ppp, enum lock_type type,
89                                  enum seg_rw rw);
90 static int      segmap_getmemid(struct seg *seg, caddr_t addr, memid_t *memidp);
91 static lgrp_mem_policy_info_t *segmap_getpolicy(struct seg *seg,
92                                                  caddr_t addr);
93 static int      segmap_capable(struct seg *seg, segcapability_t capability);

95 /* segkpm support */
96 static caddr_t segmap_pagecreate_kpm(struct seg *, vnode_t *, u_offset_t,
97                                     struct smap *, enum seg_rw);
98 struct smap      *get_smap_kpm(caddr_t, page_t **);

100 static struct seg_ops segmap_ops = {
101     .free      = segmap_free,
102     .fault     = segmap_fault,
103     .faulta   = segmap_faulta,
104     .checkprot = segmap_checkprot,
105     .kluster  = segmap_kluster,
106     .getprot  = segmap_getprot,
107     .getoffset = segmap_getoffset,
108     .gettype  = segmap_gettype,
109     .getvp    = segmap_getvp,
110     .dump     = segmap_dump,
111     .pagelock = segmap_pagelock,
112     .getmemid = segmap_getmemid,
113     .getpolicy = segmap_getpolicy,
114     .capable  = segmap_capable,
115     .inherit  = seg_inherit_notsup,
116 };

```

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
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
23 */

25 #include <sys/param.h>
26 #include <sys/user.h>
27 #include <sys/mman.h>
28 #include <sys/kmem.h>
29 #include <sys/sysmacros.h>
30 #include <sys/cmn_err.h>
31 #include <sys/system.h>
32 #include <sys/tuneable.h>
33 #include <vm/hat.h>
34 #include <vm/seg.h>
35 #include <vm/as.h>
36 #include <vm/anon.h>
37 #include <vm/page.h>
38 #include <sys/buf.h>
39 #include <sys/swap.h>
40 #include <sys/atomic.h>
41 #include <vm/seg_spt.h>
42 #include <sys/debug.h>
43 #include <sys/vtrace.h>
44 #include <sys/shm.h>
45 #include <sys/shm_impl.h>
46 #include <sys/lgrp.h>
47 #include <sys/vmsystem.h>
48 #include <sys/policy.h>
49 #include <sys/project.h>
50 #include <sys/tnf_probe.h>
51 #include <sys/zone.h>

53 #define SEGSPtADDR      (caddr_t)0x0

55 /*
56  * # pages used for spt
57  */
58 size_t spt_used;

60 /*
61  * segspt_minfree is the memory left for system after ISM

```

```

62  * locked its pages; it is set up to 5% of availrmem in
63  * sptcreate when ISM is created. ISM should not use more
64  * than ~90% of availrmem; if it does, then the performance
65  * of the system may decrease. Machines with large memories may
66  * be able to use up more memory for ISM so we set the default
67  * segspt_minfree to 5% (which gives ISM max 95% of availrmem.
68  * If somebody wants even more memory for ISM (risking hanging
69  * the system) they can patch the segspt_minfree to smaller number.
70  */
71 pgcnt_t segspt_minfree = 0;

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

79 struct seg_ops segspt_ops = {
80     .unmap      = segspt_unmap,
81     .free       = segspt_free,
82     .getpolicy  = segspt_getpolicy,
83     .inherit    = seg_inherit_notsup,
84 };

85 static int segspt_shmdup(struct seg *seg, struct seg *newseg);
86 static int segspt_shmunmap(struct seg *seg, caddr_t raddr, size_t ssize);
87 static void segspt_shmfree(struct seg *seg);
88 static faultcode_t segspt_shmfault(struct hat *hat, struct seg *seg,
89     caddr_t addr, size_t len, enum fault_type type, enum seg_rw rw);
90 static faultcode_t segspt_shmfaulta(struct seg *seg, caddr_t addr);
91 static int segspt_shmsetprot(register struct seg *seg, register caddr_t addr,
92     register size_t len, register uint_t prot);
93 static int segspt_shmcheckprot(struct seg *seg, caddr_t addr, size_t size,
94     uint_t prot);
95 static int segspt_shmkluster(struct seg *seg, caddr_t addr, ssize_t delta);
96 static size_t segspt_shmincore(struct seg *seg, caddr_t addr, size_t len,
97     register char *vec);
98 static int segspt_shmsync(struct seg *seg, register caddr_t addr, size_t len,
99     int attr, uint_t flags);
100 static int segspt_shmlockop(struct seg *seg, caddr_t addr, size_t len,
101     int attr, int op, ulong_t *lockmap, size_t pos);
102 static int segspt_shmgetprot(struct seg *seg, caddr_t addr, size_t len,
103     uint_t *protv);
104 static u_offset_t segspt_shmgetoffset(struct seg *seg, caddr_t addr);
105 static int segspt_shmgettype(struct seg *seg, caddr_t addr);
106 static int segspt_shmgetvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
107 static int segspt_shmadvise(struct seg *seg, caddr_t addr, size_t len,
108     uint_t behav);
109 static void segspt_shmdump(struct seg *seg);
110 static int segspt_shmpagelock(struct seg *, caddr_t, size_t,
111     struct page ***, enum lock_type, enum seg_rw);
112 static int segspt_shmsetpgsz(struct seg *, caddr_t, size_t, uint_t);
113 static int segspt_shmgetmemid(struct seg *, caddr_t, memid_t *);
114 static lgrp_mem_policy_info_t *segspt_shmgetpolicy(struct seg *, caddr_t);
115 static int segspt_shmcapable(struct seg *, segcapability_t);

117 struct seg_ops segspt_shmops = {
118     .dup        = segspt_shmdup,
119     .unmap     = segspt_shmunmap,
120     .free      = segspt_shmfree,
121     .fault     = segspt_shmfault,
122     .faulta   = segspt_shmfaulta,
123     .setprot  = segspt_shmsetprot,
124     .checkprot = segspt_shmcheckprot,
125     .kluster  = segspt_shmkluster,
126     .sync     = segspt_shmsync,

