```
**********************************************************
   55638 Tue Nov 24 09:35:13 2015
new/usr/src/uts/common/exec/elf/elf.c
6154 const-ify segment ops structures
**********************************************************
______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_***

```
**********************************************************
  112698 Tue Nov 24 09:35:13 2015
new/usr/src/uts/common/fs/proc/prsubr.c
6154 const-ify segment ops structures
**********************************************************
______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 Tue Nov 24 09:35:13 2015
new/usr/src/uts/common/os/sunddi.c
6154 const-ify segment ops structures
**********************************************************
______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_***

```
**********************************************************
    8552 Tue Nov 24 09:35:13 2015
new/usr/src/uts/common/os/urw.c
6154 const-ify segment ops structures
**********************************************************
```

```
/*
 * 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   */

#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);
}
______unchanged_portion_omitted_
```

****
10197 Tue Nov 24 09:35:13 2015
new/usr/src/uts/common/vm/seg.h
6154 const-ify segment ops structures
****
_____***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_***

```
**********************************************************
  113266 Tue Nov 24 09:35:14 2015
new/usr/src/uts/common/vm/seg_dev.c
6154 const-ify segment ops structures
**********************************************************
```

```
/*
 * 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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      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 - segment of a mapped device.
 *
 * This segment driver is used when mapping character special devices.
 */

#include <sys/types.h>
#include <sys/t_lock.h>
#include <sys/sysmacros.h>
#include <sys/vtrace.h>
#include <sys/systm.h>
#include <sys/vmsystm.h>
#include <sys/mman.h>
#include <sys/errno.h>
#include <sys/kmem.h>
#include <sys/cmn_err.h>
#include <sys/vnode.h>
#include <sys/proc.h>
#include <sys/conf.h>
#include <sys/debug.h>
#include <sys/ddidevmap.h>
#include <sys/ddi_implfuncs.h>
```



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

#include <vm/page.h>
#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_dev.h>
#include <vm/seg_kp.h>
#include <vm/seg_kmem.h>
#include <vm/vpage.h>

#include <sys/sunddi.h>
#include <sys/esunddi.h>
#include <sys/fs/snode.h>


#if DEBUG
int segdev_debug;
#define DEBUGF(level, args) { if (segdev_debug >= (level)) cmn_err args; }
#else
#define DEBUGF(level, args)
#endif

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

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

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

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

/*
 * VA_PA_ALIGNED checks to see if both VA and PA are on pgsize boundary
 * VA_PA_PGSIZE_ALIGNED check to see if VA is aligned with PA w.r.t. pgsize
 */
#define VA_PA_ALIGNED(uvaddr, paddr, pgsize)            \
        (((uvaddr | paddr) & (pgsize - 1)) == 0)
#define VA_PA_PGSIZE_ALIGNED(uvaddr, paddr, pgsize)     \
        (((uvaddr ^ paddr) & (pgsize - 1)) == 0)

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

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

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

/*
 * mutex, vnode and page for the page of zeros we use for the trash mappings.
 * One trash page is allocated on the first ddi_umem_setup call that uses it
 * XXX Eventually, we may want to combine this with what segnf does when all
 * hat layers implement HAT_NOFAULT.
 *
 * The trash page is used when the backing store for a userland mapping is
 * removed but the application semantics do not take kindly to a SIGBUS.
 * In that scenario, the applications pages are mapped to some dummy page
 * which returns garbage on read and writes go into a common place.
 * (Perfect for NO_FAULT semantics)
 * The device driver is responsible to communicating to the app with some
 * other mechanism that such remapping has happened and the app should take
 * corrective action.
 * 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 int      segdev_pagelock(struct seg *, caddr_t, size_t,
 182                     struct page ***, enum lock_type, enum seg_rw);
 183 static int      segdev_getmemid(struct seg *, caddr_t, memid_t *);

 185 /*
 186  * XXX  this struct is used by rootnex_map_fault to identify
 187  *      the segment it has been passed. So if you make it
 188  *      "static" you'll need to fix rootnex_map_fault.
 189  */
 190 const struct seg_ops segdev_ops = {
 190 struct seg_ops segdev_ops = {
 191         .dup            = segdev_dup,
 192         .unmap          = segdev_unmap,
```



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

*unchanged_portion_omitted_*

****
4477 Tue Nov 24 09:35:14 2015
new/usr/src/uts/common/vm/seg_dev.h
6154 const-ify segment ops structures
****
_____***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) & \
 122             (MAP_SHARED | MAP_PRIVATE)) == 0))

 124 extern void segdev_init(void);

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

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

 132 #endif  /* _KERNEL */

 134 #ifdef  __cplusplus
 135 }
```

_____***unchanged_portion_omitted_***

****
45289 Tue Nov 24 09:35:14 2015
new/usr/src/uts/common/vm/seg_kmem.c
6154 const-ify segment ops structures
****
_____***unchanged_portion_omitted_***

```
 769 static const struct seg_ops segkmem_ops = {
 769 static struct seg_ops segkmem_ops = {
 770         .dup            = SEGKMEM_BADOP(int),
 771         .unmap          = SEGKMEM_BADOP(int),
 772         .free           = SEGKMEM_BADOP(void),
 773         .fault          = segkmem_fault,
 774         .faulta         = SEGKMEM_BADOP(faultcode_t),
 775         .setprot        = segkmem_setprot,
 776         .checkprot      = segkmem_checkprot,
 777         .kluster        = segkmem_kluster,
 778         .swapout        = SEGKMEM_BADOP(size_t),
 779         .sync           = SEGKMEM_BADOP(int),
 780         .incore         = SEGKMEM_BADOP(size_t),
 781         .lockop         = SEGKMEM_BADOP(int),
 782         .getprot        = SEGKMEM_BADOP(int),
 783         .getoffset      = SEGKMEM_BADOP(u_offset_t),
 784         .gettype        = SEGKMEM_BADOP(int),
 785         .getvp          = SEGKMEM_BADOP(int),
 786         .advise         = SEGKMEM_BADOP(int),
 787         .dump           = segkmem_dump,
 788         .pagelock       = segkmem_pagelock,
 789         .setpagesize    = SEGKMEM_BADOP(int),
 790         .getmemid       = segkmem_getmemid,
 791         .capable        = segkmem_capable,
 792 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   36501 Tue Nov 24 09:35:14 2015
new/usr/src/uts/common/vm/seg_kp.c
6154 const-ify segment ops structures
**********************************************************
```

