

```

*****
10007 Fri May 8 18:03:58 2015
new/usr/src/uts/common/vm/seg.h
instead using SEGOP_* macros, define full-fledged segop_* functions
This will allow us to do some sanity checking or even implement stub
functionality in one place instead of duplicating it wherever these wrappers
are used.
*****
_____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_pinsert_check(struct seg *seg, struct anon_map *amp,
175                             caddr_t addr, size_t len, uint_t flags);
176 extern int seg_pinsert(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_pinsert and seg_pinsert_check functions.
199  */
200 #define SEGP_SUCCESS 0 /* seg_pinsert() succeeded */
201 #define SEGP_FAIL 1 /* seg_pinsert() failed */

```

```

203 /* Page status bits for segop_incore */
204 #define SEG_PAGE_INCORE 0x01 /* VA has a page backing it */
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 SEGOP_DUP(s, n) (*(s)->s_ops->dup)((s), (n))
213 #define SEGOP_UNMAP(s, a, l) (*(s)->s_ops->unmap)((s), (a), (l))
214 #define SEGOP_FREE(s) (*(s)->s_ops->free)((s))
215 #define SEGOP_FAULT(h, s, a, l, t, rw) \
216     (*(s)->s_ops->fault)((h), (s), (a), (l), (t), (rw))
217 #define SEGOP_FAULTA(s, a) (*(s)->s_ops->faulta)((s), (a))
218 #define SEGOP_SETPROT(s, a, l, p) (*(s)->s_ops->setprot)((s), (a), (l), (p))
219 #define SEGOP_CHECKPROT(s, a, l, p) (*(s)->s_ops->checkprot)((s), (a), (l), (p))
220 #define SEGOP_KLUSTER(s, a, d) (*(s)->s_ops->kluster)((s), (a), (d))
221 #define SEGOP_SYNC(s, a, l, atr, f) \
222     (*(s)->s_ops->sync)((s), (a), (l), (atr), (f))
223 #define SEGOP_INCORE(s, a, l, v) (*(s)->s_ops->incore)((s), (a), (l), (v))
224 #define SEGOP_LOCKOP(s, a, l, atr, op, b, p) \
225     (*(s)->s_ops->lockop)((s), (a), (l), (atr), (op), (b), (p))
226 #define SEGOP_GETPROT(s, a, l, p) (*(s)->s_ops->getprot)((s), (a), (l), (p))
227 #define SEGOP_GETOFFSET(s, a) (*(s)->s_ops->getoffset)((s), (a))
228 #define SEGOP_GETTYPE(s, a) (*(s)->s_ops->gettype)((s), (a))
229 #define SEGOP_GETVPP(s, a, vpp) (*(s)->s_ops->getvp)((s), (a), (vpp))
230 #define SEGOP_ADVISE(s, a, l, b) (*(s)->s_ops->advise)((s), (a), (l), (b))
231 #define SEGOP_DUMP(s) (*(s)->s_ops->dump)((s))
232 #define SEGOP_PAGELOCK(s, a, l, p, t, rw) \
233     (*(s)->s_ops->pagelock)((s), (a), (l), (p), (t), (rw))
234 #define SEGOP_SETPAGESIZE(s, a, l, szc) \
235     (*(s)->s_ops->setpagesize)((s), (a), (l), (szc))
236 #define SEGOP_GETMEMID(s, a, mp) (*(s)->s_ops->getmemid)((s), (a), (mp))
237 #define SEGOP_GETPOLICY(s, a) (*(s)->s_ops->getpolicy)((s), (a))
238 #define SEGOP_CAPABLE(s, c) (*(s)->s_ops->capable)((s), (c))
239 #define SEGOP_INHERIT(s, a, l, b) (*(s)->s_ops->inherit)((s), (a), (l), (b))

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);

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