```

```
127     .incore      = segspt_shmincore,  
128     .lockop     = segspt_shmlockop,  
129     .getprot    = segspt_shmgetprot,  
130     .getoffset  = segspt_shmgetoffset,  
131     .gettype    = segspt_shmgettype,  
132     .getvp      = segspt_shmgetvp,  
133     .advise     = segspt_shmadvise,  
134     .dump       = segspt_shmdump,  
135     .pagelock   = segspt_shmpagelock,  
136     .setpagesize = segspt_shmsetpgsz,  
137     .getmemid   = segspt_shmgetmemid,  
138     .getpolicy  = segspt_shmgetpolicy,  
139     .capable    = segspt_shmcapable,  
141     .inherit    = seg_inherit_notsup,  
140 };  
_____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_____

2157 /*
2158 * Cache control operations over the interval [addr, addr + size) in
2159 * address space "as".
2160 */
2161 /*ARGSUSED*/
2162 int
2163 as_ctl(struct as *as, caddr_t addr, size_t size, int func, int attr,
2164        uintptr_t arg, ulong_t *lock_map, size_t pos)
2165 {
2166     struct seg *seg;          /* working segment */
2167     caddr_t raddr;           /* rounded down addr */
2168     caddr_t intraddr;        /* saved initial rounded down addr */
2169     size_t rsize;            /* rounded up size */
2170     size_t initsize;         /* saved initial rounded up size */
2171     size_t ssize;           /* size of seg */
2172     int error = 0;           /* result */
2173     size_t mlock_size;       /* size of bitmap */
2174     ulong_t *mlock_map;     /* pointer to bitmap used */
2175                             /* to represent the locked */
2176                             /* pages. */
2177 retry:
2178     if (error == IE_RETRY)
2179         AS_LOCK_ENTER(as, &as->a_lock, RW_WRITER);
2180     else
2181         AS_LOCK_ENTER(as, &as->a_lock, RW_READER);
2182
2183     /*
2184     * If these are address space lock/unlock operations, loop over
2185     * all segments in the address space, as appropriate.
2186     */
2187     if (func == MC_LOCKAS) {
2188         size_t npages, idx;
2189         size_t rlen = 0;     /* rounded as length */
2190
2191         idx = pos;
2192
2193         if (arg & MCL_FUTURE) {
2194             mutex_enter(&as->a_contents);
2195             AS_SETPGLCK(as);
2196             mutex_exit(&as->a_contents);
2197         }
2198         if ((arg & MCL_CURRENT) == 0) {
2199             AS_LOCK_EXIT(as, &as->a_lock);
2200             return (0);
2201         }
2202
2203         seg = AS_SEGFIRST(as);
2204         if (seg == NULL) {
2205             AS_LOCK_EXIT(as, &as->a_lock);
2206             return (0);
2207         }
2208
2209         do {
2210             raddr = (caddr_t)((uintptr_t)seg->s_base &
2211                               (uintptr_t)PAGEMASK);
2212             rlen += (((uintptr_t)(seg->s_base + seg->s_size) +
2213                     PAGEOFFSET) & PAGEMASK) - (uintptr_t)raddr;
2214         } while ((seg = AS_SEGNEXT(as, seg)) != NULL);