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

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

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

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

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



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

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

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

  98 /*
  99  * Lock used to protect the hash table(s) and caches.
 100  */
 101 static kmutex_t segkp_lock;

 103 /*
 104  * The segkp caches
 105  */
 106 static struct segkp_cache segkp_cache[SEGKP_MAX_CACHE];

 108 #define SEGKP_BADOP(t)  (t(*)())segkp_badop

 110 /*
 111  * When there are fewer than red_minavail bytes left on the stack,
 112  * segkp_map_red() will map in the redzone (if called).  5000 seems
 113  * to work reasonably well...
 114  */
 115 long            red_minavail = 5000;

 117 /*
 118  * will be set to 1 for 32 bit x86 systems only, in startup.c
 119  */
 120 int     segkp_fromheap = 0;
 121 ulong_t *segkp_bitmap;

 123 /*
 124  * If segkp_map_red() is called with the redzone already mapped and
 125  * with less than RED_DEEP_THRESHOLD bytes available on the stack,
 126  * then the stack situation has become quite serious;  if much more stack
 127  * is consumed, we have the potential of scrogging the next thread/LWP
```

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

 134 hrtime_t        red_deep_hires;
 135 kthread_t       *red_deep_thread;

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

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

 144 static const struct seg_ops segkp_ops = {
 144 static struct   seg_ops segkp_ops = {
 145         .dup            = SEGKP_BADOP(int),
 146         .unmap          = SEGKP_BADOP(int),
 147         .free           = SEGKP_BADOP(void),
 148         .fault          = segkp_fault,
 149         .faulta         = SEGKP_BADOP(faultcode_t),
 150         .setprot        = SEGKP_BADOP(int),
 151         .checkprot      = segkp_checkprot,
 152         .kluster        = segkp_kluster,
 153         .swapout        = SEGKP_BADOP(size_t),
 154         .sync           = SEGKP_BADOP(int),
 155         .incore         = SEGKP_BADOP(size_t),
 156         .lockop         = SEGKP_BADOP(int),
 157         .getprot        = SEGKP_BADOP(int),
 158         .getoffset      = SEGKP_BADOP(u_offset_t),
 159         .gettype        = SEGKP_BADOP(int),
 160         .getvp          = SEGKP_BADOP(int),
 161         .advise         = SEGKP_BADOP(int),
 162         .dump           = segkp_dump,
 163         .pagelock       = segkp_pagelock,
 164         .setpagesize    = SEGKP_BADOP(int),
 165 };
_____unchanged_portion_omitted_
```

```
**********************************************************
    9272 Tue Nov 24 09:35:14 2015
new/usr/src/uts/common/vm/seg_kpm.c
6154 const-ify segment ops structures
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */
  22 /*
  23  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */

  27 /*
  28  * Kernel Physical Mapping (kpm) segment driver (segkpm).
  29  *
  30  * This driver delivers along with the hat_kpm* interfaces an alternative
  31  * mechanism for kernel mappings within the 64-bit Solaris operating system,
  32  * which allows the mapping of all physical memory into the kernel address
  33  * space at once. This is feasible in 64 bit kernels, e.g. for Ultrasparc II
  34  * and beyond processors, since the available VA range is much larger than
  35  * possible physical memory. Momentarily all physical memory is supported,
  36  * that is represented by the list of memory segments (memsegs).
  37  *
  38  * Segkpm mappings have also very low overhead and large pages are used
  39  * (when possible) to minimize the TLB and TSB footprint. It is also
  40  * extentable for other than Sparc architectures (e.g. AMD64). Main
  41  * advantage is the avoidance of the TLB-shootdown X-calls, which are
  42  * normally needed when a kernel (global) mapping has to be removed.
  43  *
  44  * First example of a kernel facility that uses the segkpm mapping scheme
  45  * is seg_map, where it is used as an alternative to hat_memload().
  46  * See also hat layer for more information about the hat_kpm* routines.
  47  * The kpm facilty can be turned off at boot time (e.g. /etc/system).
  48  */

  50 #include <sys/types.h>
  51 #include <sys/param.h>
  52 #include <sys/sysmacros.h>
  53 #include <sys/systm.h>
  54 #include <sys/vnode.h>
  55 #include <sys/cmn_err.h>
  56 #include <sys/debug.h>
  57 #include <sys/thread.h>
  58 #include <sys/cpuvar.h>
  59 #include <sys/bitmap.h>
  60 #include <sys/atomic.h>
  61 #include <sys/lgrp.h>
```



```
  63 #include <vm/seg_kmem.h>
  64 #include <vm/seg_kpm.h>
  65 #include <vm/hat.h>
  66 #include <vm/as.h>
  67 #include <vm/seg.h>
  68 #include <vm/page.h>

  70 /*
  71  * Global kpm controls.
  72  * See also platform and mmu specific controls.
  73  *
  74  * kpm_enable -- global on/off switch for segkpm.
  75  * . Set by default on 64bit platforms that have kpm support.
  76  * . Will be disabled from platform layer if not supported.
  77  * . Can be disabled via /etc/system.
  78  *
  79  * kpm_smallpages -- use only regular/system pagesize for kpm mappings.
  80  * . Can be useful for critical debugging of kpm clients.
  81  * . Set to zero by default for platforms that support kpm large pages.
  82  *   The use of kpm large pages reduces the footprint of kpm meta data
  83  *   and has all the other advantages of using large pages (e.g TLB
  84  *   miss reduction).
  85  * . Set by default for platforms that don't support kpm large pages or
  86  *   where large pages cannot be used for other reasons (e.g. there are
  87  *   only few full associative TLB entries available for large pages).
  88  *
  89  * segmap_kpm -- separate on/off switch for segmap using segkpm:
  90  * . Set by default.
  91  * . Will be disabled when kpm_enable is zero.
  92  * . Will be disabled when MAXBSIZE != PAGESIZE.
  93  * . Can be disabled via /etc/system.
  94  *
  95  */
  96 int kpm_enable = 1;
  97 int kpm_smallpages = 0;
  98 int segmap_kpm = 1;

 100 /*
 101  * Private seg op routines.
 102  */
 103 faultcode_t segkpm_fault(struct hat *hat, struct seg *seg, caddr_t addr,
 104                          size_t len, enum fault_type type, enum seg_rw rw);
 105 static void     segkpm_badop(void);

 107 #define SEGKPM_BADOP(t) (t(*)())segkpm_badop

 109 static const struct seg_ops segkpm_ops = {
 109 static struct seg_ops segkpm_ops = {
 110         .dup            = SEGKPM_BADOP(int),
 111         .unmap          = SEGKPM_BADOP(int),
 112         .free           = SEGKPM_BADOP(void),
 113         .fault          = segkpm_fault,
 114         .faulta         = SEGKPM_BADOP(int),
 115         .setprot        = SEGKPM_BADOP(int),
 116         .checkprot      = SEGKPM_BADOP(int),
 117         .kluster        = SEGKPM_BADOP(int),
 118         .swapout        = SEGKPM_BADOP(size_t),
 119         .sync           = SEGKPM_BADOP(int),
 120         .incore         = SEGKPM_BADOP(size_t),
 121         .lockop         = SEGKPM_BADOP(int),
 122         .getprot        = SEGKPM_BADOP(int),
 123         .getoffset      = SEGKPM_BADOP(u_offset_t),
 124         .gettype        = SEGKPM_BADOP(int),
 125         .getvp          = SEGKPM_BADOP(int),
 126         .advise         = SEGKPM_BADOP(int),
```

```
 127         .setpagesize    = SEGKPM_BADOP(int),
 128         .getmemid       = SEGKPM_BADOP(int),
 129         .getpolicy      = SEGKPM_BADOP(lgrp_mem_policy_info_t *),
 130 };
______unchanged_portion_omitted_
```

```
**********************************************************
   57704 Tue Nov 24 09:35:14 2015
new/usr/src/uts/common/vm/seg_map.c
6154 const-ify segment ops structures
**********************************************************
```

