```
**********************************************************
  110792 Mon Jan 14 13:16:35 2019
new/usr/src/uts/common/fs/vnode.c
10084 fop_open() doesn't need to check for a NULL vnode
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 * Copyright 2017, Joyent, Inc.
 * Copyright 2016 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2011, 2017 by Delphix. All rights reserved.
 */

/*      Copyright (c) 1983, 1984, 1985, 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.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/t_lock.h>
#include <sys/errno.h>
#include <sys/cred.h>
#include <sys/user.h>
#include <sys/uio.h>
#include <sys/file.h>
#include <sys/pathname.h>
#include <sys/vfs.h>
#include <sys/vfs_opreg.h>
#include <sys/vnode.h>
#include <sys/rwstlock.h>
#include <sys/fem.h>
#include <sys/stat.h>
#include <sys/mode.h>
#include <sys/conf.h>
#include <sys/sysmacros.h>
#include <sys/cmn_err.h>
```

```
#include <sys/systm.h>
#include <sys/kmem.h>
#include <sys/debug.h>
#include <c2/audit.h>
#include <sys/acl.h>
#include <sys/nbmlock.h>
#include <sys/fcntl.h>
#include <fs/fs_subr.h>
#include <sys/taskq.h>
#include <fs/fs_reparse.h>
#include <sys/time.h>
#include <sys/sdt.h>

/* Determine if this vnode is a file that is read-only */
#define ISROFILE(vp)    \
        ((vp)->v_type != VCHR && (vp)->v_type != VBLK && \
            (vp)->v_type != VFIFO && vn_is_readonly(vp))

/* Tunable via /etc/system; used only by admin/install */
int nfs_global_client_only;

/*
 * Array of vopstats_t for per-FS-type vopstats.  This array has the same
 * number of entries as and parallel to the vfssw table.  (Arguably, it could
 * be part of the vfssw table.)  Once it's initialized, it's accessed using
 * the same fstype index that is used to index into the vfssw table.
 */
vopstats_t **vopstats_fstype;

/* vopstats initialization template used for fast initialization via bcopy() */
static vopstats_t *vs_templatep;

/* Kmem cache handle for vsk_anchor_t allocations */
kmem_cache_t *vsk_anchor_cache;

/* file events cleanup routine */
extern void free_fopdata(vnode_t *);

/*
 * Root of AVL tree for the kstats associated with vopstats.  Lock protects
 * updates to vsktat_tree.
 */
avl_tree_t      vskstat_tree;
kmutex_t        vskstat_tree_lock;

/* Global variable which enables/disables the vopstats collection */
int vopstats_enabled = 1;

/* Global used for empty/invalid v_path */
char *vn_vpath_empty = "";

/*
 * forward declarations for internal vnode specific data (vsd)
 */
static void *vsd_realloc(void *, size_t, size_t);

/*
 * forward declarations for reparse point functions
 */
static int fs_reparse_mark(char *target, vattr_t *vap, xvattr_t *xvattr);

/*
 * VSD -- VNODE SPECIFIC DATA
 * The v_data pointer is typically used by a file system to store a
 * pointer to the file system's private node (e.g. ufs inode, nfs rnode).
 * However, there are times when additional project private data needs
 * to be stored separately from the data (node) pointed to by v_data.
 * This additional data could be stored by the file system itself or
 * by a completely different kernel entity.  VSD provides a way for
 * callers to obtain a key and store a pointer to private data associated
 * with a vnode.
 *
 * Callers are responsible for protecting the vsd by holding v_vsd_lock
 * for calls to vsd_set() and vsd_get().
 */

/*
 * vsd_lock protects:
 *   vsd_nkeys - creation and deletion of vsd keys
 *   vsd_list - insertion and deletion of vsd_node in the vsd_list
 *   vsd_destructor - adding and removing destructors to the list
 */
static kmutex_t         vsd_lock;
static uint_t           vsd_nkeys;       /* size of destructor array */
/* list of vsd_node's */
static list_t *vsd_list = NULL;
/* per-key destructor funcs */
static void             (**vsd_destructor)(void *);

/*
 * The following is the common set of actions needed to update the
 * vopstats structure from a vnode op.  Both VOPSTATS_UPDATE() and
 * VOPSTATS_UPDATE_IO() do almost the same thing, except for the
 * recording of the bytes transferred.  Since the code is similar
 * but small, it is nearly a duplicate.  Consequently any changes
 * to one may need to be reflected in the other.
