```
**********************************************************
   55638 Fri May  8 18:04:54 2015
new/usr/src/uts/common/exec/elf/elf.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
______unchanged_portion_omitted_

1721 int
1722 elfcore(vnode_t *vp, proc_t *p, cred_t *credp, rlim64_t rlimit, int sig,
1723     core_content_t content)
1724 {
1725         offset_t poffset, soffset;
1726         Off doffset;
1727         int error, i, nphdrs, nshdrs;
1728         int overflow = 0;
1729         struct seg *seg;
1730         struct as *as = p->p_as;
1731         union {
1732                 Ehdr ehdr;
1733                 Phdr phdr[1];
1734                 Shdr shdr[1];
1735         } *bigwad;
1736         size_t bigsize;
1737         size_t phdrsz, shdrsz;
1738         Ehdr *ehdr;
1739         Phdr *v;
1740         caddr_t brkbase;
1741         size_t brksize;
1742         caddr_t stkbase;
1743         size_t stksize;
1744         int ntries = 0;
1745         klwp_t *lwp = ttolwp(curthread);

1747 top:
1748         /*
1749          * Make sure we have everything we need (registers, etc.).
1750          * All other lwps have already stopped and are in an orderly state.
1751          */
1752         ASSERT(p == ttoproc(curthread));
1753         prstop(0, 0);

1755         AS_LOCK_ENTER(as, &as->a_lock, RW_WRITER);
1756         nphdrs = prnsegs(as, 0) + 2;            /* two CORE note sections */

1758         /*
1759          * Count the number of section headers we're going to need.
1760          */
1761         nshdrs = 0;
1762         if (content & (CC_CONTENT_CTF | CC_CONTENT_SYMTAB)) {
1763                 (void) process_scns(content, p, credp, NULL, NULL, NULL, 0,
1764                     NULL, &nshdrs);
1765         }
1766         AS_LOCK_EXIT(as, &as->a_lock);

1768         ASSERT(nshdrs == 0 || nshdrs > 1);

1770         /*
1771          * The core file contents may required zero section headers, but if
1772          * we overflow the 16 bits allotted to the program header count in
1773          * the ELF header, we'll need that program header at index zero.
1774          */
1775         if (nshdrs == 0 && nphdrs >= PN_XNUM)
1776                 nshdrs = 1;

1778         phdrsz = nphdrs * sizeof (Phdr);
```



```
1779         shdrsz = nshdrs * sizeof (Shdr);

1781         bigsize = MAX(sizeof (*bigwad), MAX(phdrsz, shdrsz));
1782         bigwad = kmem_alloc(bigsize, KM_SLEEP);

1784         ehdr = &bigwad->ehdr;
1785         bzero(ehdr, sizeof (*ehdr));

1787         ehdr->e_ident[EI_MAG0] = ELFMAG0;
1788         ehdr->e_ident[EI_MAG1] = ELFMAG1;
1789         ehdr->e_ident[EI_MAG2] = ELFMAG2;
1790         ehdr->e_ident[EI_MAG3] = ELFMAG3;
1791         ehdr->e_ident[EI_CLASS] = ELFCLASS;
1792         ehdr->e_type = ET_CORE;

1794 #if !defined(_LP64) || defined(_ELF32_COMPAT)

1796 #if defined(__sparc)
1797         ehdr->e_ident[EI_DATA] = ELFDATA2MSB;
1798         ehdr->e_machine = EM_SPARC;
1799 #elif defined(__i386) || defined(__i386_COMPAT)
1800         ehdr->e_ident[EI_DATA] = ELFDATA2LSB;
1801         ehdr->e_machine = EM_386;
1802 #else
1803 #error "no recognized machine type is defined"
1804 #endif

1806 #else   /* !defined(_LP64) || defined(_ELF32_COMPAT) */

1808 #if defined(__sparc)
1809         ehdr->e_ident[EI_DATA] = ELFDATA2MSB;
1810         ehdr->e_machine = EM_SPARCV9;
1811 #elif defined(__amd64)
1812         ehdr->e_ident[EI_DATA] = ELFDATA2LSB;
1813         ehdr->e_machine = EM_AMD64;
1814 #else
1815 #error "no recognized 64-bit machine type is defined"
1816 #endif

1818 #endif  /* !defined(_LP64) || defined(_ELF32_COMPAT) */

1820         /*
1821          * If the count of program headers or section headers or the index
1822          * of the section string table can't fit in the mere 16 bits
1823          * shortsightedly allotted to them in the ELF header, we use the
1824          * extended formats and put the real values in the section header
1825          * as index 0.
1826          */
1827         ehdr->e_version = EV_CURRENT;
1828         ehdr->e_ehsize = sizeof (Ehdr);

1830         if (nphdrs >= PN_XNUM)
1831                 ehdr->e_phnum = PN_XNUM;
1832         else
1833                 ehdr->e_phnum = (unsigned short)nphdrs;

1835         ehdr->e_phoff = sizeof (Ehdr);
1836         ehdr->e_phentsize = sizeof (Phdr);

1838         if (nshdrs > 0) {
1839                 if (nshdrs >= SHN_LORESERVE)
1840                         ehdr->e_shnum = 0;
1841                 else
1842                         ehdr->e_shnum = (unsigned short)nshdrs;

1844                 if (nshdrs - 1 >= SHN_LORESERVE)
```

```
1845                        ehdr->e_shstrndx = SHN_XINDEX;
1846                else
1847                        ehdr->e_shstrndx = (unsigned short)(nshdrs - 1);

1849                ehdr->e_shoff = ehdr->e_phoff + ehdr->e_phentsize * nphdrs;
1850                ehdr->e_shentsize = sizeof (Shdr);
1851        }

1853        if (error = core_write(vp, UIO_SYSSPACE, (offset_t)0, ehdr,
1854            sizeof (Ehdr), rlimit, credp))
1855                goto done;

1857        poffset = sizeof (Ehdr);
1858        soffset = sizeof (Ehdr) + phdrsz;
1859        doffset = sizeof (Ehdr) + phdrsz + shdrsz;

1861        v = &bigwad->phdr[0];
1862        bzero(v, phdrsz);

1864        setup_old_note_header(&v[0], p);
1865        v[0].p_offset = doffset = roundup(doffset, sizeof (Word));
1866        doffset += v[0].p_filesz;

1868        setup_note_header(&v[1], p);
1869        v[1].p_offset = doffset = roundup(doffset, sizeof (Word));
1870        doffset += v[1].p_filesz;

1872        mutex_enter(&p->p_lock);

1874        brkbase = p->p_brkbase;
1875        brksize = p->p_brksize;

1877        stkbase = p->p_usrstack - p->p_stksize;
1878        stksize = p->p_stksize;

1880        mutex_exit(&p->p_lock);

1882        AS_LOCK_ENTER(as, &as->a_lock, RW_WRITER);
1883        i = 2;
1884        for (seg = AS_SEGFIRST(as); seg != NULL; seg = AS_SEGNEXT(as, seg)) {
1885                caddr_t eaddr = seg->s_base + pr_getsegsize(seg, 0);
1886                caddr_t saddr, naddr;
1887                void *tmp = NULL;
1888                extern const struct seg_ops segspt_shmops;
1888                extern struct seg_ops segspt_shmops;

1890                for (saddr = seg->s_base; saddr < eaddr; saddr = naddr) {
1891                        uint_t prot;
1892                        size_t size;
1893                        int type;
1894                        vnode_t *mvp;

1896                        prot = pr_getprot(seg, 0, &tmp, &saddr, &naddr, eaddr);
1897                        prot &= PROT_READ | PROT_WRITE | PROT_EXEC;
1898                        if ((size = (size_t)(naddr - saddr)) == 0)
1899                                continue;
1900                        if (i == nphdrs) {
1901                                overflow++;
1902                                continue;
1903                        }
1904                        v[i].p_type = PT_LOAD;
1905                        v[i].p_vaddr = (Addr)(uintptr_t)saddr;
1906                        v[i].p_memsz = size;
1907                        if (prot & PROT_READ)
1908                                v[i].p_flags |= PF_R;
1909                        if (prot & PROT_WRITE)
```



```
1910                                v[i].p_flags |= PF_W;
1911                        if (prot & PROT_EXEC)
1912                                v[i].p_flags |= PF_X;

1914                        /*
1915                         * Figure out which mappings to include in the core.
1916                         */
1917                        type = segop_gettype(seg, saddr);

1919                        if (saddr == stkbase && size == stksize) {
1920                                if (!(content & CC_CONTENT_STACK))
1921                                        goto exclude;

1923                        } else if (saddr == brkbase && size == brksize) {
1924                                if (!(content & CC_CONTENT_HEAP))
1925                                        goto exclude;

1927                        } else if (seg->s_ops == &segspt_shmops) {
1928                                if (type & MAP_NORESERVE) {
1929                                        if (!(content & CC_CONTENT_DISM))
1930                                                goto exclude;
1931                                } else {
1932                                        if (!(content & CC_CONTENT_ISM))
1933                                                goto exclude;
1934                                }

1936                        } else if (seg->s_ops != &segvn_ops) {
1937                                goto exclude;

1939                        } else if (type & MAP_SHARED) {
1940                                if (shmgetid(p, saddr) != SHMID_NONE) {
1941                                        if (!(content & CC_CONTENT_SHM))
1942                                                goto exclude;

1944                                } else if (segop_getvp(seg, seg->s_base,
1945                                    &mvp) != 0 || mvp == NULL ||
1946                                    mvp->v_type != VREG) {
1947                                        if (!(content & CC_CONTENT_SHANON))
1948                                                goto exclude;

1950                                } else {
1951                                        if (!(content & CC_CONTENT_SHFILE))
1952                                                goto exclude;
1953                                }

1955                        } else if (segop_getvp(seg, seg->s_base, &mvp) != 0 ||
1956                            mvp == NULL || mvp->v_type != VREG) {
1957                                if (!(content & CC_CONTENT_ANON))
1958                                        goto exclude;

1960                        } else if (prot == (PROT_READ | PROT_EXEC)) {
1961                                if (!(content & CC_CONTENT_TEXT))
1962                                        goto exclude;

1964                        } else if (prot == PROT_READ) {
1965                                if (!(content & CC_CONTENT_RODATA))
1966                                        goto exclude;

1968                        } else {
1969                                if (!(content & CC_CONTENT_DATA))
1970                                        goto exclude;
1971                        }

1973                        doffset = roundup(doffset, sizeof (Word));
1974                        v[i].p_offset = doffset;
1975                        v[i].p_filesz = size;
```

```
1976                        doffset += size;
1977 exclude:
1978                        i++;
1979                }
1980                ASSERT(tmp == NULL);
1981        }
1982        AS_LOCK_EXIT(as, &as->a_lock);

1984        if (overflow || i != nphdrs) {
1985                if (ntries++ == 0) {
1986                        kmem_free(bigwad, bigsize);
1987                        overflow = 0;
1988                        goto top;
1989                }
1990                cmn_err(CE_WARN, "elfcore: core dump failed for "
1991                    "process %d; address space is changing", p->p_pid);
1992                error = EIO;
1993                goto done;
1994        }

1996        if ((error = core_write(vp, UIO_SYSSPACE, poffset,
1997            v, phdrsz, rlimit, credp)) != 0)
1998                goto done;

2000        if ((error = write_old_elfnotes(p, sig, vp, v[0].p_offset, rlimit,
2001            credp)) != 0)
2002                goto done;

2004        if ((error = write_elfnotes(p, sig, vp, v[1].p_offset, rlimit,
2005            credp, content)) != 0)
2006                goto done;

2008        for (i = 2; i < nphdrs; i++) {
2009                prkillinfo_t killinfo;
2010                sigqueue_t *sq;
2011                int sig, j;

2013                if (v[i].p_filesz == 0)
2014                        continue;

2016                /*
2017                 * If dumping out this segment fails, rather than failing
2018                 * the core dump entirely, we reset the size of the mapping
2019                 * to zero to indicate that the data is absent from the core
2020                 * file and or in the PF_SUNW_FAILURE flag to differentiate
2021                 * this from mappings that were excluded due to the core file
2022                 * content settings.
2023                 */
2024                if ((error = core_seg(p, vp, v[i].p_offset,
2025                    (caddr_t)(uintptr_t)v[i].p_vaddr, v[i].p_filesz,
2026                    rlimit, credp)) == 0) {
2027                        continue;
2028                }

2030                if ((sig = lwp->lwp_cursig) == 0) {
2031                        /*
2032                         * We failed due to something other than a signal.
2033                         * Since the space reserved for the segment is now
2034                         * unused, we stash the errno in the first four
2035                         * bytes. This undocumented interface will let us
2036                         * understand the nature of the failure.
2037                         */
2038                        (void) core_write(vp, UIO_SYSSPACE, v[i].p_offset,
2039                            &error, sizeof (error), rlimit, credp);

2041                        v[i].p_filesz = 0;
```



```
2042                        v[i].p_flags |= PF_SUNW_FAILURE;
2043                        if ((error = core_write(vp, UIO_SYSSPACE,
2044                            poffset + sizeof (v[i]) * i, &v[i], sizeof (v[i]),
2045                            rlimit, credp)) != 0)
2046                                goto done;

2048                        continue;
2049                }

2051                /*
2052                 * We took a signal.  We want to abort the dump entirely, but
2053                 * we also want to indicate what failed and why.  We therefore
2054                 * use the space reserved for the first failing segment to
2055                 * write our error (which, for purposes of compatability with
2056                 * older core dump readers, we set to EINTR) followed by any
2057                 * siginfo associated with the signal.
2058                 */
2059                bzero(&killinfo, sizeof (killinfo));
2060                killinfo.prk_error = EINTR;

2062                sq = sig == SIGKILL ? curproc->p_killsqp : lwp->lwp_curinfo;

2064                if (sq != NULL) {
2065                        bcopy(&sq->sq_info, &killinfo.prk_info,
2066                            sizeof (sq->sq_info));
2067                } else {
2068                        killinfo.prk_info.si_signo = lwp->lwp_cursig;
2069                        killinfo.prk_info.si_code = SI_NOINFO;
2070                }

2072 #if (defined(_SYSCALL32_IMPL) || defined(_LP64))
2073                /*
2074                 * If this is a 32-bit process, we need to translate from the
2075                 * native siginfo to the 32-bit variant.  (Core readers must
2076                 * always have the same data model as their target or must
2077                 * be aware of -- and compensate for -- data model differences.)
2078                 */
2079                if (curproc->p_model == DATAMODEL_ILP32) {
2080                        siginfo32_t si32;

2082                        siginfo_kto32((k_siginfo_t *)&killinfo.prk_info, &si32);
2083                        bcopy(&si32, &killinfo.prk_info, sizeof (si32));
2084                }
2085 #endif

2087                (void) core_write(vp, UIO_SYSSPACE, v[i].p_offset,
2088                    &killinfo, sizeof (killinfo), rlimit, credp);

2090                /*
2091                 * For the segment on which we took the signal, indicate that
2092                 * its data now refers to a siginfo.
2093                 */
2094                v[i].p_filesz = 0;
2095                v[i].p_flags |= PF_SUNW_FAILURE | PF_SUNW_KILLED |
2096                    PF_SUNW_SIGINFO;

2098                /*
2099                 * And for every other segment, indicate that its absence
2100                 * is due to a signal.
2101                 */
2102                for (j = i + 1; j < nphdrs; j++) {
2103                        v[j].p_filesz = 0;
2104                        v[j].p_flags |= PF_SUNW_FAILURE | PF_SUNW_KILLED;
2105                }

2107                /*
```

```
2108                  * Finally, write out our modified program headers.
2109                  */
2110                 if ((error = core_write(vp, UIO_SYSSPACE,
2111                     poffset + sizeof (v[i]) * i, &v[i],
2112                     sizeof (v[i]) * (nphdrs - i), rlimit, credp)) != 0)
2113                         goto done;

2115                 break;
2116         }

2118         if (nshdrs > 0) {
2119                 bzero(&bigwad->shdr[0], shdrsz);

2121                 if (nshdrs >= SHN_LORESERVE)
2122                         bigwad->shdr[0].sh_size = nshdrs;

2124                 if (nshdrs - 1 >= SHN_LORESERVE)
2125                         bigwad->shdr[0].sh_link = nshdrs - 1;

2127                 if (nphdrs >= PN_XNUM)
2128                         bigwad->shdr[0].sh_info = nphdrs;

2130                 if (nshdrs > 1) {
2131                         AS_LOCK_ENTER(as, &as->a_lock, RW_WRITER);
2132                         if ((error = process_scns(content, p, credp, vp,
2133                             &bigwad->shdr[0], nshdrs, rlimit, &doffset,
2134                             NULL)) != 0) {
2135                                 AS_LOCK_EXIT(as, &as->a_lock);
2136                                 goto done;
2137                         }
2138                         AS_LOCK_EXIT(as, &as->a_lock);
2139                 }

2141                 if ((error = core_write(vp, UIO_SYSSPACE, soffset,
2142                     &bigwad->shdr[0], shdrsz, rlimit, credp)) != 0)
2143                         goto done;
2144         }

2146 done:
2147         kmem_free(bigwad, bigsize);
2148         return (error);
2149 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  112647 Fri May  8 18:04:54 2015
new/usr/src/uts/common/fs/proc/prsubr.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
______unchanged_portion_omitted_

  98 size_t pagev_lim = 256 * 1024;  /* limit on number of pages in prpagev_t */

 100 extern const struct seg_ops segdev_ops; /* needs a header file */
 101 extern const struct seg_ops segspt_shmops;      /* needs a header file */
 100 extern struct seg_ops segdev_ops;       /* needs a header file */
 101 extern struct seg_ops segspt_shmops;    /* needs a header file */

 103 static  int     set_watched_page(proc_t *, caddr_t, caddr_t, ulong_t, ulong_t);
 104 static  void    clear_watched_page(proc_t *, caddr_t, caddr_t, ulong_t);

 106 /*
 107  * Choose an lwp from the complete set of lwps for the process.
 108  * This is called for any operation applied to the process
 109  * file descriptor that requires an lwp to operate upon.
 110  *
 111  * Returns a pointer to the thread for the selected LWP,
 112  * and with the dispatcher lock held for the thread.
 113  *
 114  * The algorithm for choosing an lwp is critical for /proc semantics;
 115  * don't touch this code unless you know all of the implications.
 116  */
 117 kthread_t *
 118 prchoose(proc_t *p)
 119 {
 120         kthread_t *t;
 121         kthread_t *t_onproc = NULL;     /* running on processor */
 122         kthread_t *t_run = NULL;        /* runnable, on disp queue */
 123         kthread_t *t_sleep = NULL;      /* sleeping */
 124         kthread_t *t_hold = NULL;       /* sleeping, performing hold */
 125         kthread_t *t_susp = NULL;       /* suspended stop */
 126         kthread_t *t_jstop = NULL;      /* jobcontrol stop, w/o directed stop */
 127         kthread_t *t_jdstop = NULL;     /* jobcontrol stop with directed stop */
 128         kthread_t *t_req = NULL;        /* requested stop */
 129         kthread_t *t_istop = NULL;      /* event-of-interest stop */
 130         kthread_t *t_dtrace = NULL;     /* DTrace stop */

 132         ASSERT(MUTEX_HELD(&p->p_lock));

 134         /*
 135          * If the agent lwp exists, it takes precedence over all others.
 136          */
 137         if ((t = p->p_agenttp) != NULL) {
 138                 thread_lock(t);
 139                 return (t);
 140         }

 142         if ((t = p->p_tlist) == NULL)   /* start at the head of the list */
 143                 return (t);
 144         do {            /* for eacn lwp in the process */
 145                 if (VSTOPPED(t)) {      /* virtually stopped */
 146                         if (t_req == NULL)
 147                                 t_req = t;
 148                         continue;
 149                 }

 151                 thread_lock(t);         /* make sure thread is in good state */
 152                 switch (t->t_state) {
 153                 default:
```



```
 154                         panic("prchoose: bad thread state %d, thread 0x%p",
 155                             t->t_state, (void *)t);
 156                         /*NOTREACHED*/
 157                 case TS_SLEEP:
 158                         /* this is filthy */
 159                         if (t->t_wchan == (caddr_t)&p->p_holdlwps &&
 160                             t->t_wchan0 == NULL) {
 161                                 if (t_hold == NULL)
 162                                         t_hold = t;
 163                         } else {
 164                                 if (t_sleep == NULL)
 165                                         t_sleep = t;
 166                         }
 167                         break;
 168                 case TS_RUN:
 169                 case TS_WAIT:
 170                         if (t_run == NULL)
 171                                 t_run = t;
 172                         break;
 173                 case TS_ONPROC:
 174                         if (t_onproc == NULL)
 175                                 t_onproc = t;
 176                         break;
 177                 case TS_ZOMB:           /* last possible choice */
 178                         break;
 179                 case TS_STOPPED:
 180                         switch (t->t_whystop) {
 181                         case PR_SUSPENDED:
 182                                 if (t_susp == NULL)
 183                                         t_susp = t;
 184                                 break;
 185                         case PR_JOBCONTROL:
 186                                 if (t->t_proc_flag & TP_PRSTOP) {
 187                                         if (t_jdstop == NULL)
 188                                                 t_jdstop = t;
 189                                 } else {
 190                                         if (t_jstop == NULL)
 191                                                 t_jstop = t;
 192                                 }
 193                                 break;
 194                         case PR_REQUESTED:
 195                                 if (t->t_dtrace_stop && t_dtrace == NULL)
 196                                         t_dtrace = t;
 197                                 else if (t_req == NULL)
 198                                         t_req = t;
 199                                 break;
 200                         case PR_SYSENTRY:
 201                         case PR_SYSEXIT:
 202                         case PR_SIGNALLED:
 203                         case PR_FAULTED:
 204                                 /*
 205                                  * Make an lwp calling exit() be the
 206                                  * last lwp seen in the process.
 207                                  */
 208                                 if (t_istop == NULL ||
 209                                     (t_istop->t_whystop == PR_SYSENTRY &&
 210                                     t_istop->t_whatstop == SYS_exit))
 211                                         t_istop = t;
 212                                 break;
 213                         case PR_CHECKPOINT:     /* can't happen? */
 214                                 break;
 215                         default:
 216                                 panic("prchoose: bad t_whystop %d, thread 0x%p",
 217                                     t->t_whystop, (void *)t);
 218                                 /*NOTREACHED*/
 219                         }
```

```
220                         break;
221                 }
222                 thread_unlock(t);
223         } while ((t = t->t_forw) != p->p_tlist);

225         if (t_onproc)
226                 t = t_onproc;
227         else if (t_run)
228                 t = t_run;
229         else if (t_sleep)
230                 t = t_sleep;
231         else if (t_jstop)
232                 t = t_jstop;
233         else if (t_jdstop)
234                 t = t_jdstop;
235         else if (t_istop)
236                 t = t_istop;
237         else if (t_dtrace)
238                 t = t_dtrace;
239         else if (t_req)
240                 t = t_req;
241         else if (t_hold)
242                 t = t_hold;
243         else if (t_susp)
244                 t = t_susp;
245         else                    /* TS_ZOMB */
246                 t = p->p_tlist;

248         if (t != NULL)
249                 thread_lock(t);
250         return (t);
251 }
```