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

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

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

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

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

  61 #include <vm/seg_kmem.h>
```



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

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

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

  98 #define SEGMAP_BADOP(t) (t(*)())segmap_badop

 100 static const struct seg_ops segmap_ops = {
 100 static struct seg_ops segmap_ops = {
 101         .dup            = SEGMAP_BADOP(int),
 102         .unmap          = SEGMAP_BADOP(int),
 103         .free           = segmap_free,
 104         .fault          = segmap_fault,
 105         .faulta         = segmap_faulta,
 106         .setprot        = SEGMAP_BADOP(int),
 107         .checkprot      = segmap_checkprot,
 108         .kluster        = segmap_kluster,
 109         .swapout        = SEGMAP_BADOP(size_t),
 110         .sync           = SEGMAP_BADOP(int),
 111         .incore         = SEGMAP_BADOP(size_t),
 112         .lockop         = SEGMAP_BADOP(int),
 113         .getprot        = segmap_getprot,
 114         .getoffset      = segmap_getoffset,
 115         .gettype        = segmap_gettype,
 116         .getvp          = segmap_getvp,
 117         .advise         = SEGMAP_BADOP(int),
 118         .dump           = segmap_dump,
 119         .pagelock       = segmap_pagelock,
 120         .setpagesize    = SEGMAP_BADOP(int),
 121         .getmemid       = segmap_getmemid,
 122 };
______unchanged_portion_omitted_
```

```
**********************************************************
   83543 Tue Nov 24 09:35:14 2015
new/usr/src/uts/common/vm/seg_spt.c
6154 const-ify segment ops structures
**********************************************************
______unchanged_portion_omitted_

  86 #define SEGSPT_BADOP(t) (t(*)())segspt_badop

  88 const struct seg_ops segspt_ops = {
  88 struct seg_ops segspt_ops = {
  89         .dup            = SEGSPT_BADOP(int),
  90         .unmap          = segspt_unmap,
  91         .free           = segspt_free,
  92         .fault          = SEGSPT_BADOP(int),
  93         .faulta         = SEGSPT_BADOP(faultcode_t),
  94         .setprot        = SEGSPT_BADOP(int),
  95         .checkprot      = SEGSPT_BADOP(int),
  96         .kluster        = SEGSPT_BADOP(int),
  97         .swapout        = SEGSPT_BADOP(size_t),
  98         .sync           = SEGSPT_BADOP(int),
  99         .incore         = SEGSPT_BADOP(size_t),
 100         .lockop         = SEGSPT_BADOP(int),
 101         .getprot        = SEGSPT_BADOP(int),
 102         .getoffset      = SEGSPT_BADOP(u_offset_t),
 103         .gettype        = SEGSPT_BADOP(int),
 104         .getvp          = SEGSPT_BADOP(int),
 105         .advise         = SEGSPT_BADOP(int),
 106         .dump           = SEGSPT_BADOP(void),
 107         .pagelock       = SEGSPT_BADOP(int),
 108         .setpagesize    = SEGSPT_BADOP(int),
 109         .getmemid       = SEGSPT_BADOP(int),
 110         .getpolicy      = segspt_getpolicy,
 111         .capable        = SEGSPT_BADOP(int),
 112 };

 114 static int segspt_shmdup(struct seg *seg, struct seg *newseg);
 115 static int segspt_shmunmap(struct seg *seg, caddr_t raddr, size_t ssize);
 116 static void segspt_shmfree(struct seg *seg);
 117 static faultcode_t segspt_shmfault(struct hat *hat, struct seg *seg,
 118                 caddr_t addr, size_t len, enum fault_type type, enum seg_rw rw);
 119 static faultcode_t segspt_shmfaulta(struct seg *seg, caddr_t addr);
 120 static int segspt_shmsetprot(register struct seg *seg, register caddr_t addr,
 121                         register size_t len, register uint_t prot);
 122 static int segspt_shmcheckprot(struct seg *seg, caddr_t addr, size_t size,
 123                         uint_t prot);
 124 static int      segspt_shmkluster(struct seg *seg, caddr_t addr, ssize_t delta);
 125 static size_t   segspt_shmswapout(struct seg *seg);
 126 static size_t segspt_shmincore(struct seg *seg, caddr_t addr, size_t len,
 127                         register char *vec);
 128 static int segspt_shmsync(struct seg *seg, register caddr_t addr, size_t len,
 129                         int attr, uint_t flags);
 130 static int segspt_shmlockop(struct seg *seg, caddr_t addr, size_t len,
 131                         int attr, int op, ulong_t *lockmap, size_t pos);
 132 static int segspt_shmgetprot(struct seg *seg, caddr_t addr, size_t len,
 133                         uint_t *protv);
 134 static u_offset_t segspt_shmgetoffset(struct seg *seg, caddr_t addr);
 135 static int segspt_shmgettype(struct seg *seg, caddr_t addr);
 136 static int segspt_shmgetvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
 137 static int segspt_shmadvise(struct seg *seg, caddr_t addr, size_t len,
 138                         uint_t behav);
 139 static int segspt_shmpagelock(struct seg *, caddr_t, size_t,
 140                         struct page ***, enum lock_type, enum seg_rw);
 141 static int segspt_shmgetmemid(struct seg *, caddr_t, memid_t *);