```

```

2216     mlock_size = BT_BITOUL(btopr(rlen));
2217     if ((mlock_map = (ulong_t *)kmem_zalloc(mlock_size *
2218                                             sizeof(ulong_t), KM_NOSLEEP)) == NULL) {
2219         AS_LOCK_EXIT(as, &as->a_lock);
2220         return (EAGAIN);
2221     }
2222
2223     for (seg = AS_SEGFIRST(as); seg; seg = AS_SEGNEXT(as, seg)) {
2224         error = segop_lockop(seg, seg->s_base,
2225                               seg->s_size, attr, MC_LOCK, mlock_map, pos);
2226         if (error != 0)
2227             break;
2228         pos += seg_pages(seg);
2229     }
2230
2231     if (error) {
2232         for (seg = AS_SEGFIRST(as); seg != NULL;
2233              seg = AS_SEGNEXT(as, seg)) {
2234
2235             raddr = (caddr_t)((uintptr_t)seg->s_base &
2236                               (uintptr_t)PAGEMASK);
2237             npages = seg_pages(seg);
2238             as_segunlock(seg, raddr, attr, mlock_map,
2239                          idx, npages);
2240             idx += npages;
2241         }
2242     }
2243
2244     kmem_free(mlock_map, mlock_size * sizeof(ulong_t));
2245     AS_LOCK_EXIT(as, &as->a_lock);
2246     goto lockerr;
2247 } else if (func == MC_UNLOCKAS) {
2248     mutex_enter(&as->a_contents);
2249     AS_CLRPLCK(as);
2250     mutex_exit(&as->a_contents);
2251
2252     for (seg = AS_SEGFIRST(as); seg; seg = AS_SEGNEXT(as, seg)) {
2253         error = segop_lockop(seg, seg->s_base,
2254                               seg->s_size, attr, MC_UNLOCK, NULL, 0);
2255         if (error != 0)
2256             break;
2257     }
2258
2259     AS_LOCK_EXIT(as, &as->a_lock);
2260     goto lockerr;
2261 }
2262
2263 /*
2264 * Normalize addresses and sizes.
2265 */
2266     intraddr = raddr = (caddr_t)((uintptr_t)addr & (uintptr_t)PAGEMASK);
2267     initsize = rsize = (((size_t)(addr + size) + PAGEOFFSET) & PAGEMASK) -
2268                    (size_t)raddr;
2269
2270     if (raddr + rsize < raddr) { /* check for wraparound */
2271         AS_LOCK_EXIT(as, &as->a_lock);
2272         return (ENOMEM);
2273     }
2274
2275     /*
2276     * Get initial segment.
2277     */
2278     if ((seg = as_segat(as, raddr)) == NULL) {
2279         AS_LOCK_EXIT(as, &as->a_lock);
2280         return (ENOMEM);
2281     }

```

```

2283     if (func == MC_LOCK) {
2284         mlock_size = BT_BITOUL(btopr(rsize));
2285         if ((mlock_map = (ulong_t *)kmem_zalloc(mlock_size *
2286             sizeof(ulong_t), KM_NOSLEEP)) == NULL) {
2287             AS_LOCK_EXIT(as, &as->a_lock);
2288             return (EAGAIN);
2289         }
2290     }
2291
2292     /*
2293     * Loop over all segments.  If a hole in the address range is
2294     * discovered, then fail.  For each segment, perform the appropriate
2295     * control operation.
2296     */
2297     while (rsize != 0) {
2298
2299         /*
2300         * Make sure there's no hole, calculate the portion
2301         * of the next segment to be operated over.
2302         */
2303         if (raddr >= seg->s_base + seg->s_size) {
2304             seg = AS_SEGNEXT(as, seg);
2305             if (seg == NULL || raddr != seg->s_base) {
2306                 if (func == MC_LOCK) {
2307                     as_unlockerr(as, attr, mlock_map,
2308                         intraddr, initsize - rsize);
2309                     kmem_free(mlock_map,
2310                         mlock_size * sizeof(ulong_t));
2311                 }
2312                 AS_LOCK_EXIT(as, &as->a_lock);
2313                 return (ENOMEM);
2314             }
2315         }
2316         if ((raddr + rsize) > (seg->s_base + seg->s_size))
2317             ssize = seg->s_base + seg->s_size - raddr;
2318         else
2319             ssize = rsize;
2320
2321         /*
2322         * Dispatch on specific function.
2323         */
2324         switch (func) {
2325
2326         /*
2327         * Synchronize cached data from mappings with backing
2328         * objects.
2329         */
2330         case MC_SYNC:
2331             if (error = segop_sync(seg, raddr, ssize,
2332                 attr, (uint_t)arg)) {
2333                 AS_LOCK_EXIT(as, &as->a_lock);
2334                 return (error);
2335             }
2336             break;
2337
2338         /*
2339         * Lock pages in memory.
2340         */
2341         case MC_LOCK:
2342             if (error = segop_lockop(seg, raddr, ssize,
2343                 attr, func, mlock_map, pos)) {
2344                 as_unlockerr(as, attr, mlock_map, intraddr,
2345                     initsize - rsize + ssize);
2346                 kmem_free(mlock_map, mlock_size *
2347                     sizeof(ulong_t));