unchanged_portion_omitted_

```
**********************************************************
  248852 Fri May  8 18:04:54 2015
new/usr/src/uts/common/os/sunddi.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
______unchanged_portion_omitted_

8219 /*
8220  * A consolidation private function which is essentially equivalent to
8221  * ddi_umem_lock but with the addition of arguments ops_vector and procp.
8222  * A call to as_add_callback is done if DDI_UMEMLOCK_LONGTERM is set, and
8223  * the ops_vector is valid.
8224  *
8225  * Lock the virtual address range in the current process and create a
8226  * ddi_umem_cookie (of type UMEM_LOCKED). This can be used to pass to
8227  * ddi_umem_iosetup to create a buf or do devmap_umem_setup/remap to export
8228  * to user space.
8229  *
8230  * Note: The resource control accounting currently uses a full charge model
8231  * in other words attempts to lock the same/overlapping areas of memory
8232  * will deduct the full size of the buffer from the projects running
8233  * counter for the device locked memory.
8234  *
8235  * addr, size should be PAGESIZE aligned
8236  *
8237  * flags - DDI_UMEMLOCK_READ, DDI_UMEMLOCK_WRITE or both
8238  *      identifies whether the locked memory will be read or written or both
8239  *      DDI_UMEMLOCK_LONGTERM  must be set when the locking will
8240  * be maintained for an indefinitely long period (essentially permanent),
8241  * rather than for what would be required for a typical I/O completion.
8242  * When DDI_UMEMLOCK_LONGTERM is set, umem_lockmemory will return EFAULT
8243  * if the memory pertains to a regular file which is mapped MAP_SHARED.
8244  * This is to prevent a deadlock if a file truncation is attempted after
8245  * after the locking is done.
8246  *
8247  * Returns 0 on success
8248  *      EINVAL - for invalid parameters
8249  *      EPERM, ENOMEM and other error codes returned by as_pagelock
8250  *      ENOMEM - is returned if the current request to lock memory exceeds
8251  *              *.max-locked-memory resource control value.
8252  *      EFAULT - memory pertains to a regular file mapped shared and
8253  *              and DDI_UMEMLOCK_LONGTERM flag is set
8254  *      EAGAIN - could not start the ddi_umem_unlock list processing thread
8255  */
8256 int
8257 umem_lockmemory(caddr_t addr, size_t len, int flags, ddi_umem_cookie_t *cookie,
8258                 struct umem_callback_ops *ops_vector,
8259                 proc_t *procp)
8260 {
8261         int     error;
8262         struct ddi_umem_cookie *p;
8263         void    (*driver_callback)() = NULL;
8264         struct as *as;
8265         struct seg              *seg;
8266         vnode_t                 *vp;

8268         /* Allow device drivers to not have to reference "curproc" */
8269         if (procp == NULL)
8270                 procp = curproc;
8271         as = procp->p_as;
8272         *cookie = NULL;         /* in case of any error return */

8274         /* These are the only three valid flags */
8275         if ((flags & ~(DDI_UMEMLOCK_READ | DDI_UMEMLOCK_WRITE |
8276             DDI_UMEMLOCK_LONGTERM)) != 0)
```



```
8277                 return (EINVAL);

8279         /* At least one (can be both) of the two access flags must be set */
8280         if ((flags & (DDI_UMEMLOCK_READ | DDI_UMEMLOCK_WRITE)) == 0)
8281                 return (EINVAL);

8283         /* addr and len must be page-aligned */
8284         if (((uintptr_t)addr & PAGEOFFSET) != 0)
8285                 return (EINVAL);

8287         if ((len & PAGEOFFSET) != 0)
8288                 return (EINVAL);

8290         /*
8291          * For longterm locking a driver callback must be specified; if
8292          * not longterm then a callback is optional.
8293          */
8294         if (ops_vector != NULL) {
8295                 if (ops_vector->cbo_umem_callback_version !=
8296                     UMEM_CALLBACK_VERSION)
8297                         return (EINVAL);
8298                 else
8299                         driver_callback = ops_vector->cbo_umem_lock_cleanup;
8300         }
8301         if ((driver_callback == NULL) && (flags & DDI_UMEMLOCK_LONGTERM))
8302                 return (EINVAL);

8304         /*
8305          * Call i_ddi_umem_unlock_thread_start if necessary.  It will
8306          * be called on first ddi_umem_lock or umem_lockmemory call.
8307          */
8308         if (ddi_umem_unlock_thread == NULL)
8309                 i_ddi_umem_unlock_thread_start();

8311         /* Allocate memory for the cookie */
8312         p = kmem_zalloc(sizeof (struct ddi_umem_cookie), KM_SLEEP);

8314         /* Convert the flags to seg_rw type */
8315         if (flags & DDI_UMEMLOCK_WRITE) {
8316                 p->s_flags = S_WRITE;
8317         } else {
8318                 p->s_flags = S_READ;
8319         }

8321         /* Store procp in cookie for later iosetup/unlock */
8322         p->procp = (void *)procp;

8324         /*
8325          * Store the struct as pointer in cookie for later use by
8326          * ddi_umem_unlock.  The proc->p_as will be stale if ddi_umem_unlock
8327          * is called after relvm is called.
8328          */
8329         p->asp = as;

8331         /*
8332          * The size field is needed for lockmem accounting.
8333          */
8334         p->size = len;
8335         init_lockedmem_rctl_flag(p);

8337         if (umem_incr_devlockmem(p) != 0) {
8338                 /*
8339                  * The requested memory cannot be locked
8340                  */
8341                 kmem_free(p, sizeof (struct ddi_umem_cookie));
8342                 *cookie = (ddi_umem_cookie_t)NULL;
```

```
8343                return (ENOMEM);
8344        }

8346        /* Lock the pages corresponding to addr, len in memory */
8347        error = as_pagelock(as, &(p->pparray), addr, len, p->s_flags);
8348        if (error != 0) {
8349                umem_decr_devlockmem(p);
8350                kmem_free(p, sizeof (struct ddi_umem_cookie));
8351                *cookie = (ddi_umem_cookie_t)NULL;
8352                return (error);
8353        }

8355        /*
8356         * For longterm locking the addr must pertain to a seg_vn segment or
8357         * or a seg_spt segment.
8358         * If the segment pertains to a regular file, it cannot be
8359         * mapped MAP_SHARED.
8360         * This is to prevent a deadlock if a file truncation is attempted
8361         * after the locking is done.
8362         * Doing this after as_pagelock guarantees persistence of the as; if
8363         * an unacceptable segment is found, the cleanup includes calling
8364         * as_pageunlock before returning EFAULT.
8365         *
8366         * segdev is allowed here as it is already locked.  This allows
8367         * for memory exported by drivers through mmap() (which is already
8368         * locked) to be allowed for LONGTERM.
8369         */
8370        if (flags & DDI_UMEMLOCK_LONGTERM) {
8371                extern const struct seg_ops segspt_shmops;
8372                extern const struct seg_ops segdev_ops;
8371                extern  struct seg_ops segspt_shmops;
8372                extern  struct seg_ops segdev_ops;
8373                AS_LOCK_ENTER(as, &as->a_lock, RW_READER);
8374                for (seg = as_segat(as, addr); ; seg = AS_SEGNEXT(as, seg)) {
8375                        if (seg == NULL || seg->s_base > addr + len)
8376                                break;
8377                        if (seg->s_ops == &segdev_ops)
8378                                continue;
8379                        if (((seg->s_ops != &segvn_ops) &&
8380                            (seg->s_ops != &segspt_shmops)) ||
8381                            ((segop_getvp(seg, addr, &vp) == 0 &&
8382                            vp != NULL && vp->v_type == VREG) &&
8383                            (segop_gettype(seg, addr) & MAP_SHARED))) {
8384                                as_pageunlock(as, p->pparray,
8385                                    addr, len, p->s_flags);
8386                                AS_LOCK_EXIT(as, &as->a_lock);
8387                                umem_decr_devlockmem(p);
8388                                kmem_free(p, sizeof (struct ddi_umem_cookie));
8389                                *cookie = (ddi_umem_cookie_t)NULL;
8390                                return (EFAULT);
8391                        }
8392                }
8393                AS_LOCK_EXIT(as, &as->a_lock);
8394        }


8397        /* Initialize the fields in the ddi_umem_cookie */
8398        p->cvaddr = addr;
8399        p->type = UMEM_LOCKED;
8400        if (driver_callback != NULL) {
8401                /* i_ddi_umem_unlock and umem_lock_undo may need the cookie */
8402                p->cook_refcnt = 2;
8403                p->callbacks = *ops_vector;
8404        } else {
8405                /* only i_ddi_umme_unlock needs the cookie */
8406                p->cook_refcnt = 1;
```



```
8407        }

8409        *cookie = (ddi_umem_cookie_t)p;

8411        /*
8412         * If a driver callback was specified, add an entry to the
8413         * as struct callback list. The as_pagelock above guarantees
8414         * the persistence of as.
8415         */
8416        if (driver_callback) {
8417                error = as_add_callback(as, umem_lock_undo, p, AS_ALL_EVENT,
8418                    addr, len, KM_SLEEP);
8419                if (error != 0) {
8420                        as_pageunlock(as, p->pparray,
8421                            addr, len, p->s_flags);
8422                        umem_decr_devlockmem(p);
8423                        kmem_free(p, sizeof (struct ddi_umem_cookie));
8424                        *cookie = (ddi_umem_cookie_t)NULL;
8425                }
8426        }
8427        return (error);
8428 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    8581 Fri May  8 18:04:55 2015
new/usr/src/uts/common/os/urw.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
/*
 * 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 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/atomic.h>
#include <sys/errno.h>
#include <sys/stat.h>
#include <sys/modctl.h>
#include <sys/conf.h>
#include <sys/systm.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/cpuvar.h>
#include <sys/kmem.h>
#include <sys/strsubr.h>
#include <sys/sysmacros.h>
#include <sys/frame.h>
#include <sys/stack.h>
#include <sys/proc.h>
#include <sys/priv.h>
#include <sys/policy.h>
#include <sys/ontrap.h>
#include <sys/vmsystm.h>
#include <sys/prsystm.h>

#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_dev.h>
#include <vm/seg_vn.h>
#include <vm/seg_spt.h>
#include <vm/seg_kmem.h>

extern const struct seg_ops segdev_ops; /* needs a header file */
extern const struct seg_ops segspt_shmops;      /* needs a header file */
```



```
extern struct seg_ops segdev_ops;       /* needs a header file */
extern struct seg_ops segspt_shmops;    /* needs a header file */

static int
page_valid(struct seg *seg, caddr_t addr)
{
        struct segvn_data *svd;
        vnode_t *vp;
        vattr_t vattr;

        /*
         * Fail if the page doesn't map to a page in the underlying
         * mapped file, if an underlying mapped file exists.
         */
        vattr.va_mask = AT_SIZE;
        if (seg->s_ops == &segvn_ops &&
            segop_getvp(seg, addr, &vp) == 0 &&
            vp != NULL && vp->v_type == VREG &&
            VOP_GETATTR(vp, &vattr, 0, CRED(), NULL) == 0) {
                u_offset_t size = roundup(vattr.va_size, (u_offset_t)PAGESIZE);
                u_offset_t offset = segop_getoffset(seg, addr);

                if (offset >= size)
                        return (0);
        }

        /*
         * Fail if this is an ISM shared segment and the address is
         * not within the real size of the spt segment that backs it.
         */
        if (seg->s_ops == &segspt_shmops &&
            addr >= seg->s_base + spt_realsize(seg))
                return (0);

        /*
         * Fail if the segment is mapped from /dev/null.
         * The key is that the mapping comes from segdev and the
         * type is neither MAP_SHARED nor MAP_PRIVATE.
         */
        if (seg->s_ops == &segdev_ops &&
            ((segop_gettype(seg, addr) & (MAP_SHARED | MAP_PRIVATE)) == 0))
                return (0);

        /*
         * Fail if the page is a MAP_NORESERVE page that has
         * not actually materialized.
         * We cheat by knowing that segvn is the only segment
         * driver that supports MAP_NORESERVE.
         */
        if (seg->s_ops == &segvn_ops &&
            (svd = (struct segvn_data *)seg->s_data) != NULL &&
            (svd->vp == NULL || svd->vp->v_type != VREG) &&
            (svd->flags & MAP_NORESERVE)) {
                /*
                 * Guilty knowledge here.  We know that
                 * segvn_incore returns more than just the
                 * low-order bit that indicates the page is
                 * actually in memory.  If any bits are set,
                 * then there is backing store for the page.
                 */
                char incore = 0;
                (void) segop_incore(seg, addr, PAGESIZE, &incore);
                if (incore == 0)
                        return (0);
        }
        return (1);
```

```
 125 }
______unchanged_portion_omitted_
```