```

```
240 #ifdef VMDEBUG
242 uint_t seg_page(struct seg *, caddr_t);
243 uint_t seg_pages(struct seg *);
245 #endif /* VMDEBUG */
247 boolean_t seg_can_change_zones(struct seg *);
248 size_t seg_swresv(struct seg *);
250 /* segop wrappers */
251 int segop_dup(struct seg *, struct seg *);
252 int segop_unmap(struct seg *, caddr_t, size_t);
253 void segop_free(struct seg *);
254 faultcode_t segop_fault(struct hat *, struct seg *, caddr_t, size_t, enum fault_
255 faultcode_t segop_faulta(struct seg *, caddr_t);
256 int segop_setprot(struct seg *, caddr_t, size_t, uint_t);
257 int segop_checkprot(struct seg *, caddr_t, size_t, uint_t);
258 int segop_kluster(struct seg *, caddr_t, ssize_t);
259 int segop_sync(struct seg *, caddr_t, size_t, int, uint_t);
260 size_t segop_incore(struct seg *, caddr_t, size_t, char *);
261 int segop_lockop(struct seg *, caddr_t, size_t, int, int, ulong_t *, size_t );
262 int segop_getprot(struct seg *, caddr_t, size_t, uint_t *);
263 u_offset_t segop_getoffset(struct seg *, caddr_t);
264 int segop_gettype(struct seg *, caddr_t);
265 int segop_getvp(struct seg *, caddr_t, struct vnode **);
266 int segop_advise(struct seg *, caddr_t, size_t, uint_t);
267 void segop_dump(struct seg *);
268 int segop_pagelock(struct seg *, caddr_t, size_t, struct page ***, enum lock_typ
269 int segop_setpagesize(struct seg *, caddr_t, size_t, uint_t);
270 int segop_getmemid(struct seg *, caddr_t, memid_t *);
271 struct lgrp_mem_policy_info *segop_getpolicy(struct seg *, caddr_t);
272 int segop_capable(struct seg *, segcapability_t);
273 int segop_inherit(struct seg *, caddr_t, size_t, uint_t);
274 #endif /* ! codereview */
276 #endif /* _KERNEL */
278 #ifdef __cplusplus
279 }
280 #endif
282 #endif /* _VM_SEG_H */
```

```

*****
55290 Fri May 8 18:03:58 2015
new/usr/src/uts/common/vm/vm_seg.c
instead using SEGOP_* macros, define full-fledged segop_* functions
This will allow us to do some sanity checking or even implement stub
functionality in one place instead of duplicating it wherever these wrappers
are used.
*****
_____unchanged_portion_omitted_____

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

1874     return (seg->s_ops->dup(seg, new));
1875 }

1877 int
1878 segop_unmap(struct seg *seg, caddr_t addr, size_t len)
1879 {
1880     VERIFY3P(seg->s_ops->unmap, !=, NULL);

1882     return (seg->s_ops->unmap(seg, addr, len));
1883 }

1885 void
1886 segop_free(struct seg *seg)
1887 {
1888     VERIFY3P(seg->s_ops->free, !=, NULL);

1890     seg->s_ops->free(seg);
1891 }

1893 faultcode_t
1894 segop_fault(struct hat *hat, struct seg *seg, caddr_t addr, size_t len,
1895             enum fault_type type, enum seg_rw rw)
1896 {
1897     VERIFY3P(seg->s_ops->fault, !=, NULL);

1899     return (seg->s_ops->fault(hat, seg, addr, len, type, rw));
1900 }

1902 faultcode_t
1903 segop_faulta(struct seg *seg, caddr_t addr)
1904 {
1905     VERIFY3P(seg->s_ops->faulta, !=, NULL);

1907     return (seg->s_ops->faulta(seg, addr));
1908 }

1910 int
1911 segop_setprot(struct seg *seg, caddr_t addr, size_t len, uint_t prot)
1912 {
1913     VERIFY3P(seg->s_ops->setprot, !=, NULL);

1915     return (seg->s_ops->setprot(seg, addr, len, prot));
1916 }

1918 int
1919 segop_checkprot(struct seg *seg, caddr_t addr, size_t len, uint_t prot)
1920 {
1921     VERIFY3P(seg->s_ops->checkprot, !=, NULL);