 142 static lgrp_mem_policy_info_t *segspt_shmgetpolicy(struct seg *, caddr_t);
```



```
 144 const struct seg_ops segspt_shmops = {
 144 struct seg_ops segspt_shmops = {
 145         .dup            = segspt_shmdup,
 146         .unmap          = segspt_shmunmap,
 147         .free           = segspt_shmfree,
 148         .fault          = segspt_shmfault,
 149         .faulta         = segspt_shmfaulta,
 150         .setprot        = segspt_shmsetprot,
 151         .checkprot      = segspt_shmcheckprot,
 152         .kluster        = segspt_shmkluster,
 153         .swapout        = segspt_shmswapout,
 154         .sync           = segspt_shmsync,
 155         .incore         = segspt_shmincore,
 156         .lockop         = segspt_shmlockop,
 157         .getprot        = segspt_shmgetprot,
 158         .getoffset      = segspt_shmgetoffset,
 159         .gettype        = segspt_shmgettype,
 160         .getvp          = segspt_shmgetvp,
 161         .advise         = segspt_shmadvise,
 162         .pagelock       = segspt_shmpagelock,
 163         .getmemid       = segspt_shmgetmemid,
 164         .getpolicy      = segspt_shmgetpolicy,
 165 };
______unchanged_portion_omitted_
```

```
**********************************************************
  285812 Tue Nov 24 09:35:15 2015
new/usr/src/uts/common/vm/seg_vn.c
6154 const-ify segment ops structures
**********************************************************
```

```
   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) 1986, 2010, Oracle and/or its affiliates. All rights reserved.
  23  * Copyright 2015, Joyent, Inc. All rights reserved.
  24  * Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
  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 - shared or copy-on-write from a vnode/anonymous memory.
  42  */

  44 #include <sys/types.h>
  45 #include <sys/param.h>
  46 #include <sys/t_lock.h>
  47 #include <sys/errno.h>
  48 #include <sys/systm.h>
  49 #include <sys/mman.h>
  50 #include <sys/debug.h>
  51 #include <sys/cred.h>
  52 #include <sys/vmsystm.h>
  53 #include <sys/tuneable.h>
  54 #include <sys/bitmap.h>
  55 #include <sys/swap.h>
  56 #include <sys/kmem.h>
  57 #include <sys/sysmacros.h>
  58 #include <sys/vtrace.h>
  59 #include <sys/cmn_err.h>
  60 #include <sys/callb.h>
  61 #include <sys/vm.h>
```



```
  62 #include <sys/dumphdr.h>
  63 #include <sys/lgrp.h>

  65 #include <vm/hat.h>
  66 #include <vm/as.h>
  67 #include <vm/seg.h>
  68 #include <vm/seg_vn.h>
  69 #include <vm/pvn.h>
  70 #include <vm/anon.h>
  71 #include <vm/page.h>
  72 #include <vm/vpage.h>
  73 #include <sys/proc.h>
  74 #include <sys/task.h>
  75 #include <sys/project.h>
  76 #include <sys/zone.h>
  77 #include <sys/shm_impl.h>

  79 /*
  80  * segvn_fault needs a temporary page list array.  To avoid calling kmem all
  81  * the time, it creates a small (PVN_GETPAGE_NUM entry) array and uses it if
  82  * it can.  In the rare case when this page list is not large enough, it
  83  * goes and gets a large enough array from kmem.
  84  *
  85  * This small page list array covers either 8 pages or 64kB worth of pages -
  86  * whichever is smaller.
  87  */
  88 #define PVN_MAX_GETPAGE_SZ      0x10000
  89 #define PVN_MAX_GETPAGE_NUM     0x8

  91 #if PVN_MAX_GETPAGE_SZ > PVN_MAX_GETPAGE_NUM * PAGESIZE
  92 #define PVN_GETPAGE_SZ  ptob(PVN_MAX_GETPAGE_NUM)
  93 #define PVN_GETPAGE_NUM PVN_MAX_GETPAGE_NUM
  94 #else
  95 #define PVN_GETPAGE_SZ  PVN_MAX_GETPAGE_SZ
  96 #define PVN_GETPAGE_NUM btop(PVN_MAX_GETPAGE_SZ)
  97 #endif

  99 /*
 100  * Private seg op routines.
 101  */
 102 static int      segvn_dup(struct seg *seg, struct seg *newseg);
 103 static int      segvn_unmap(struct seg *seg, caddr_t addr, size_t len);
 104 static void     segvn_free(struct seg *seg);
 105 static faultcode_t segvn_fault(struct hat *hat, struct seg *seg,
 106                     caddr_t addr, size_t len, enum fault_type type,
 107                     enum seg_rw rw);
 108 static faultcode_t segvn_faulta(struct seg *seg, caddr_t addr);
 109 static int      segvn_setprot(struct seg *seg, caddr_t addr,
 110                     size_t len, uint_t prot);
 111 static int      segvn_checkprot(struct seg *seg, caddr_t addr,
 112                     size_t len, uint_t prot);
 113 static int      segvn_kluster(struct seg *seg, caddr_t addr, ssize_t delta);
 114 static size_t   segvn_swapout(struct seg *seg);
 115 static int      segvn_sync(struct seg *seg, caddr_t addr, size_t len,
 116                     int attr, uint_t flags);
 117 static size_t   segvn_incore(struct seg *seg, caddr_t addr, size_t len,
 118                     char *vec);
 119 static int      segvn_lockop(struct seg *seg, caddr_t addr, size_t len,
 120                     int attr, int op, ulong_t *lockmap, size_t pos);
 121 static int      segvn_getprot(struct seg *seg, caddr_t addr, size_t len,
 122                     uint_t *protv);
 123 static u_offset_t       segvn_getoffset(struct seg *seg, caddr_t addr);
 124 static int      segvn_gettype(struct seg *seg, caddr_t addr);
 125 static int      segvn_getvp(struct seg *seg, caddr_t addr,
 126                     struct vnode **vpp);
 127 static int      segvn_advise(struct seg *seg, caddr_t addr, size_t len,
```