****
9948 Fri May 8 18:04:55 2015
new/usr/src/uts/common/vm/seg.h
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
****
_____***unchanged_portion_omitted_***

```
 102 typedef struct seg {
 103         caddr_t s_base;                 /* base virtual address */
 104         size_t  s_size;                 /* size in bytes */
 105         uint_t  s_szc;                  /* max page size code */
 106         uint_t  s_flags;                /* flags for segment, see below */
 107         struct  as *s_as;               /* containing address space */
 108         avl_node_t s_tree;              /* AVL tree links to segs in this as */
 109         const struct seg_ops *s_ops;    /* ops vector: see below */
 109         struct  seg_ops *s_ops;         /* ops vector: see below */
 110         void *s_data;                   /* private data for instance */
 111         kmutex_t s_pmtx;                /* protects seg's pcache list */
 112         pcache_link_t s_phead;          /* head of seg's pcache list */
 113 } seg_t;
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  113443 Fri May  8 18:04:55 2015
new/usr/src/uts/common/vm/seg_dev.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
   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/systm.h>
  51 #include <sys/vmsystm.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/snode.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 pgsize boundary
  99  * VA_PA_PGSIZE_ALIGNED check to see if VA is aligned with PA w.r.t. pgsize
 100  */
 101 #define VA_PA_ALIGNED(uvaddr, paddr, pgsize)            \
 102         (((uvaddr | paddr) & (pgsize - 1)) == 0)
 103 #define VA_PA_PGSIZE_ALIGNED(uvaddr, paddr, pgsize)     \
 104         (((uvaddr ^ paddr) & (pgsize - 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_getmemid(struct seg *, caddr_t, memid_t *);

 186 /*
 187  * XXX  this struct is used by rootnex_map_fault to identify
 188  *      the segment it has been passed. So if you make it
 189  *      "static" you'll need to fix rootnex_map_fault.
 190  */
 191 const struct seg_ops segdev_ops = {
 191 struct seg_ops segdev_ops = {
```



```
 192         .dup            = segdev_dup,
 193         .unmap          = segdev_unmap,
 194         .free           = segdev_free,
 195         .fault          = segdev_fault,
 196         .faulta         = segdev_faulta,
 197         .setprot        = segdev_setprot,
 198         .checkprot      = segdev_checkprot,
 199         .kluster        = (int (*)())segdev_badop,
 200         .sync           = segdev_sync,
 201         .incore         = segdev_incore,
 202         .lockop         = segdev_lockop,
 203         .getprot        = segdev_getprot,
 204         .getoffset      = segdev_getoffset,
 205         .gettype        = segdev_gettype,
 206         .getvp          = segdev_getvp,
 207         .advise         = segdev_advise,
 208         .dump           = segdev_dump,
 209         .pagelock       = segdev_pagelock,
 210         .getmemid       = segdev_getmemid,
 211 };
______unchanged_portion_omitted_
```

