****
26638 Mon Dec 1 22:44:39 2014
new/usr/src/uts/common/fs/dev/sdev_profile.c
5360 Race condition in devfs upgrades reader to writer incidentally and causes p
****

```
/*
 * 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 2014 Coraid, Inc.
 * Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * This file implements /dev filesystem operations for non-global
 * instances. Three major entry points:
 * devname_profile_update()
 *   Update matching rules determining which names to export
 * prof_readdir()
 *   Return the list of exported names
 * prof_lookup()
 *   Implements lookup
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/sysmacros.h>
#include <sys/vnode.h>
#include <sys/uio.h>
#include <sys/dirent.h>
#include <sys/pathname.h>
#include <sys/fs/dv_node.h>
#include <sys/fs/sdev_impl.h>
#include <sys/sunndi.h>
#include <sys/modctl.h>

enum {
        PROFILE_TYPE_INCLUDE,
        PROFILE_TYPE_EXCLUDE,
        PROFILE_TYPE_MAP,
        PROFILE_TYPE_SYMLINK
};
```

*unchanged_portion_omitted_*

```
/*
 * Return True if directory cache is out of date and should be updated.
 */
static boolean_t
```



```
prof_dev_needupdate(sdev_node_t *ddv)
{
        sdev_node_t *gdir = ddv->sdev_origin;

        /*
         * Caller can have either reader or writer lock
         */
        ASSERT(RW_LOCK_HELD(&ddv->sdev_contents));

        /*
         * We need to rebuild the directory content if
         * - ddv is not in a SDEV_ZOMBIE state
         * - SDEV_BUILD is set OR
         * - The device tree generation number has changed OR
         * - The corresponding /dev namespace has been updated
         */
        return ((ddv->sdev_state != SDEV_ZOMBIE) &&
            (((ddv->sdev_flags & SDEV_BUILD) != 0) ||
            (ddv->sdev_devtree_gen != devtree_gen) ||
            ((gdir != NULL) &&
            (ddv->sdev_ldir_gen != gdir->sdev_gdir_gen))));
}

/*
 * Build directory vnodes based on the profile and the global
 * dev instance.
 */
void
prof_filldir(sdev_node_t *ddv)
prof_filldir(struct sdev_node *ddv)
{
        sdev_node_t *gdir;
        int firsttime = 1;
        struct sdev_node *gdir = ddv->sdev_origin;

        ASSERT(RW_READ_HELD(&ddv->sdev_contents));

        if (!prof_dev_needupdate(ddv)) {
                ASSERT(RW_READ_HELD(&ddv->sdev_contents));
                return;
        }
        /*
         * Upgrade to writer lock
         * We need to rebuild the directory content if
         * - SDEV_BUILD is set
         * - The device tree generation number has changed
         * - The corresponding /dev namespace has been updated
         */
        if (rw_tryupgrade(&ddv->sdev_contents) == 0) {
                /*
                 * We need to drop the read lock and re-acquire it as a
                 * write lock. While we do this the condition may change so we
                 * need to re-check condition.
                 * NOTE: if device becomes a zombie, prof_dev_needupdate returns
                 * false, so nothing we will just return in this case.
                 */
check_build:
        if ((ddv->sdev_flags & SDEV_BUILD) == 0 &&
            ddv->sdev_devtree_gen == devtree_gen &&
            (gdir == NULL || ddv->sdev_ldir_gen
            == gdir->sdev_gdir_gen))
                return;         /* already up to date */

        /* We may have become a zombie (across a try) */
        if (ddv->sdev_state == SDEV_ZOMBIE)
                return;
```

```
686         if (firsttime && rw_tryupgrade(&ddv->sdev_contents) == 0) {
711                 rw_exit(&ddv->sdev_contents);
688                 firsttime = 0;
712                 rw_enter(&ddv->sdev_contents, RW_WRITER);
713                 if (!prof_dev_needupdate(ddv)) {
714                         /* Downgrade back to the read lock before returning */
715                         rw_downgrade(&ddv->sdev_contents);
716                         return;
690                 goto check_build;
717                 }
718         }
719         /* At this point we should have a write lock */
720         ASSERT(RW_WRITE_HELD(&ddv->sdev_contents));

722         sdcmn_err10(("devtree_gen (%s): %ld -> %ld\n",
723             ddv->sdev_path, ddv->sdev_devtree_gen, devtree_gen));

725         gdir = ddv->sdev_origin;

727         if (gdir != NULL)
694         if (gdir)
728                 sdcmn_err10(("sdev_dir_gen (%s): %ld -> %ld\n",
729                     ddv->sdev_path, ddv->sdev_ldir_gen,
730                     gdir->sdev_gdir_gen));

732         /* update flags and generation number so next filldir is quick */
733         if ((ddv->sdev_flags & SDEV_BUILD) == SDEV_BUILD) {
734                 ddv->sdev_flags &= ~SDEV_BUILD;
735         }
736         ddv->sdev_devtree_gen = devtree_gen;
737         if (gdir != NULL)
702         if (gdir)
738                 ddv->sdev_ldir_gen = gdir->sdev_gdir_gen;

740         prof_make_symlinks(ddv);
741         prof_make_maps(ddv);
742         prof_make_names(ddv);
743         rw_downgrade(&ddv->sdev_contents);
744 }
```

_____***unchanged_portion_omitted_***