```

```

1923     return (seg->s_ops->checkprot(seg, addr, len, prot));
1924 }

1926 int
1927 segop_kluster(struct seg *seg, caddr_t addr, ssize_t d)
1928 {
1929     VERIFY3P(seg->s_ops->kluster, !=, NULL);

1931     return (seg->s_ops->kluster(seg, addr, d));
1932 }

1934 int
1935 segop_sync(struct seg *seg, caddr_t addr, size_t len, int atr, uint_t f)
1936 {
1937     VERIFY3P(seg->s_ops->sync, !=, NULL);

1939     return (seg->s_ops->sync(seg, addr, len, atr, f));
1940 }

1942 size_t
1943 segop_incore(struct seg *seg, caddr_t addr, size_t len, char *v)
1944 {
1945     VERIFY3P(seg->s_ops->incore, !=, NULL);

1947     return (seg->s_ops->incore(seg, addr, len, v));
1948 }

1950 int
1951 segop_lockop(struct seg *seg, caddr_t addr, size_t len, int atr, int op,
1952             ulong_t *b, size_t p)
1953 {
1954     VERIFY3P(seg->s_ops->lockop, !=, NULL);

1956     return (seg->s_ops->lockop(seg, addr, len, atr, op, b, p));
1957 }

1959 int
1960 segop_getprot(struct seg *seg, caddr_t addr, size_t len, uint_t *p)
1961 {
1962     VERIFY3P(seg->s_ops->getprot, !=, NULL);

1964     return (seg->s_ops->getprot(seg, addr, len, p));
1965 }

1967 u_offset_t
1968 segop_getoffset(struct seg *seg, caddr_t addr)
1969 {
1970     VERIFY3P(seg->s_ops->getoffset, !=, NULL);

1972     return (seg->s_ops->getoffset(seg, addr));
1973 }

1975 int
1976 segop_gettype(struct seg *seg, caddr_t addr)
1977 {
1978     VERIFY3P(seg->s_ops->gettype, !=, NULL);

1980     return (seg->s_ops->gettype(seg, addr));
1981 }

1983 int
1984 segop_getvp(struct seg *seg, caddr_t addr, struct vnode **vpp)
1985 {
1986     VERIFY3P(seg->s_ops->getvp, !=, NULL);

```

```

1988     return (seg->s_ops->getvp(seg, addr, vpp));
1989 }

1991 int
1992 segop_advise(struct seg *seg, caddr_t addr, size_t len, uint_t b)
1993 {
1994     VERIFY3P(seg->s_ops->advise, !=, NULL);

1996     return (seg->s_ops->advise(seg, addr, len, b));
1997 }

1999 void
2000 segop_dump(struct seg *seg)
2001 {
2002     VERIFY3P(seg->s_ops->dump, !=, NULL);

2004     seg->s_ops->dump(seg);
2005 }

2007 int
2008 segop_pagelock(struct seg *seg, caddr_t addr, size_t len, struct page ***page,
2009     enum lock_type type, enum seg_rw rw)
2010 {
2011     VERIFY3P(seg->s_ops->pagelock, !=, NULL);

2013     return (seg->s_ops->pagelock(seg, addr, len, page, type, rw));
2014 }

2016 int
2017 segop_setpagesize(struct seg *seg, caddr_t addr, size_t len, uint_t szc)
2018 {
2019     VERIFY3P(seg->s_ops->setpagesize, !=, NULL);

2021     return (seg->s_ops->setpagesize(seg, addr, len, szc));
2022 }

2024 int
2025 segop_getmemid(struct seg *seg, caddr_t addr, memid_t *mp)
2026 {
2027     VERIFY3P(seg->s_ops->getmemid, !=, NULL);

2029     return (seg->s_ops->getmemid(seg, addr, mp));
2030 }

2032 struct lgrp_mem_policy_info *
2033 segop_getpolicy(struct seg *seg, caddr_t addr)
2034 {
2035     if (seg->s_ops->getpolicy == NULL)
2036         return (NULL);

2038     return (seg->s_ops->getpolicy(seg, addr));
2039 }

2041 int
2042 segop_capable(struct seg *seg, segcapability_t cap)
2043 {
2044     VERIFY3P(seg->s_ops->capable, !=, NULL);

2046     return (seg->s_ops->capable(seg, cap));
2047 }

2049 int
2050 segop_inherit(struct seg *seg, caddr_t addr, size_t len, uint_t op)
2051 {
2052     if (seg->s_ops->inherit == NULL)
2053         return (ENOTSUP);

```

```

2055     return (seg->s_ops->inherit(seg, addr, len, op));
2056 }
2057 #endif /* ! codereview */

```