****
4470 Fri May 8 18:04:56 2015
new/usr/src/uts/common/vm/seg_dev.h
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
****
_____***unchanged_portion_omitted_***

```
 113 #ifdef _KERNEL

 115 /*
 116  * Mappings of /dev/null come from segdev and have no mapping type.
 117  */

 119 #define SEG_IS_DEVNULL_MAPPING(seg)     \
 120         ((seg)->s_ops == &segdev_ops && \
 121         ((segop_gettype((seg), (seg)->s_base) & (MAP_SHARED | MAP_PRIVATE)) == 0

 123 extern void segdev_init(void);

 125 extern int segdev_create(struct seg *, void *);

 127 extern int segdev_copyto(struct seg *, caddr_t, const void *, void *, size_t);
 128 extern int segdev_copyfrom(struct seg *, caddr_t, const void *, void *, size_t);
 129 extern const struct seg_ops segdev_ops;
 129 extern struct seg_ops segdev_ops;

 131 #endif  /* _KERNEL */

 133 #ifdef  __cplusplus
 134 }
```

_____***unchanged_portion_omitted_***

****
44758 Fri May 8 18:04:56 2015
new/usr/src/uts/common/vm/seg_kmem.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
****
_____***unchanged_portion_omitted_***

```
 761 static const struct seg_ops segkmem_ops = {
 761 static struct seg_ops segkmem_ops = {
 762         .fault          = segkmem_fault,
 763         .setprot        = segkmem_setprot,
 764         .checkprot      = segkmem_checkprot,
 765         .kluster        = segkmem_kluster,
 766         .dump           = segkmem_dump,
 767         .pagelock       = segkmem_pagelock,
 768         .getmemid       = segkmem_getmemid,
 769         .capable        = segkmem_capable,
 770 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   35647 Fri May  8 18:04:56 2015
new/usr/src/uts/common/vm/seg_kp.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
/*
 * 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);

/*
 * 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
```

```
 127  * in red_deep_hires and red_deep_thread respectively.
 128  */
 129 #define RED_DEEP_THRESHOLD      2000

 131 hrtime_t        red_deep_hires;
 132 kthread_t       *red_deep_thread;

 134 uint32_t        red_nmapped;
 135 uint32_t        red_closest = UINT_MAX;
 136 uint32_t        red_ndoubles;

 138 pgcnt_t anon_segkp_pages_locked;        /* See vm/anon.h */
 139 pgcnt_t anon_segkp_pages_resv;          /* anon reserved by seg_kp */

 141 static const struct seg_ops segkp_ops = {
 141 static struct   seg_ops segkp_ops = {
 142         .fault          = segkp_fault,
 143         .checkprot      = segkp_checkprot,
 144         .kluster        = segkp_kluster,
 145         .dump           = segkp_dump,
 146         .pagelock       = segkp_pagelock,
 147 };
_____unchanged_portion_omitted_
```

```
**********************************************************
    9314 Fri May  8 18:04:56 2015
new/usr/src/uts/common/vm/seg_kpm.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
/*
 * 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 const struct seg_ops segkpm_ops = {
static struct seg_ops segkpm_ops = {
        .fault          = segkpm_fault,
        .dump           = segkpm_dump,
        .pagelock       = segkpm_pagelock,
//#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,
```

```
 126         .lockop         = nop,
 127         .getprot        = nop,
 128         .getoffset      = nop,
 129         .gettype        = nop,
 130         .getvp          = nop,
 131         .advise         = nop,
 132         .getpolicy      = nop,
 133 #endif
 134 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   57271 Fri May  8 18:04:57 2015
new/usr/src/uts/common/vm/seg_map.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
/*
 * 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);

/* 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 const struct seg_ops segmap_ops = {
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,
};
______unchanged_portion_omitted_
```

```
**********************************************************
   82311 Fri May  8 18:04:57 2015
new/usr/src/uts/common/vm/seg_spt.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
   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/systm.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/vmsystm.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 const struct seg_ops segspt_ops = {
  79 struct seg_ops segspt_ops = {
  80         .unmap          = segspt_unmap,
  81         .free           = segspt_free,
  82         .getpolicy      = segspt_getpolicy,
  83 };

  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_shmgetmemid(struct seg *, caddr_t, memid_t *);
 113 static lgrp_mem_policy_info_t *segspt_shmgetpolicy(struct seg *, caddr_t);

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