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

139 const struct seg_ops segvn_ops = {
139 struct  seg_ops segvn_ops = {
140         .dup            = segvn_dup,
141         .unmap          = segvn_unmap,
142         .free           = segvn_free,
143         .fault          = segvn_fault,
144         .faulta         = segvn_faulta,
145         .setprot        = segvn_setprot,
146         .checkprot      = segvn_checkprot,
147         .kluster        = segvn_kluster,
148         .swapout        = segvn_swapout,
149         .sync           = segvn_sync,
150         .incore         = segvn_incore,
151         .lockop         = segvn_lockop,
152         .getprot        = segvn_getprot,
153         .getoffset      = segvn_getoffset,
154         .gettype        = segvn_gettype,
155         .getvp          = segvn_getvp,
156         .advise         = segvn_advise,
157         .dump           = segvn_dump,
158         .pagelock       = segvn_pagelock,
159         .setpagesize    = segvn_setpagesize,
160         .getmemid       = segvn_getmemid,
161         .getpolicy      = segvn_getpolicy,
162         .inherit        = segvn_inherit,
163 };
```

*unchanged_portion_omitted_*

```
**********************************************************
    9263 Tue Nov 24 09:35:15 2015
new/usr/src/uts/common/vm/seg_vn.h
6154 const-ify segment ops structures
**********************************************************
______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_
```

```
**********************************************************
   94448 Tue Nov 24 09:35:15 2015
new/usr/src/uts/common/vm/vm_as.c
6154 const-ify segment ops structures
**********************************************************
______unchanged_portion_omitted_
 410 #endif /* VERIFY_SEGLIST */

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

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

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

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

 434                 if (as->a_lastgaphl->s_base > newseg->s_base) {
 435                         hseg = as->a_lastgaphl;
 436                         lseg = AVL_PREV(&as->a_segtree, hseg);
 437                 } else {
 438                         lseg = as->a_lastgaphl;
 439                         hseg = AVL_NEXT(&as->a_segtree, lseg);
 440                 }

 442                 if (hseg && lseg && lseg->s_base < newseg->s_base &&
 443                     hseg->s_base > newseg->s_base) {
 444                         avl_insert_here(&as->a_segtree, newseg, lseg,
 445                             AVL_AFTER);
 446                         as->a_lastgaphl = NULL;
 447                         as->a_seglast = newseg;
 448                         return (0);
 449                 }
 450                 as->a_lastgaphl = NULL;
 451         }

