```
**********************************************************
   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_
```

```
#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 SEGOP_DUP(s, n)             (*(s)->s_ops->dup)((s), (n))
#define SEGOP_UNMAP(s, a, l)        (*(s)->s_ops->unmap)((s), (a), (l))
#define SEGOP_FREE(s)               (*(s)->s_ops->free)((s))
#define SEGOP_FAULT(h, s, a, l, t, rw) \
                (*(s)->s_ops->fault)((h), (s), (a), (l), (t), (rw))
#define SEGOP_FAULTA(s, a)          (*(s)->s_ops->faulta)((s), (a))
#define SEGOP_SETPROT(s, a, l, p)   (*(s)->s_ops->setprot)((s), (a), (l), (p))
#define SEGOP_CHECKPROT(s, a, l, p) (*(s)->s_ops->checkprot)((s), (a), (l), (p))
#define SEGOP_KLUSTER(s, a, d)      (*(s)->s_ops->kluster)((s), (a), (d))
#define SEGOP_SYNC(s, a, l, atr, f) \
                (*(s)->s_ops->sync)((s), (a), (l), (atr), (f))
#define SEGOP_INCORE(s, a, l, v)    (*(s)->s_ops->incore)((s), (a), (l), (v))
#define SEGOP_LOCKOP(s, a, l, atr, op, b, p) \
                (*(s)->s_ops->lockop)((s), (a), (l), (atr), (op), (b), (p))
#define SEGOP_GETPROT(s, a, l, p)   (*(s)->s_ops->getprot)((s), (a), (l), (p))
#define SEGOP_GETOFFSET(s, a)       (*(s)->s_ops->getoffset)((s), (a))
#define SEGOP_GETTYPE(s, a)         (*(s)->s_ops->gettype)((s), (a))
#define SEGOP_GETVP(s, a, vpp)      (*(s)->s_ops->getvp)((s), (a), (vpp))
#define SEGOP_ADVISE(s, a, l, b)    (*(s)->s_ops->advise)((s), (a), (l), (b))
#define SEGOP_DUMP(s)               (*(s)->s_ops->dump)((s))
#define SEGOP_PAGELOCK(s, a, l, p, t, rw) \
                (*(s)->s_ops->pagelock)((s), (a), (l), (p), (t), (rw))
#define SEGOP_SETPAGESIZE(s, a, l, szc) \
                (*(s)->s_ops->setpagesize)((s), (a), (l), (szc))
#define SEGOP_GETMEMID(s, a, mp)    (*(s)->s_ops->getmemid)((s), (a), (mp))
#define SEGOP_GETPOLICY(s, a)       (*(s)->s_ops->getpolicy)((s), (a))
#define SEGOP_CAPABLE(s, c)         (*(s)->s_ops->capable)((s), (c))
#define SEGOP_INHERIT(s, a, l, b)   (*(s)->s_ops->inherit)((s), (a), (l), (b))

#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 /* ! codereview */

#endif  /* _KERNEL */

#ifdef  __cplusplus
}
#endif

#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_

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

int
segop_getvp(struct seg *seg, caddr_t addr, struct vnode **vpp)
{
        VERIFY3P(seg->s_ops->getvp, !=, NULL);
        return (seg->s_ops->getvp(seg, addr, vpp));
}

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

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

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

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

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

        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)
{
        VERIFY3P(seg->s_ops->setpagesize, !=, NULL);

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

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

        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)
{
        VERIFY3P(seg->s_ops->capable, !=, NULL);

        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));
}
#endif /* ! codereview */
```