```
 125         .incore         = segspt_shmincore,
 126         .lockop         = segspt_shmlockop,
 127         .getprot        = segspt_shmgetprot,
 128         .getoffset      = segspt_shmgetoffset,
 129         .gettype        = segspt_shmgettype,
 130         .getvp          = segspt_shmgetvp,
 131         .advise         = segspt_shmadvise,
 132         .dump           = segspt_shmdump,
 133         .pagelock       = segspt_shmpagelock,
 134         .getmemid       = segspt_shmgetmemid,
 135         .getpolicy      = segspt_shmgetpolicy,
 136 };
```

unchanged_portion_omitted_

```
**********************************************************
  280464 Fri May  8 18:04:57 2015
new/usr/src/uts/common/vm/seg_vn.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
/*
 * 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) 1986, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2015, Joyent, Inc. All rights reserved.
 * Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
 */

/*      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 - shared or copy-on-write from a vnode/anonymous memory.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/t_lock.h>
#include <sys/errno.h>
#include <sys/systm.h>
#include <sys/mman.h>
#include <sys/debug.h>
#include <sys/cred.h>
#include <sys/vmsystm.h>
#include <sys/tuneable.h>
#include <sys/bitmap.h>
#include <sys/swap.h>
#include <sys/kmem.h>
#include <sys/sysmacros.h>
#include <sys/vtrace.h>
#include <sys/cmn_err.h>
#include <sys/callb.h>
```



```
#include <sys/vm.h>
#include <sys/dumphdr.h>
#include <sys/lgrp.h>

#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_vn.h>
#include <vm/pvn.h>
#include <vm/anon.h>
#include <vm/page.h>
#include <vm/vpage.h>
#include <sys/proc.h>
#include <sys/task.h>
#include <sys/project.h>
#include <sys/zone.h>
#include <sys/shm_impl.h>

/*
 * segvn_fault needs a temporary page list array.  To avoid calling kmem all
 * the time, it creates a small (PVN_GETPAGE_NUM entry) array and uses it if
 * it can.  In the rare case when this page list is not large enough, it
 * goes and gets a large enough array from kmem.
 *
 * This small page list array covers either 8 pages or 64kB worth of pages -
 * whichever is smaller.
 */
#define PVN_MAX_GETPAGE_SZ      0x10000
#define PVN_MAX_GETPAGE_NUM     0x8

#if PVN_MAX_GETPAGE_SZ > PVN_MAX_GETPAGE_NUM * PAGESIZE
#define PVN_GETPAGE_SZ  ptob(PVN_MAX_GETPAGE_NUM)
#define PVN_GETPAGE_NUM PVN_MAX_GETPAGE_NUM
#else
#define PVN_GETPAGE_SZ  PVN_MAX_GETPAGE_SZ
#define PVN_GETPAGE_NUM btop(PVN_MAX_GETPAGE_SZ)
#endif

/*
 * Private seg op routines.
 */
static int      segvn_dup(struct seg *seg, struct seg *newseg);
static int      segvn_unmap(struct seg *seg, caddr_t addr, size_t len);
static void     segvn_free(struct seg *seg);
static faultcode_t segvn_fault(struct hat *hat, struct seg *seg,
                    caddr_t addr, size_t len, enum fault_type type,
                    enum seg_rw rw);
static faultcode_t segvn_faulta(struct seg *seg, caddr_t addr);
static int      segvn_setprot(struct seg *seg, caddr_t addr,
                    size_t len, uint_t prot);
static int      segvn_checkprot(struct seg *seg, caddr_t addr,
                    size_t len, uint_t prot);
static int      segvn_kluster(struct seg *seg, caddr_t addr, ssize_t delta);
static int      segvn_sync(struct seg *seg, caddr_t addr, size_t len,
                    int attr, uint_t flags);
static size_t   segvn_incore(struct seg *seg, caddr_t addr, size_t len,
                    char *vec);
static int      segvn_lockop(struct seg *seg, caddr_t addr, size_t len,
                    int attr, int op, ulong_t *lockmap, size_t pos);
static int      segvn_getprot(struct seg *seg, caddr_t addr, size_t len,
                    uint_t *protv);
static u_offset_t       segvn_getoffset(struct seg *seg, caddr_t addr);
static int      segvn_gettype(struct seg *seg, caddr_t addr);
static int      segvn_getvp(struct seg *seg, caddr_t addr,
                    struct vnode **vpp);
static int      segvn_advise(struct seg *seg, caddr_t addr, size_t len,
```

```
 127                     uint_t behav);
 128 static void     segvn_dump(struct seg *seg);
 129 static int      segvn_pagelock(struct seg *seg, caddr_t addr, size_t len,
 130                     struct page ***ppp, enum lock_type type, enum seg_rw rw);
 131 static int      segvn_setpagesize(struct seg *seg, caddr_t addr, size_t len,
 132                     uint_t szc);
 133 static int      segvn_getmemid(struct seg *seg, caddr_t addr,
 134                     memid_t *memidp);
 135 static lgrp_mem_policy_info_t   *segvn_getpolicy(struct seg *, caddr_t);
 136 static int      segvn_inherit(struct seg *, caddr_t, size_t, uint_t);

 138 const struct seg_ops segvn_ops = {
 138 struct  seg_ops segvn_ops = {
 139         .dup            = segvn_dup,
 140         .unmap          = segvn_unmap,
 141         .free           = segvn_free,
 142         .fault          = segvn_fault,
 143         .faulta         = segvn_faulta,
 144         .setprot        = segvn_setprot,
 145         .checkprot      = segvn_checkprot,
 146         .kluster        = segvn_kluster,
 147         .sync           = segvn_sync,
 148         .incore         = segvn_incore,
 149         .lockop         = segvn_lockop,
 150         .getprot        = segvn_getprot,
 151         .getoffset      = segvn_getoffset,
 152         .gettype        = segvn_gettype,
 153         .getvp          = segvn_getvp,
 154         .advise         = segvn_advise,
 155         .dump           = segvn_dump,
 156         .pagelock       = segvn_pagelock,
 157         .setpagesize    = segvn_setpagesize,
 158         .getmemid       = segvn_getmemid,
 159         .getpolicy      = segvn_getpolicy,
 160         .inherit        = segvn_inherit,
 161 };
```

*unchanged_portion_omitted_*

```
*********************************************************
    9263 Fri May  8 18:04:58 2015
new/usr/src/uts/common/vm/seg_vn.h
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
*********************************************************
______unchanged_portion_omitted_

 230 extern void     segvn_init(void);
 231 extern int      segvn_create(struct seg *, void *);

 233 extern const struct seg_ops segvn_ops;
 233 extern  struct seg_ops segvn_ops;

 235 /*
 236  * Provided as shorthand for creating user zfod segments.
 237  */
 238 extern  caddr_t zfod_argsp;
 239 extern  caddr_t kzfod_argsp;
 240 extern  caddr_t stack_exec_argsp;
 241 extern  caddr_t stack_noexec_argsp;

 243 #endif  /* _KERNEL */

 245 #ifdef  __cplusplus
 246 }
______unchanged_portion_omitted_
```

```
**********************************************************
   90474 Fri May  8 18:04:58 2015
new/usr/src/uts/common/vm/vm_as.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
______unchanged_portion_omitted_
```

```
 409 #endif /* VERIFY_SEGLIST */

 411 /*
 412  * Add a new segment to the address space. The avl_find()
 413  * may be expensive so we attempt to use last segment accessed
 414  * in as_gap() as an insertion point.
 415  */
 416 int
 417 as_addseg(struct as  *as, struct seg *newseg)
 418 {
 419         struct seg *seg;
 420         caddr_t addr;
 421         caddr_t eaddr;
 422         avl_index_t where;

 424         ASSERT(AS_WRITE_HELD(as, &as->a_lock));

 426         as->a_updatedir = 1;    /* inform /proc */
 427         gethrestime(&as->a_updatetime);

 429         if (as->a_lastgaphl != NULL) {
 430                 struct seg *hseg = NULL;
 431                 struct seg *lseg = NULL;

 433                 if (as->a_lastgaphl->s_base > newseg->s_base) {
 434                         hseg = as->a_lastgaphl;
 435                         lseg = AVL_PREV(&as->a_segtree, hseg);
 436                 } else {
 437                         lseg = as->a_lastgaphl;
 438                         hseg = AVL_NEXT(&as->a_segtree, lseg);
 439                 }

 441                 if (hseg && lseg && lseg->s_base < newseg->s_base &&
 442                     hseg->s_base > newseg->s_base) {
 443                         avl_insert_here(&as->a_segtree, newseg, lseg,
 444                             AVL_AFTER);
 445                         as->a_lastgaphl = NULL;
 446                         as->a_seglast = newseg;
 447                         return (0);
 448                 }
 449                 as->a_lastgaphl = NULL;
 450         }

 452         addr = newseg->s_base;
 453         eaddr = addr + newseg->s_size;
 454 again:

 456         seg = avl_find(&as->a_segtree, &addr, &where);

 458         if (seg == NULL)
 459                 seg = avl_nearest(&as->a_segtree, where, AVL_AFTER);

 461         if (seg == NULL)
 462                 seg = avl_last(&as->a_segtree);

 464         if (seg != NULL) {
 465                 caddr_t base = seg->s_base;

 467                 /*
```



```
 468                  * If top of seg is below the requested address, then
 469                  * the insertion point is at the end of the linked list,
 470                  * and seg points to the tail of the list.  Otherwise,
 471                  * the insertion point is immediately before seg.
 472                  */
 473                 if (base + seg->s_size > addr) {
 474                         if (addr >= base || eaddr > base) {
 475 #ifdef __sparc
 476                                 extern const struct seg_ops segnf_ops;
 476                                 extern struct seg_ops segnf_ops;

 478                                 /*
 479                                  * no-fault segs must disappear if overlaid.
 480                                  * XXX need new segment type so
 481                                  * we don't have to check s_ops
 482                                  */
 483                                 if (seg->s_ops == &segnf_ops) {
 484                                         seg_unmap(seg);
 485                                         goto again;
 486                                 }
 487 #endif
 488                                 return (-1);    /* overlapping segment */
 489                         }
 490                 }
 491         }
 492         as->a_seglast = newseg;
 493         avl_insert(&as->a_segtree, newseg, where);

 495 #ifdef VERIFY_SEGLIST
 496         as_verify(as);
 497 #endif
 498         return (0);
 499 }
______unchanged_portion_omitted_

2000 /*
2001  * Return the next range within [base, base + len) that is backed
2002  * with "real memory".  Skip holes and non-seg_vn segments.
2003  * We're lazy and only return one segment at a time.
2004  */
2005 int
2006 as_memory(struct as *as, caddr_t *basep, size_t *lenp)
2007 {
2008         extern const struct seg_ops segspt_shmops;      /* needs a header file *
2008         extern struct seg_ops segspt_shmops;    /* needs a header file */
2009         struct seg *seg;
2010         caddr_t addr, eaddr;
2011         caddr_t segend;