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

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

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

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

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

 468                 /*
 469                  * If top of seg is below the requested address, then
```



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

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

 496 #ifdef VERIFY_SEGLIST
 497         as_verify(as);
 498 #endif
 499         return (0);
 500 }
______unchanged_portion_omitted_

 867 /*
 868  * Handle a ''fault'' at addr for size bytes.
 869  */
 870 faultcode_t
 871 as_fault(struct hat *hat, struct as *as, caddr_t addr, size_t size,
 872         enum fault_type type, enum seg_rw rw)
 873 {
 874         struct seg *seg;
 875         caddr_t raddr;                  /* rounded down addr */
 876         size_t rsize;                   /* rounded up size */
 877         size_t ssize;
 878         faultcode_t res = 0;
 879         caddr_t addrsav;
 880         struct seg *segsav;
 881         int as_lock_held;
 882         klwp_t *lwp = ttolwp(curthread);
 883         int is_xhat = 0;
 884         int holding_wpage = 0;
 885         extern struct seg_ops   segdev_ops;


 886         if (as->a_hat != hat) {
 887                 /* This must be an XHAT then */
 888                 is_xhat = 1;

 890                 if ((type != F_INVAL) || (as == &kas))
 891                         return (FC_NOSUPPORT);
 892         }

 894 retry:
 895         if (!is_xhat) {
```

```
896                 /*
897                  * Indicate that the lwp is not to be stopped while waiting
898                  * for a pagefault.  This is to avoid deadlock while debugging
899                  * a process via /proc over NFS (in particular).
900                  */
901                 if (lwp != NULL)
902                         lwp->lwp_nostop++;

904                 /*
905                  * same length must be used when we softlock and softunlock.
906                  * We don't support softunlocking lengths less than
907                  * the original length when there is largepage support.
908                  * See seg_dev.c for more comments.
909                  */
910                 switch (type) {

912                 case F_SOFTLOCK:
913                         CPU_STATS_ADD_K(vm, softlock, 1);
914                         break;

916                 case F_SOFTUNLOCK:
917                         break;

919                 case F_PROT:
920                         CPU_STATS_ADD_K(vm, prot_fault, 1);
921                         break;

923                 case F_INVAL:
924                         CPU_STATS_ENTER_K();
925                         CPU_STATS_ADDQ(CPU, vm, as_fault, 1);
926                         if (as == &kas)
927                                 CPU_STATS_ADDQ(CPU, vm, kernel_asflt, 1);
928                         CPU_STATS_EXIT_K();
929                         break;
930                 }
931         }

933         /* Kernel probe */
934         TNF_PROBE_3(address_fault, "vm pagefault", /* CSTYLED */,
935             tnf_opaque, address,        addr,
936             tnf_fault_type,     fault_type,     type,
937             tnf_seg_access,     access,         rw);

939         raddr = (caddr_t)((uintptr_t)addr & (uintptr_t)PAGEMASK);
940         rsize = (((size_t)(addr + size) + PAGEOFFSET) & PAGEMASK) -
941             (size_t)raddr;

943         /*
944          * XXX -- Don't grab the as lock for segkmap. We should grab it for
945          * correctness, but then we could be stuck holding this lock for
946          * a LONG time if the fault needs to be resolved on a slow
947          * filesystem, and then no-one will be able to exec new commands,
948          * as exec'ing requires the write lock on the as.
949          */
950         if (as == &kas && segkmap && segkmap->s_base <= raddr &&
951             raddr + size < segkmap->s_base + segkmap->s_size) {
952                 /*
953                  * if (as==&kas), this can't be XHAT: we've already returned
954                  * FC_NOSUPPORT.
955                  */
956                 seg = segkmap;
957                 as_lock_held = 0;
958         } else {
959                 AS_LOCK_ENTER(as, &as->a_lock, RW_READER);
960                 if (is_xhat && avl_numnodes(&as->a_wpage) != 0) {
961                         /*
```



```
962                          * Grab and hold the writers' lock on the as
963                          * if the fault is to a watched page.
964                          * This will keep CPUs from "peeking" at the
965                          * address range while we're temporarily boosting
966                          * the permissions for the XHAT device to
967                          * resolve the fault in the segment layer.
968                          *
969                          * We could check whether faulted address
970                          * is within a watched page and only then grab
971                          * the writer lock, but this is simpler.
972                          */
973                         AS_LOCK_EXIT(as, &as->a_lock);
974                         AS_LOCK_ENTER(as, &as->a_lock, RW_WRITER);
975                 }

977                 seg = as_segat(as, raddr);
978                 if (seg == NULL) {
979                         AS_LOCK_EXIT(as, &as->a_lock);
980                         if ((lwp != NULL) && (!is_xhat))
981                                 lwp->lwp_nostop--;
982                         return (FC_NOMAP);
983                 }

985                 as_lock_held = 1;
986         }

988         addrsav = raddr;
989         segsav = seg;

991         for (; rsize != 0; rsize -= ssize, raddr += ssize) {
992                 if (raddr >= seg->s_base + seg->s_size) {
993                         seg = AS_SEGNEXT(as, seg);
994                         if (seg == NULL || raddr != seg->s_base) {
995                                 res = FC_NOMAP;
996                                 break;
997                         }
998                 }
999                 if (raddr + rsize > seg->s_base + seg->s_size)
1000                        ssize = seg->s_base + seg->s_size - raddr;
1001                else
1002                        ssize = rsize;

1004                if (!is_xhat || (seg->s_ops != &segdev_ops)) {

1006                        if (is_xhat && avl_numnodes(&as->a_wpage) != 0 &&
1007                            pr_is_watchpage_as(raddr, rw, as)) {
1008                                /*
1009                                 * Handle watch pages.  If we're faulting on a
1010                                 * watched page from an X-hat, we have to
1011                                 * restore the original permissions while we
1012                                 * handle the fault.
1013                                 */
1014                                as_clearwatch(as);
1015                                holding_wpage = 1;
1016                        }

1018                        res = segop_fault(hat, seg, raddr, ssize, type, rw);

1020                        /* Restore watchpoints */
1021                        if (holding_wpage) {
1022                                as_setwatch(as);
1023                                holding_wpage = 0;
1024                        }

1026                        if (res != 0)
1027                                break;
```

```
1028                } else {
1029                        /* XHAT does not support seg_dev */
1030                        res = FC_NOSUPPORT;
1031                        break;
1032                }
1033        }

1035        /*
1036         * If we were SOFTLOCKing and encountered a failure,
1037         * we must SOFTUNLOCK the range we already did. (Maybe we
1038         * should just panic if we are SOFTLOCKing or even SOFTUNLOCKing
1039         * right here...)
1040         */
1041        if (res != 0 && type == F_SOFTLOCK) {
1042                for (seg = segsav; addrsav < raddr; addrsav += ssize) {
1043                        if (addrsav >= seg->s_base + seg->s_size)
1044                                seg = AS_SEGNEXT(as, seg);
1045                        ASSERT(seg != NULL);
1046                        /*
1047                         * Now call the fault routine again to perform the
1048                         * unlock using S_OTHER instead of the rw variable
1049                         * since we never got a chance to touch the pages.
1050                         */
1051                        if (raddr > seg->s_base + seg->s_size)
1052                                ssize = seg->s_base + seg->s_size - addrsav;
1053                        else
1054                                ssize = raddr - addrsav;
1055                        (void) segop_fault(hat, seg, addrsav, ssize,
1056                            F_SOFTUNLOCK, S_OTHER);
1057                }
1058        }
1059        if (as_lock_held)
1060                AS_LOCK_EXIT(as, &as->a_lock);
1061        if ((lwp != NULL) && (!is_xhat))
1062                lwp->lwp_nostop--;

1064        /*
1065         * If the lower levels returned EDEADLK for a fault,
1066         * It means that we should retry the fault.  Let's wait
1067         * a bit also to let the deadlock causing condition clear.
1068         * This is part of a gross hack to work around a design flaw
1069         * in the ufs/sds logging code and should go away when the
1070         * logging code is re-designed to fix the problem. See bug
1071         * 4125102 for details of the problem.
1072         */
1073        if (FC_ERRNO(res) == EDEADLK) {
1074                delay(deadlk_wait);
1075                res = 0;
1076                goto retry;
1077        }
1078        return (res);
1079 }
```