 * Rundown of the variables:
 * vp - Pointer to the vnode
 * counter - Partial name structure member to update in vopstats for counts
 * bytecounter - Partial name structure member to update in vopstats for bytes
 * bytesval - Value to update in vopstats for bytes
 * fstype - Index into vsanchor_fstype[], same as index into vfssw[]
 * vsp - Pointer to vopstats structure (either in vfs or vsanchor_fstype[i])
 */

#define VOPSTATS_UPDATE(vp, counter) {                                  \
        vfs_t *vfsp = (vp)->v_vfsp;                                     \
        if (vfsp && vfsp->vfs_implp &&                                  \
            (vfsp->vfs_flag & VFS_STATS) && (vp)->v_type != VBAD) {     \
                vopstats_t *vsp = &vfsp->vfs_vopstats;                  \
                uint64_t *stataddr = &(vsp->n##counter.value.ui64);     \
                extern void __dtrace_probe___fsinfo_##counter(vnode_t *, \
                    size_t, uint64_t *);                                \
                __dtrace_probe___fsinfo_##counter(vp, 0, stataddr);     \
                (*stataddr)++;                                          \
                if ((vsp = vfsp->vfs_fstypevsp) != NULL) {              \
                        vsp->n##counter.value.ui64++;                   \
                }                                                       \
        }                                                               \
}
```

______***unchanged_portion_omitted_***

```
/* VOP_XXX() macros call the corresponding fop_xxx() function */

int
fop_open(
        vnode_t **vpp,
        int mode,
        cred_t *cr,
        caller_context_t *ct)
{
        int ret;
```

```
        vnode_t *vp = *vpp;

        VN_HOLD(vp);
        /*
         * Adding to the vnode counts before calling open
         * avoids the need for a mutex. It circumvents a race
         * condition where a query made on the vnode counts results in a
         * false negative. The inquirer goes away believing the file is
         * not open when there is an open on the file already under way.
         *
         * The counts are meant to prevent NFS from granting a delegation
         * when it would be dangerous to do so.
         *
         * The vnode counts are only kept on regular files
         */
        if ((*vpp)->v_type == VREG) {
                if (mode & FREAD)
                        atomic_inc_32(&(*vpp)->v_rdcnt);
                if (mode & FWRITE)
                        atomic_inc_32(&(*vpp)->v_wrcnt);
        }

        VOPXID_MAP_CR(vp, cr);

        ret = (*(*(vpp))->v_op->vop_open)(vpp, mode, cr, ct);

        if (ret) {
                /*
                 * Use the saved vp just in case the vnode ptr got trashed
                 * by the error.
                 */
                VOPSTATS_UPDATE(vp, open);
                if ((vp->v_type == VREG) && (mode & FREAD))
                        atomic_dec_32(&vp->v_rdcnt);
                if ((vp->v_type == VREG) && (mode & FWRITE))
                        atomic_dec_32(&vp->v_wrcnt);
        } else {
                /*
                 * Some filesystems will return a different vnode,
                 * but the same path was still used to open it.
                 * So if we do change the vnode and need to
                 * copy over the path, do so here, rather than special
                 * casing each filesystem. Adjust the vnode counts to
                 * reflect the vnode switch.
                 */
                VOPSTATS_UPDATE(*vpp, open);
                if (*vpp != vp) {
                if (*vpp != vp && *vpp != NULL) {
                        vn_copypath(vp, *vpp);
                        if (((*vpp)->v_type == VREG) && (mode & FREAD))
                                atomic_inc_32(&(*vpp)->v_rdcnt);
                        if ((vp->v_type == VREG) && (mode & FREAD))
                                atomic_dec_32(&vp->v_rdcnt);
                        if (((*vpp)->v_type == VREG) && (mode & FWRITE))
                                atomic_inc_32(&(*vpp)->v_wrcnt);
                        if ((vp->v_type == VREG) && (mode & FWRITE))
                                atomic_dec_32(&vp->v_wrcnt);
                }
        }
        VN_RELE(vp);
        return (ret);
}
```

______***unchanged_portion_omitted_***