```

```

2348             AS_LOCK_EXIT(as, &as->a_lock);
2349             goto lockerr;
2350         }
2351         break;
2352
2353     /*
2354     * Unlock mapped pages.
2355     */
2356     case MC_UNLOCK:
2357         (void) segop_lockop(seg, raddr, ssize, attr, func,
2358             (ulong_t *)NULL, (size_t)NULL);
2359         break;
2360
2361     /*
2362     * Store VM advise for mapped pages in segment layer.
2363     */
2364     case MC_ADVISE:
2365         error = segop_advise(seg, raddr, ssize, (uint_t)arg);
2366
2367         /*
2368         * Check for regular errors and special retry error
2369         */
2370         if (error) {
2371             if (error == IE_RETRY) {
2372                 /*
2373                 * Need to acquire writers lock, so
2374                 * have to drop readers lock and start
2375                 * all over again
2376                 */
2377                 AS_LOCK_EXIT(as, &as->a_lock);
2378                 goto retry;
2379             } else if (error == IE_REATTACH) {
2380                 /*
2381                 * Find segment for current address
2382                 * because current segment just got
2383                 * split or concatenated
2384                 */
2385                 seg = as_segat(as, raddr);
2386                 if (seg == NULL) {
2387                     AS_LOCK_EXIT(as, &as->a_lock);
2388                     return (ENOMEM);
2389                 }
2390             } else {
2391                 /*
2392                 * Regular error
2393                 */
2394                 AS_LOCK_EXIT(as, &as->a_lock);
2395                 return (error);
2396             }
2397         }
2398         break;
2399
2400     case MC_INHERIT_ZERO:
2401         error = segop_inherit(seg, raddr, ssize, SEGP_INH_ZERO);
2402         if (seg->s_ops->inherit == NULL) {
2403             error = ENOTSUP;
2404         } else {
2405             error = segop_inherit(seg, raddr, ssize,
2406                 SEGP_INH_ZERO);
2407         }
2408         if (error != 0) {
2409             AS_LOCK_EXIT(as, &as->a_lock);
2410             return (error);
2411         }
2412         break;

```

```
2408             /*
2409              * Can't happen.
2410              */
2411             default:
2412                 panic("as_ctl: bad operation %d", func);
2413                 /*NOTREACHED*/
2414             }
2415
2416             rsize -= ssize;
2417             raddr += ssize;
2418         }
2419
2420         if (func == MC_LOCK)
2421             kmem_free(mlock_map, mlock_size * sizeof (ulong_t));
2422         AS_LOCK_EXIT(as, &as->a_lock);
2423         return (0);
2424 lockerr:
2425
2426         /*
2427          * If the lower levels returned EDEADLK for a segment lockop,
2428          * it means that we should retry the operation. Let's wait
2429          * a bit also to let the deadlock causing condition clear.
2430          * This is part of a gross hack to work around a design flaw
2431          * in the ufs/sds logging code and should go away when the
2432          * logging code is re-designed to fix the problem. See bug
2433          * 4125102 for details of the problem.
2434          */
2435         if (error == EDEADLK) {
2436             delay(deadlk_wait);
2437             error = 0;
2438             goto retry;
2439         }
2440         return (error);
2441     }
    unchanged_portion_omitted
```

new/usr/src/uts/common/vm/vm\_seg.c

1

```
*****
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_

1856 /*
1857  * General not supported function for segop_inherit
1858  */
1859 /* ARGSUSED */
1860 int
1861 seg_inherit_notsup(struct seg *seg, caddr_t addr, size_t len, uint_t op)
1862 {
1863     return (ENOTSUP);
1864 }

1866 /*
1867  * segop wrappers
1868  */
1869 int
1870 segop_dup(struct seg *seg, struct seg *new)
1871 {
1872     VERIFY3P(seg->s_ops->dup, !=, NULL);
1873
1874     return (seg->s_ops->dup(seg, new));
1875 }
_____unchanged_portion_omitted_
```



new/usr/src/uts/i86xpv/vm/seg\_mf.c

1

```
*****
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\_\_\_\_\_

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

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_