2013         AS_LOCK_ENTER(as, &as->a_lock, RW_READER);

2015         addr = *basep;
2016         eaddr = addr + *lenp;

2018         seg = as_findseg(as, addr, 0);
2019         if (seg != NULL)
2020                 addr = MAX(seg->s_base, addr);

2022         for (;;) {
2023                 if (seg == NULL || addr >= eaddr || eaddr <= seg->s_base) {
2024                         AS_LOCK_EXIT(as, &as->a_lock);
2025                         return (EINVAL);
2026                 }

2028                 if (seg->s_ops == &segvn_ops) {
2029                         segend = seg->s_base + seg->s_size;
```

```
2030                            break;
2031                    }

2033                    /*
2034                     * We do ISM by looking into the private data
2035                     * to determine the real size of the segment.
2036                     */
2037                    if (seg->s_ops == &segspt_shmops) {
2038                            segend = seg->s_base + spt_realsize(seg);
2039                            if (addr < segend)
2040                                    break;
2041                    }

2043                    seg = AS_SEGNEXT(as, seg);

2045                    if (seg != NULL)
2046                            addr = seg->s_base;
2047            }

2049            *basep = addr;

2051            if (segend > eaddr)
2052                    *lenp = eaddr - addr;
2053            else
2054                    *lenp = segend - addr;

2056            AS_LOCK_EXIT(as, &as->a_lock);
2057            return (0);
2058 }
_____unchanged_portion_omitted_

2462 /*
2463  * Pagelock pages from a range that spans more than 1 segment.  Obtain shadow
2464  * lists from each segment and copy them to one contiguous shadow list (plist)
2465  * as expected by the caller.  Save pointers to per segment shadow lists at
2466  * the tail of plist so that they can be used during as_pageunlock().
2467  */
2468 static int
2469 as_pagelock_segs(struct as *as, struct seg *seg, struct page ***ppp,
2470     caddr_t addr, size_t size, enum seg_rw rw)
2471 {
2472            caddr_t sv_addr = addr;
2473            size_t sv_size = size;
2474            struct seg *sv_seg = seg;
2475            ulong_t segcnt = 1;
2476            ulong_t cnt;
2477            size_t ssize;
2478            pgcnt_t npages = btop(size);
2479            page_t **plist;
2480            page_t **pl;
2481            int error;
2482            caddr_t eaddr;
2483            faultcode_t fault_err = 0;
2484            pgcnt_t pl_off;
2485            extern const struct seg_ops segspt_shmops;
2485            extern struct seg_ops segspt_shmops;

2487            ASSERT(AS_LOCK_HELD(as, &as->a_lock));
2488            ASSERT(seg != NULL);
2489            ASSERT(addr >= seg->s_base && addr < seg->s_base + seg->s_size);
2490            ASSERT(addr + size > seg->s_base + seg->s_size);
2491            ASSERT(IS_P2ALIGNED(size, PAGESIZE));
2492            ASSERT(IS_P2ALIGNED(addr, PAGESIZE));

2494            /*
2495             * Count the number of segments covered by the range we are about to
```



```
2496             * lock. The segment count is used to size the shadow list we return
2497             * back to the caller.
2498             */
2499            for (; size != 0; size -= ssize, addr += ssize) {
2500                    if (addr >= seg->s_base + seg->s_size) {

2502                            seg = AS_SEGNEXT(as, seg);
2503                            if (seg == NULL || addr != seg->s_base) {
2504                                    AS_LOCK_EXIT(as, &as->a_lock);
2505                                    return (EFAULT);
2506                            }
2507                            /*
2508                             * Do a quick check if subsequent segments
2509                             * will most likely support pagelock.
2510                             */
2511                            if (seg->s_ops == &segvn_ops) {
2512                                    vnode_t *vp;

2514                                    if (segop_getvp(seg, addr, &vp) != 0 ||
2515                                        vp != NULL) {
2516                                            AS_LOCK_EXIT(as, &as->a_lock);
2517                                            goto slow;
2518                                    }
2519                            } else if (seg->s_ops != &segspt_shmops) {
2520                                    AS_LOCK_EXIT(as, &as->a_lock);
2521                                    goto slow;
2522                            }
2523                            segcnt++;
2524                    }
2525                    if (addr + size > seg->s_base + seg->s_size) {
2526                            ssize = seg->s_base + seg->s_size - addr;
2527                    } else {
2528                            ssize = size;
2529                    }
2530            }
2531            ASSERT(segcnt > 1);

2533            plist = kmem_zalloc((npages + segcnt) * sizeof (page_t *), KM_SLEEP);

2535            addr = sv_addr;
2536            size = sv_size;
2537            seg = sv_seg;

2539            for (cnt = 0, pl_off = 0; size != 0; size -= ssize, addr += ssize) {
2540                    if (addr >= seg->s_base + seg->s_size) {
2541                            seg = AS_SEGNEXT(as, seg);
2542                            ASSERT(seg != NULL && addr == seg->s_base);
2543                            cnt++;
2544                            ASSERT(cnt < segcnt);
2545                    }
2546                    if (addr + size > seg->s_base + seg->s_size) {
2547                            ssize = seg->s_base + seg->s_size - addr;
2548                    } else {
2549                            ssize = size;
2550                    }
2551                    pl = &plist[npages + cnt];
2552                    error = segop_pagelock(seg, addr, ssize, (page_t ***)pl,
2553                        L_PAGELOCK, rw);
2554                    if (error) {
2555                            break;
2556                    }
2557                    ASSERT(plist[npages + cnt] != NULL);
2558                    ASSERT(pl_off + btop(ssize) <= npages);
2559                    bcopy(plist[npages + cnt], &plist[pl_off],
2560                        btop(ssize) * sizeof (page_t *));
2561                    pl_off += btop(ssize);
```

```
2562         }

2564         if (size == 0) {
2565                 AS_LOCK_EXIT(as, &as->a_lock);
2566                 ASSERT(cnt == segcnt - 1);
2567                 *ppp = plist;
2568                 return (0);
2569         }

2571         /*
2572          * one of pagelock calls failed. The error type is in error variable.
2573          * Unlock what we've locked so far and retry with F_SOFTLOCK if error
2574          * type is either EFAULT or ENOTSUP. Otherwise just return the error
2575          * back to the caller.
2576          */

2578         eaddr = addr;
2579         seg = sv_seg;

2581         for (cnt = 0, addr = sv_addr; addr < eaddr; addr += ssize) {
2582                 if (addr >= seg->s_base + seg->s_size) {
2583                         seg = AS_SEGNEXT(as, seg);
2584                         ASSERT(seg != NULL && addr == seg->s_base);
2585                         cnt++;
2586                         ASSERT(cnt < segcnt);
2587                 }
2588                 if (eaddr > seg->s_base + seg->s_size) {
2589                         ssize = seg->s_base + seg->s_size - addr;
2590                 } else {
2591                         ssize = eaddr - addr;
2592                 }
2593                 pl = &plist[npages + cnt];
2594                 ASSERT(*pl != NULL);
2595                 (void) segop_pagelock(seg, addr, ssize, (page_t ***)pl,
2596                     L_PAGEUNLOCK, rw);
2597         }

2599         AS_LOCK_EXIT(as, &as->a_lock);

2601         kmem_free(plist, (npages + segcnt) * sizeof (page_t *));

2603         if (error != ENOTSUP && error != EFAULT) {
2604                 return (error);
2605         }

2607 slow:
2608         /*
2609          * If we are here because pagelock failed due to the need to cow fault
2610          * in the pages we want to lock F_SOFTLOCK will do this job and in
2611          * next as_pagelock() call for this address range pagelock will
2612          * hopefully succeed.
2613          */
2614         fault_err = as_fault(as->a_hat, as, sv_addr, sv_size, F_SOFTLOCK, rw);
2615         if (fault_err != 0) {
2616                 return (fc_decode(fault_err));
2617         }
2618         *ppp = NULL;

2620         return (0);
2621 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   55156 Fri May  8 18:04:58 2015
new/usr/src/uts/common/vm/vm_seg.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
______unchanged_portion_omitted_

 183 #define seg_pdisabled                   pctrl1.p_disabled
 184 #define seg_pmaxwindow                  pctrl1.p_maxwin
 185 #define seg_phashsize_win               pctrl1.p_hashwin_sz
 186 #define seg_phashtab_win                pctrl1.p_htabwin
 187 #define seg_phashsize_wired             pctrl1.p_hashwired_sz
 188 #define seg_phashtab_wired              pctrl1.p_htabwired
 189 #define seg_pkmcache                    pctrl1.p_kmcache
 190 #define seg_pmem_mtx                    pctrl2.p_mem_mtx
 191 #define seg_plocked_window              pctrl2.p_locked_win
 192 #define seg_plocked                     pctrl2.p_locked
 193 #define seg_pahcur                      pctrl2.p_ahcur
 194 #define seg_pathr_on                    pctrl2.p_athr_on
 195 #define seg_pahhead                     pctrl2.p_ahhead
 196 #define seg_pmax_pcpage                 pctrl3.p_pcp_maxage
 197 #define seg_pathr_empty_ahb             pctrl3.p_athr_empty_ahb
 198 #define seg_pathr_full_ahb              pctrl3.p_athr_full_ahb
 199 #define seg_pshrink_shift               pctrl3.p_shrink_shft
 200 #define seg_pmaxapurge_npages           pctrl3.p_maxapurge_npages

 202 #define P_HASHWIN_MASK                  (seg_phashsize_win - 1)
 203 #define P_HASHWIRED_MASK                (seg_phashsize_wired - 1)
 204 #define P_BASESHIFT                     (6)

 206 kthread_t *seg_pasync_thr;

 208 extern const struct seg_ops segvn_ops;
 209 extern const struct seg_ops segspt_shmops;
 208 extern struct seg_ops segvn_ops;
 209 extern struct seg_ops segspt_shmops;

 211 #define IS_PFLAGS_WIRED(flags) ((flags) & SEGP_FORCE_WIRED)
 212 #define IS_PCP_WIRED(pcp) IS_PFLAGS_WIRED((pcp)->p_flags)

 214 #define LBOLT_DELTA(t)  ((ulong_t)(ddi_get_lbolt() - (t)))

 216 #define PCP_AGE(pcp)    LBOLT_DELTA((pcp)->p_lbolt)