*unchanged_portion_omitted_*

```
2080 /*
2081  * Return the next range within [base, base + len) that is backed
2082  * with "real memory".  Skip holes and non-seg_vn segments.
2083  * We're lazy and only return one segment at a time.
2084  */
2085 int
2086 as_memory(struct as *as, caddr_t *basep, size_t *lenp)
2087 {
2088         extern const struct seg_ops segspt_shmops; /* needs a header file */
2091         extern struct seg_ops segspt_shmops;    /* needs a header file */
2089         struct seg *seg;
2090         caddr_t addr, eaddr;
```



```
2091         caddr_t segend;

2093         AS_LOCK_ENTER(as, &as->a_lock, RW_READER);

2095         addr = *basep;
2096         eaddr = addr + *lenp;

2098         seg = as_findseg(as, addr, 0);
2099         if (seg != NULL)
2100                 addr = MAX(seg->s_base, addr);

2102         for (;;) {
2103                 if (seg == NULL || addr >= eaddr || eaddr <= seg->s_base) {
2104                         AS_LOCK_EXIT(as, &as->a_lock);
2105                         return (EINVAL);
2106                 }

2108                 if (seg->s_ops == &segvn_ops) {
2109                         segend = seg->s_base + seg->s_size;
2110                         break;
2111                 }

2113                 /*
2114                  * We do ISM by looking into the private data
2115                  * to determine the real size of the segment.
2116                  */
2117                 if (seg->s_ops == &segspt_shmops) {
2118                         segend = seg->s_base + spt_realsize(seg);
2119                         if (addr < segend)
2120                                 break;
2121                 }

2123                 seg = AS_SEGNEXT(as, seg);

2125                 if (seg != NULL)
2126                         addr = seg->s_base;
2127         }

2129         *basep = addr;

2131         if (segend > eaddr)
2132                 *lenp = eaddr - addr;
2133         else
2134                 *lenp = segend - addr;

2136         AS_LOCK_EXIT(as, &as->a_lock);
2137         return (0);
2138 }

2140 /*
2141  * Swap the pages associated with the address space as out to
2142  * secondary storage, returning the number of bytes actually
2143  * swapped.
2144  *
2145  * The value returned is intended to correlate well with the process's
2146  * memory requirements.  Its usefulness for this purpose depends on
2147  * how well the segment-level routines do at returning accurate
2148  * information.
2149  */
2150 size_t
2151 as_swapout(struct as *as)
2152 {
2153         struct seg *seg;
2154         size_t swpcnt = 0;

2156         /*
```

```
2157          * Kernel-only processes have given up their address
2158          * spaces.  Of course, we shouldn't be attempting to
2159          * swap out such processes in the first place...
2160          */
2161         if (as == NULL)
2162                 return (0);

2164         AS_LOCK_ENTER(as, &as->a_lock, RW_READER);

2166         /* Prevent XHATs from attaching */
2167         mutex_enter(&as->a_contents);
2168         AS_SETBUSY(as);
2169         mutex_exit(&as->a_contents);


2172         /*
2173          * Free all mapping resources associated with the address
2174          * space.  The segment-level swapout routines capitalize
2175          * on this unmapping by scavanging pages that have become
2176          * unmapped here.
2177          */
2178         hat_swapout(as->a_hat);
2179         if (as->a_xhat != NULL)
2180                 xhat_swapout_all(as);

2182         mutex_enter(&as->a_contents);
2183         AS_CLRBUSY(as);
2184         mutex_exit(&as->a_contents);

2186         /*
2187          * Call the swapout routines of all segments in the address
2188          * space to do the actual work, accumulating the amount of
2189          * space reclaimed.
2190          */
2191         for (seg = AS_SEGFIRST(as); seg != NULL; seg = AS_SEGNEXT(as, seg)) {
2192                 const struct seg_ops *ov = seg->s_ops;
2195                 struct seg_ops *ov = seg->s_ops;

2194                 /*
2195                  * We have to check to see if the seg has
2196                  * an ops vector because the seg may have
2197                  * been in the middle of being set up when
2198                  * the process was picked for swapout.
2199                  */
2200                 if ((ov != NULL) && (ov->swapout != NULL))
2201                         swpcnt += segop_swapout(seg);
2202         }
2203         AS_LOCK_EXIT(as, &as->a_lock);
2204         return (swpcnt);
2205 }
```