 218 /*
 219  * htag0 argument can be a seg or amp pointer.
 220  */
 221 #define P_HASHBP(seg, htag0, addr, flags)                               \
 222         (IS_PFLAGS_WIRED((flags)) ?                                     \
 223             ((struct seg_phash *)&seg_phashtab_wired[P_HASHWIRED_MASK & \
 224             ((uintptr_t)(htag0) >> P_BASESHIFT)]) :                     \
 225             (&seg_phashtab_win[P_HASHWIN_MASK &                         \
 226             (((uintptr_t)(htag0) >> 3) ^                                \
 227             ((uintptr_t)(addr) >> ((flags & SEGP_PSHIFT) ?              \
 228             (flags >> 16) : page_get_shift((seg)->s_szc))))]))

 230 /*
 231  * htag0 argument can be a seg or amp pointer.
 232  */
 233 #define P_MATCH(pcp, htag0, addr, len)                                  \
 234         ((pcp)->p_htag0 == (htag0) &&                                   \
 235         (pcp)->p_addr == (addr) &&                                      \
 236         (pcp)->p_len >= (len))

 238 #define P_MATCH_PP(pcp, htag0, addr, len, pp)                           \
```



```
 239         ((pcp)->p_pp == (pp) &&                                         \
 240         (pcp)->p_htag0 == (htag0) &&                                    \
 241         (pcp)->p_addr == (addr) &&                                      \
 242         (pcp)->p_len >= (len))

 244 #define plink2pcache(pl)        ((struct seg_pcache *)((uintptr_t)(pl) - \
 245     offsetof(struct seg_pcache, p_plink)))

 247 #define hlink2phash(hl, l)      ((struct seg_phash *)((uintptr_t)(hl) - \
 248     offsetof(struct seg_phash, p_halink[l])))

 250 /*
 251  * seg_padd_abuck()/seg_premove_abuck() link and unlink hash buckets from
 252  * active hash bucket lists. We maintain active bucket lists to reduce the
 253  * overhead of finding active buckets during asynchronous purging since there
 254  * can be 10s of millions of buckets on a large system but only a small subset
 255  * of them in actual use.
 256  *
 257  * There're 2 active bucket lists. Current active list (as per seg_pahcur) is
 258  * used by seg_pinsert()/seg_pinactive()/seg_ppurge() to add and delete
 259  * buckets. The other list is used by asynchronous purge thread. This allows
 260  * the purge thread to walk its active list without holding seg_pmem_mtx for a
 261  * long time. When asynchronous thread is done with its list it switches to
 262  * current active list and makes the list it just finished processing as
 263  * current active list.
 264  *
 265  * seg_padd_abuck() only adds the bucket to current list if the bucket is not
 266  * yet on any list.  seg_premove_abuck() may remove the bucket from either
 267  * list. If the bucket is on current list it will be always removed. Otherwise
 268  * the bucket is only removed if asynchronous purge thread is not currently
 269  * running or seg_premove_abuck() is called by asynchronous purge thread
 270  * itself. A given bucket can only be on one of active lists at a time. These
 271  * routines should be called with per bucket lock held.  The routines use
 272  * seg_pmem_mtx to protect list updates. seg_padd_abuck() must be called after
 273  * the first entry is added to the bucket chain and seg_premove_abuck() must
 274  * be called after the last pcp entry is deleted from its chain. Per bucket
 275  * lock should be held by the callers.  This avoids a potential race condition
 276  * when seg_premove_abuck() removes a bucket after pcp entries are added to
 277  * its list after the caller checked that the bucket has no entries. (this
 278  * race would cause a loss of an active bucket from the active lists).
 279  *
 280  * Both lists are circular doubly linked lists anchored at seg_pahhead heads.
 281  * New entries are added to the end of the list since LRU is used as the
 282  * purging policy.
 283  */
 284 static void
 285 seg_padd_abuck(struct seg_phash *hp)
 286 {
 287         int lix;

 289         ASSERT(MUTEX_HELD(&hp->p_hmutex));
 290         ASSERT((struct seg_phash *)hp->p_hnext != hp);
 291         ASSERT((struct seg_phash *)hp->p_hprev != hp);
 292         ASSERT(hp->p_hnext == hp->p_hprev);
 293         ASSERT(!IS_PCP_WIRED(hp->p_hnext));
 294         ASSERT(hp->p_hnext->p_hnext == (struct seg_pcache *)hp);
 295         ASSERT(hp->p_hprev->p_hprev == (struct seg_pcache *)hp);
 296         ASSERT(hp >= seg_phashtab_win &&
 297             hp < &seg_phashtab_win[seg_phashsize_win]);

 299         /*
 300          * This bucket can already be on one of active lists
 301          * since seg_premove_abuck() may have failed to remove it
 302          * before.
 303          */
 304         mutex_enter(&seg_pmem_mtx);
```

```
305         lix = seg_pahcur;
306         ASSERT(lix >= 0 && lix <= 1);
307         if (hp->p_halink[lix].p_lnext != NULL) {
308                 ASSERT(hp->p_halink[lix].p_lprev != NULL);
309                 ASSERT(hp->p_halink[!lix].p_lnext == NULL);
310                 ASSERT(hp->p_halink[!lix].p_lprev == NULL);
311                 mutex_exit(&seg_pmem_mtx);
312                 return;
313         }
314         ASSERT(hp->p_halink[lix].p_lprev == NULL);

316         /*
317          * If this bucket is still on list !lix async thread can't yet remove
318          * it since we hold here per bucket lock. In this case just return
319          * since async thread will eventually find and process this bucket.
320          */
321         if (hp->p_halink[!lix].p_lnext != NULL) {
322                 ASSERT(hp->p_halink[!lix].p_lprev != NULL);
323                 mutex_exit(&seg_pmem_mtx);
324                 return;
325         }
326         ASSERT(hp->p_halink[!lix].p_lprev == NULL);
327         /*
328          * This bucket is not on any active bucket list yet.
329          * Add the bucket to the tail of current active list.
330          */
331         hp->p_halink[lix].p_lnext = &seg_pahead[lix];
332         hp->p_halink[lix].p_lprev = seg_pahead[lix].p_lprev;
333         seg_pahead[lix].p_lprev->p_lnext = &hp->p_halink[lix];
334         seg_pahead[lix].p_lprev = &hp->p_halink[lix];
335         mutex_exit(&seg_pmem_mtx);
336 }
```

unchanged_portion_omitted_

****
57936 Fri May 8 18:04:59 2015
new/usr/src/uts/common/vm/vm_usage.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
****
_____***unchanged_portion_omitted_***

```
 297 extern struct as kas;
 298 extern proc_t *practive;
 299 extern zone_t *global_zone;
 300 extern const struct seg_ops segvn_ops;
 301 extern const struct seg_ops segspt_shmops;
 300 extern struct seg_ops segvn_ops;
 301 extern struct seg_ops segspt_shmops;

 303 static vmu_data_t vmu_data;
 304 static kmem_cache_t *vmu_bound_cache;
 305 static kmem_cache_t *vmu_object_cache;

 307 /*
 308  * Comparison routine for AVL tree. We base our comparison on vmb_start.
 309  */
 310 static int
 311 bounds_cmp(const void *bnd1, const void *bnd2)
 312 {
 313         const vmu_bound_t *bound1 = bnd1;
 314         const vmu_bound_t *bound2 = bnd2;

 316         if (bound1->vmb_start == bound2->vmb_start) {
 317                 return (0);
 318         }
 319         if (bound1->vmb_start < bound2->vmb_start) {
 320                 return (-1);
 321         }

 323         return (1);
 324 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  142125 Fri May  8 18:04:59 2015
new/usr/src/uts/i86pc/io/rootnex.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
______unchanged_portion_omitted_
```

```
 343 #endif

 345 /*
 346  *  extern hacks
 347  */
 348 extern const struct seg_ops segdev_ops;
 348 extern struct seg_ops segdev_ops;
 349 extern int ignore_hardware_nodes;       /* force flag from ddi_impl.c */
 350 #ifdef  DDI_MAP_DEBUG
 351 extern int ddi_map_debug_flag;
 352 #define ddi_map_debug   if (ddi_map_debug_flag) prom_printf
 353 #endif
 354 extern void i86_pp_map(page_t *pp, caddr_t kaddr);
 355 extern void i86_va_map(caddr_t vaddr, struct as *asp, caddr_t kaddr);
 356 extern int (*psm_intr_ops)(dev_info_t *, ddi_intr_handle_impl_t *,
 357     psm_intr_op_t, int *);
 358 extern int impl_ddi_sunbus_initchild(dev_info_t *dip);
 359 extern void impl_ddi_sunbus_removechild(dev_info_t *dip);

 361 /*
 362  * Use device arena to use for device control register mappings.
 363  * Various kernel memory walkers (debugger, dtrace) need to know
 364  * to avoid this address range to prevent undesired device activity.
 365  */
 366 extern void *device_arena_alloc(size_t size, int vm_flag);
 367 extern void device_arena_free(void * vaddr, size_t size);


 370 /*
 371  *  Internal functions
 372  */
 373 static int rootnex_dma_init();
 374 static void rootnex_add_props(dev_info_t *);
 375 static int rootnex_ctl_reportdev(dev_info_t *dip);
 376 static struct intrspec *rootnex_get_ispec(dev_info_t *rdip, int inum);
 377 static int rootnex_map_regspec(ddi_map_req_t *mp, caddr_t *vaddrp);
 378 static int rootnex_unmap_regspec(ddi_map_req_t *mp, caddr_t *vaddrp);
 379 static int rootnex_map_handle(ddi_map_req_t *mp);
 380 static void rootnex_clean_dmahdl(ddi_dma_impl_t *hp);
 381 static int rootnex_valid_alloc_parms(ddi_dma_attr_t *attr, uint_t maxsegsize);
 382 static int rootnex_valid_bind_parms(ddi_dma_req_t *dmareq,
 383     ddi_dma_attr_t *attr);
 384 static void rootnex_get_sgl(ddi_dma_obj_t *dmar_object, ddi_dma_cookie_t *sgl,
 385     rootnex_sglinfo_t *sglinfo);
 386 static void rootnex_dvma_get_sgl(ddi_dma_obj_t *dmar_object,
 387     ddi_dma_cookie_t *sgl, rootnex_sglinfo_t *sglinfo);
 388 static int rootnex_bind_slowpath(ddi_dma_impl_t *hp, struct ddi_dma_req *dmareq,
 389     rootnex_dma_t *dma, ddi_dma_attr_t *attr, ddi_dma_obj_t *dmao, int kmflag);
 390 static int rootnex_setup_copybuf(ddi_dma_impl_t *hp, struct ddi_dma_req *dmareq,
 391     rootnex_dma_t *dma, ddi_dma_attr_t *attr);
 392 static void rootnex_teardown_copybuf(rootnex_dma_t *dma);
 393 static int rootnex_setup_windows(ddi_dma_impl_t *hp, rootnex_dma_t *dma,
 394     ddi_dma_attr_t *attr, ddi_dma_obj_t *dmao, int kmflag);
 395 static void rootnex_teardown_windows(rootnex_dma_t *dma);
 396 static void rootnex_init_win(ddi_dma_impl_t *hp, rootnex_dma_t *dma,
 397     rootnex_window_t *window, ddi_dma_cookie_t *cookie, off_t cur_offset);
 398 static void rootnex_setup_cookie(ddi_dma_obj_t *dmar_object,
 399     rootnex_dma_t *dma, ddi_dma_cookie_t *cookie, off_t cur_offset,
 400     size_t *copybuf_used, page_t **cur_pp);
```