*unchanged_portion_omitted_*

```
2609 /*
2610  * Pagelock pages from a range that spans more than 1 segment.  Obtain shadow
2611  * lists from each segment and copy them to one contiguous shadow list (plist)
2612  * as expected by the caller.  Save pointers to per segment shadow lists at
2613  * the tail of plist so that they can be used during as_pageunlock().
2614  */
2615 static int
2616 as_pagelock_segs(struct as *as, struct seg *seg, struct page ***ppp,
2617     caddr_t addr, size_t size, enum seg_rw rw)
2618 {
2619         caddr_t sv_addr = addr;
2620         size_t sv_size = size;
2621         struct seg *sv_seg = seg;
2622         ulong_t segcnt = 1;
```



```
2623         ulong_t cnt;
2624         size_t ssize;
2625         pgcnt_t npages = btop(size);
2626         page_t **plist;
2627         page_t **pl;
2628         int error;
2629         caddr_t eaddr;
2630         faultcode_t fault_err = 0;
2631         pgcnt_t pl_off;
2632         extern const struct seg_ops segspt_shmops;
2635         extern struct seg_ops segspt_shmops;

2634         ASSERT(AS_LOCK_HELD(as, &as->a_lock));
2635         ASSERT(seg != NULL);
2636         ASSERT(addr >= seg->s_base && addr < seg->s_base + seg->s_size);
2637         ASSERT(addr + size > seg->s_base + seg->s_size);
2638         ASSERT(IS_P2ALIGNED(size, PAGESIZE));
2639         ASSERT(IS_P2ALIGNED(addr, PAGESIZE));

2641         /*
2642          * Count the number of segments covered by the range we are about to
2643          * lock. The segment count is used to size the shadow list we return
2644          * back to the caller.
2645          */
2646         for (; size != 0; size -= ssize, addr += ssize) {
2647                 if (addr >= seg->s_base + seg->s_size) {

2649                         seg = AS_SEGNEXT(as, seg);
2650                         if (seg == NULL || addr != seg->s_base) {
2651                                 AS_LOCK_EXIT(as, &as->a_lock);
2652                                 return (EFAULT);
2653                         }
2654                         /*
2655                          * Do a quick check if subsequent segments
2656                          * will most likely support pagelock.
2657                          */
2658                         if (seg->s_ops == &segvn_ops) {
2659                                 vnode_t *vp;

2661                                 if (segop_getvp(seg, addr, &vp) != 0 ||
2662                                     vp != NULL) {
2663                                         AS_LOCK_EXIT(as, &as->a_lock);
2664                                         goto slow;
2665                                 }
2666                         } else if (seg->s_ops != &segspt_shmops) {
2667                                 AS_LOCK_EXIT(as, &as->a_lock);
2668                                 goto slow;
2669                         }
2670                         segcnt++;
2671                 }
2672                 if (addr + size > seg->s_base + seg->s_size) {
2673                         ssize = seg->s_base + seg->s_size - addr;
2674                 } else {
2675                         ssize = size;
2676                 }
2677         }
2678         ASSERT(segcnt > 1);

2680         plist = kmem_zalloc((npages + segcnt) * sizeof (page_t *), KM_SLEEP);

2682         addr = sv_addr;
2683         size = sv_size;
2684         seg = sv_seg;

2686         for (cnt = 0, pl_off = 0; size != 0; size -= ssize, addr += ssize) {
2687                 if (addr >= seg->s_base + seg->s_size) {
```

```
2688                        seg = AS_SEGNEXT(as, seg);
2689                        ASSERT(seg != NULL && addr == seg->s_base);
2690                        cnt++;
2691                        ASSERT(cnt < segcnt);
2692                }
2693                if (addr + size > seg->s_base + seg->s_size) {
2694                        ssize = seg->s_base + seg->s_size - addr;
2695                } else {
2696                        ssize = size;
2697                }
2698                pl = &plist[npages + cnt];
2699                error = segop_pagelock(seg, addr, ssize, (page_t ***)pl,
2700                    L_PAGELOCK, rw);
2701                if (error) {
2702                        break;
2703                }
2704                ASSERT(plist[npages + cnt] != NULL);
2705                ASSERT(pl_off + btop(ssize) <= npages);
2706                bcopy(plist[npages + cnt], &plist[pl_off],
2707                    btop(ssize) * sizeof (page_t *));
2708                pl_off += btop(ssize);
2709        }

2711        if (size == 0) {
2712                AS_LOCK_EXIT(as, &as->a_lock);
2713                ASSERT(cnt == segcnt - 1);
2714                *ppp = plist;
2715                return (0);
2716        }

2718        /*
2719         * one of pagelock calls failed. The error type is in error variable.
2720         * Unlock what we've locked so far and retry with F_SOFTLOCK if error
2721         * type is either EFAULT or ENOTSUP. Otherwise just return the error
2722         * back to the caller.
2723         */

2725        eaddr = addr;
2726        seg = sv_seg;

2728        for (cnt = 0, addr = sv_addr; addr < eaddr; addr += ssize) {
2729                if (addr >= seg->s_base + seg->s_size) {
2730                        seg = AS_SEGNEXT(as, seg);
2731                        ASSERT(seg != NULL && addr == seg->s_base);
2732                        cnt++;
2733                        ASSERT(cnt < segcnt);
2734                }
2735                if (eaddr > seg->s_base + seg->s_size) {
2736                        ssize = seg->s_base + seg->s_size - addr;
2737                } else {
2738                        ssize = eaddr - addr;
2739                }
2740                pl = &plist[npages + cnt];
2741                ASSERT(*pl != NULL);
2742                (void) segop_pagelock(seg, addr, ssize, (page_t ***)pl,
2743                    L_PAGEUNLOCK, rw);
2744        }

2746        AS_LOCK_EXIT(as, &as->a_lock);

2748        kmem_free(plist, (npages + segcnt) * sizeof (page_t *));

2750        if (error != ENOTSUP && error != EFAULT) {
2751                return (error);
2752        }
```



```
2754 slow:
2755        /*
2756         * If we are here because pagelock failed due to the need to cow fault
2757         * in the pages we want to lock F_SOFTLOCK will do this job and in
2758         * next as_pagelock() call for this address range pagelock will
2759         * hopefully succeed.
2760         */
2761        fault_err = as_fault(as->a_hat, as, sv_addr, sv_size, F_SOFTLOCK, rw);
2762        if (fault_err != 0) {
2763                return (fc_decode(fault_err));
2764        }
2765        *ppp = NULL;

2767        return (0);
2768 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   54646 Tue Nov 24 09:35:15 2015
new/usr/src/uts/common/vm/vm_seg.c
6154 const-ify segment ops structures
**********************************************************
______unchanged_portion_omitted_

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

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

 207 kthread_t *seg_pasync_thr;

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

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

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

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

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

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

 239 #define P_MATCH_PP(pcp, htag0, addr, len, pp)                           \
 240         ((pcp)->p_pp == (pp) &&                                         \
```



```
 241         (pcp)->p_htag0 == (htag0) &&                                    \
 242         (pcp)->p_addr == (addr) &&                                      \
 243         (pcp)->p_len >= (len))

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

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

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

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

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

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

317         /*
318          * If this bucket is still on list !lix async thread can't yet remove
319          * it since we hold here per bucket lock. In this case just return
320          * since async thread will eventually find and process this bucket.
321          */
322         if (hp->p_halink[!lix].p_lnext != NULL) {
323                 ASSERT(hp->p_halink[!lix].p_lprev != NULL);
324                 mutex_exit(&seg_pmem_mtx);
325                 return;
326         }
327         ASSERT(hp->p_halink[!lix].p_lprev == NULL);
328         /*
329          * This bucket is not on any active bucket list yet.
330          * Add the bucket to the tail of current active list.
331          */
332         hp->p_halink[lix].p_lnext = &seg_pahhead[lix];
333         hp->p_halink[lix].p_lprev = seg_pahhead[lix].p_lprev;
334         seg_pahhead[lix].p_lprev->p_lnext = &hp->p_halink[lix];
335         seg_pahhead[lix].p_lprev = &hp->p_halink[lix];
336         mutex_exit(&seg_pmem_mtx);
337 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   57936 Tue Nov 24 09:35:15 2015
new/usr/src/uts/common/vm/vm_usage.c
6154 const-ify segment ops structures
**********************************************************
______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 Tue Nov 24 09:35:16 2015
new/usr/src/uts/i86pc/io/rootnex.c
6154 const-ify segment ops structures
**********************************************************
______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_
```

```
**********************************************************
   16469 Tue Nov 24 09:35:16 2015
new/usr/src/uts/i86xpv/vm/seg_mf.c
6154 const-ify segment ops structures
**********************************************************
______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_

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

```
**********************************************************
   11607 Tue Nov 24 09:35:16 2015
new/usr/src/uts/sparc/v9/vm/seg_nf.c
6154 const-ify segment ops structures
**********************************************************
```

```
   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 void     segnf_badop(void);
  70 static int      segnf_nop(void);
  71 static int      segnf_getprot(struct seg *seg, caddr_t addr,
  72                     size_t len, uint_t *protv);
  73 static u_offset_t segnf_getoffset(struct seg *seg, caddr_t addr);
  74 static int      segnf_gettype(struct seg *seg, caddr_t addr);
  75 static int      segnf_getvp(struct seg *seg, caddr_t addr, struct vnode **vpp);
  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         .kluster        = (int (*)())segnf_badop,
  90         .sync           = (int (*)(struct seg *, caddr_t, size_t, int, uint_t))
  91                 segnf_nop,
  92         .incore         = (size_t (*)(struct seg *, caddr_t, size_t, char *))
  93                 segnf_nop,
  94         .lockop         = (int (*)(struct seg *, caddr_t, size_t, int, int,
  95             ulong_t *, size_t))segnf_nop,
  96         .getprot        = segnf_getprot,
  97         .getoffset      = segnf_getoffset,
  98         .gettype        = segnf_gettype,
  99         .getvp          = segnf_getvp,
 100         .advise         = (int (*)(struct seg *, caddr_t, size_t, uint_t))
 101                 segnf_nop,
 102         .pagelock       = segnf_pagelock,
 103 };
_____unchanged_portion_omitted_
```

```
**********************************************************
   23606 Tue Nov 24 09:35:16 2015
new/usr/src/uts/sun4/io/rootnex.c
6154 const-ify segment ops structures
**********************************************************
______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_
```