```
 401 static int rootnex_sgllen_window_boundary(ddi_dma_impl_t *hp,
 402     rootnex_dma_t *dma, rootnex_window_t **windowp, ddi_dma_cookie_t *cookie,
 403     ddi_dma_attr_t *attr, off_t cur_offset);
 404 static int rootnex_copybuf_window_boundary(ddi_dma_impl_t *hp,
 405     rootnex_dma_t *dma, rootnex_window_t **windowp,
 406     ddi_dma_cookie_t *cookie, off_t cur_offset, size_t *copybuf_used);
 407 static int rootnex_maxxfer_window_boundary(ddi_dma_impl_t *hp,
 408     rootnex_dma_t *dma, rootnex_window_t **windowp, ddi_dma_cookie_t *cookie);
 409 static int rootnex_valid_sync_parms(ddi_dma_impl_t *hp, rootnex_window_t *win,
 410     off_t offset, size_t size, uint_t cache_flags);
 411 static int rootnex_verify_buffer(rootnex_dma_t *dma);
 412 static int rootnex_dma_check(dev_info_t *dip, const void *handle,
 413     const void *comp_addr, const void *not_used);
 414 static boolean_t rootnex_need_bounce_seg(ddi_dma_obj_t *dmar_object,
 415     rootnex_sglinfo_t *sglinfo);
 416 static struct as *rootnex_get_as(ddi_dma_obj_t *dmar_object);

 418 /*
 419  * _init()
 420  *
 421  */
 422 int
 423 _init(void)
 424 {

 426         rootnex_state = NULL;
 427         return (mod_install(&rootnex_modlinkage));
 428 }
```

```
______unchanged_portion_omitted_
```

```
**********************************************************
   16548 Fri May  8 18:04:59 2015
new/usr/src/uts/i86xpv/vm/seg_mf.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
______unchanged_portion_omitted_

 102 static const struct seg_ops segmf_ops;
 102 static struct seg_ops segmf_ops;

 104 static int segmf_fault_gref_range(struct seg *seg, caddr_t addr, size_t len);

 106 static struct segmf_data *
 107 segmf_data_zalloc(struct seg *seg)
 108 {
 109         struct segmf_data *data = kmem_zalloc(sizeof (*data), KM_SLEEP);

 111         mutex_init(&data->lock, "segmf.lock", MUTEX_DEFAULT, NULL);
 112         seg->s_ops = &segmf_ops;
 113         seg->s_data = data;
 114         return (data);
 115 }
______unchanged_portion_omitted_

 739 static const struct seg_ops segmf_ops = {
 739 static struct seg_ops segmf_ops = {
 740         .dup            = segmf_dup,
 741         .unmap          = segmf_unmap,
 742         .free           = segmf_free,
 743         .fault          = segmf_fault,
 744         .faulta         = segmf_faulta,
 745         .setprot        = segmf_setprot,
 746         .checkprot      = segmf_checkprot,
 747         .kluster        = segmf_kluster,
 748         .sync           = segmf_sync,
 749         .incore         = segmf_incore,
 750         .lockop         = segmf_lockop,
 751         .getprot        = segmf_getprot,
 752         .getoffset      = segmf_getoffset,
 753         .gettype        = segmf_gettype,
 754         .getvp          = segmf_getvp,
 755         .advise         = segmf_advise,
 756         .dump           = segmf_dump,
 757         .pagelock       = segmf_pagelock,
 758         .getmemid       = segmf_getmemid,
 759 };
______unchanged_portion_omitted_
```

```
**********************************************************
   11644 Fri May  8 18:04:59 2015
new/usr/src/uts/sparc/v9/vm/seg_nf.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
**********************************************************
```

```
   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 2006 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 - segment for non-faulting loads.
  36  */

  38 #include <sys/types.h>
  39 #include <sys/t_lock.h>
  40 #include <sys/param.h>
  41 #include <sys/mman.h>
  42 #include <sys/errno.h>
  43 #include <sys/kmem.h>
  44 #include <sys/cmn_err.h>
  45 #include <sys/vnode.h>
  46 #include <sys/proc.h>
  47 #include <sys/conf.h>
  48 #include <sys/debug.h>
  49 #include <sys/archsystm.h>
  50 #include <sys/lgrp.h>

  52 #include <vm/page.h>
  53 #include <vm/hat.h>
  54 #include <vm/as.h>
  55 #include <vm/seg.h>
  56 #include <vm/vpage.h>

  58 /*
  59  * Private seg op routines.
  60  */
```



```
  61 static int      segnf_dup(struct seg *seg, struct seg *newseg);
  62 static int      segnf_unmap(struct seg *seg, caddr_t addr, size_t len);
  63 static void     segnf_free(struct seg *seg);
  64 static faultcode_t segnf_nomap(void);
  65 static int      segnf_setprot(struct seg *seg, caddr_t addr,
  66                     size_t len, uint_t prot);
  67 static int      segnf_checkprot(struct seg *seg, caddr_t addr,
  68                     size_t len, uint_t prot);
  69 static int      segnf_nop(void);
  70 static int      segnf_getprot(struct seg *seg, caddr_t addr,
  71                     size_t len, uint_t *protv);
  72 static u_offset_t segnf_getoffset(struct seg *seg, caddr_t addr);
  73 static int      segnf_gettype(struct seg *seg, caddr_t addr);
  74 static int      segnf_getvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
  75 static void     segnf_dump(struct seg *seg);
  76 static int      segnf_pagelock(struct seg *seg, caddr_t addr, size_t len,
  77                     struct page ***ppp, enum lock_type type, enum seg_rw rw);


  80 const struct seg_ops segnf_ops = {
  80 struct seg_ops segnf_ops = {
  81         .dup            = segnf_dup,
  82         .unmap          = segnf_unmap,
  83         .free           = segnf_free,
  84         .fault          = (faultcode_t (*)(struct hat *, struct seg *, caddr_t,
  85             size_t, enum fault_type, enum seg_rw))segnf_nomap,
  86         .faulta         = (faultcode_t (*)(struct seg *, caddr_t)) segnf_nomap,
  87         .setprot        = segnf_setprot,
  88         .checkprot      = segnf_checkprot,
  89         .sync           = (int (*)(struct seg *, caddr_t, size_t, int, uint_t))
  90                 segnf_nop,
  91         .incore         = (size_t (*)(struct seg *, caddr_t, size_t, char *))
  92                 segnf_nop,
  93         .lockop         = (int (*)(struct seg *, caddr_t, size_t, int, int,
  94             ulong_t *, size_t))segnf_nop,
  95         .getprot        = segnf_getprot,
  96         .getoffset      = segnf_getoffset,
  97         .gettype        = segnf_gettype,
  98         .getvp          = segnf_getvp,
  99         .advise         = (int (*)(struct seg *, caddr_t, size_t, uint_t))
 100                 segnf_nop,
 101         .dump           = segnf_dump,
 102         .pagelock       = segnf_pagelock,
 103 };
```

_____*unchanged_portion_omitted_*

```
***********************************************************
   23606 Fri May  8 18:05:00 2015
new/usr/src/uts/sun4/io/rootnex.c
const-ify make segment ops structures
There is no reason to keep the segment ops structures writable.
***********************************************************
______unchanged_portion_omitted_


 697 /*
 698  * Shorthand defines
 699  */

 701 #define DMAOBJ_PP_PP    dmao_obj.pp_obj.pp_pp
 702 #define DMAOBJ_PP_OFF   dmao_ogj.pp_obj.pp_offset
 703 #define ALO             dma_lim->dlim_addr_lo
 704 #define AHI             dma_lim->dlim_addr_hi
 705 #define OBJSIZE         dmareq->dmar_object.dmao_size
 706 #define ORIGVADDR       dmareq->dmar_object.dmao_obj.virt_obj.v_addr
 707 #define RED             ((mp->dmai_rflags & DDI_DMA_REDZONE)? 1 : 0)
 708 #define DIRECTION       (mp->dmai_rflags & DDI_DMA_RDWR)

 710 /*
 711  * rootnex_map_fault:
 712  *
 713  *      fault in mappings for requestors
 714  */

 716 /*ARGSUSED*/
 717 static int
 718 rootnex_map_fault(dev_info_t *dip, dev_info_t *rdip,
 719     struct hat *hat, struct seg *seg, caddr_t addr,
 720     struct devpage *dp, pfn_t pfn, uint_t prot, uint_t lock)
 721 {
 722         extern const struct seg_ops segdev_ops;
 722         extern struct seg_ops segdev_ops;

 724         DPRINTF(ROOTNEX_MAP_DEBUG, ("rootnex_map_fault: address <%p> "
 725             "pfn <%lx>", (void *)addr, pfn));
 726         DPRINTF(ROOTNEX_MAP_DEBUG, (" Seg <%s>\n",
 727             seg->s_ops == &segdev_ops ? "segdev" :
 728             seg == &kvseg ? "segkmem" : "NONE!"));

 730         /*
 731          * This is all terribly broken, but it is a start
 732          *
 733          * XXX  Note that this test means that segdev_ops
 734          *      must be exported from seg_dev.c.
 735          * XXX  What about devices with their own segment drivers?
 736          */
 737         if (seg->s_ops == &segdev_ops) {
 738                 register struct segdev_data *sdp =
 739                     (struct segdev_data *)seg->s_data;

 741                 if (hat == NULL) {
 742                         /*
 743                          * This is one plausible interpretation of
 744                          * a null hat i.e. use the first hat on the
 745                          * address space hat list which by convention is
 746                          * the hat of the system MMU.  At alternative
 747                          * would be to panic .. this might well be better ..
 748                          */
 749                         ASSERT(AS_READ_HELD(seg->s_as, &seg->s_as->a_lock));
 750                         hat = seg->s_as->a_hat;
 751                         cmn_err(CE_NOTE, "rootnex_map_fault: nil hat");
 752                 }
```



```
 753                 hat_devload(hat, addr, MMU_PAGESIZE, pfn, prot | sdp->hat_attr,
 754                     (lock ? HAT_LOAD_LOCK : HAT_LOAD));
 755         } else if (seg == &kvseg && dp == (struct devpage *)0) {
 756                 hat_devload(kas.a_hat, addr, MMU_PAGESIZE, pfn, prot,
 757                     HAT_LOAD_LOCK);
 758         } else
 759                 return (DDI_FAILURE);
 760         return (DDI_SUCCESS);
 761 }
______unchanged_portion_omitted_
```