```
**********************************************************
  238265 Tue Jan 15 10:29:45 2019
new/usr/src/uts/common/os/sunmdi.c
10094 i_mdi_client_free() doesn’t need to check for a NULL cdip
**********************************************************
/*
 * 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) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2014 Nexenta Systems Inc. All rights reserved.
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * Multipath driver interface (MDI) implementation; see mdi_impldefs.h for a
 * more detailed discussion of the overall mpxio architecture.
 *
 * Default locking order:
 *
 * _NOTE(LOCK_ORDER(mdi_mutex, mdi_vhci:vh_phci_mutex);
 * _NOTE(LOCK_ORDER(mdi_mutex, mdi_vhci:vh_client_mutex);
 * _NOTE(LOCK_ORDER(mdi_vhci:vh_phci_mutex, mdi_phci::ph_mutex);
 * _NOTE(LOCK_ORDER(mdi_vhci:vh_client_mutex, mdi_client::ct_mutex);
 * _NOTE(LOCK_ORDER(mdi_phci::ph_mutex mdi_pathinfo::pi_mutex))
 * _NOTE(LOCK_ORDER(mdi_phci::ph_mutex mdi_client::ct_mutex))
 * _NOTE(LOCK_ORDER(mdi_client::ct_mutex mdi_pathinfo::pi_mutex))
 */

#include <sys/note.h>
#include <sys/types.h>
#include <sys/varargs.h>
#include <sys/param.h>
#include <sys/errno.h>
#include <sys/uio.h>
#include <sys/buf.h>
#include <sys/modctl.h>
#include <sys/open.h>
#include <sys/kmem.h>
#include <sys/poll.h>
#include <sys/conf.h>
#include <sys/bootconf.h>
#include <sys/cmn_err.h>
#include <sys/stat.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/ddipropdefs.h>
#include <sys/sunndi.h>
#include <sys/ndi_impldefs.h>
```

```
#include <sys/promif.h>
#include <sys/sunmdi.h>
#include <sys/mdi_impldefs.h>
#include <sys/taskq.h>
#include <sys/epm.h>
#include <sys/sunpm.h>
#include <sys/modhash.h>
#include <sys/disp.h>
#include <sys/autoconf.h>
#include <sys/sysmacros.h>

#ifdef  DEBUG
#include <sys/debug.h>
int     mdi_debug = 1;
int     mdi_debug_logonly = 0;
#define MDI_DEBUG(dbglevel, pargs) if (mdi_debug >= (dbglevel)) i_mdi_log pargs
#define MDI_WARN        CE_WARN, __func__
#define MDI_NOTE        CE_NOTE, __func__
#define MDI_CONT        CE_CONT, __func__
static void i_mdi_log(int, const char *, dev_info_t *, const char *, ...);
#else   /* !DEBUG */
#define MDI_DEBUG(dbglevel, pargs)
#endif  /* DEBUG */
int     mdi_debug_consoleonly = 0;
int     mdi_delay = 3;

extern pri_t    minclsyspri;
extern int      modrootloaded;

/*
 * Global mutex:
 * Protects vHCI list and structure members.
 */
kmutex_t        mdi_mutex;

/*
 * Registered vHCI class driver lists
 */
int             mdi_vhci_count;
mdi_vhci_t      *mdi_vhci_head;
mdi_vhci_t      *mdi_vhci_tail;

/*
 * Client Hash Table size
 */
static int      mdi_client_table_size = CLIENT_HASH_TABLE_SIZE;

/*
 * taskq interface definitions
 */
#define MDI_TASKQ_N_THREADS     8
#define MDI_TASKQ_PRI           minclsyspri
#define MDI_TASKQ_MINALLOC      (4*mdi_taskq_n_threads)
#define MDI_TASKQ_MAXALLOC      (500*mdi_taskq_n_threads)

taskq_t                         *mdi_taskq;
static uint_t                   mdi_taskq_n_threads = MDI_TASKQ_N_THREADS;

#define TICKS_PER_SECOND        (drv_usectohz(1000000))

/*
 * The data should be "quiet" for this interval (in seconds) before the
 * vhci cached data is flushed to the disk.
 */
static int mdi_vhcache_flush_delay = 10;
```

```
/* number of seconds the vhcache flush daemon will sleep idle before exiting */
static int mdi_vhcache_flush_daemon_idle_time = 60;

/*
 * MDI falls back to discovery of all paths when a bus_config_one fails.
 * The following parameters can be used to tune this operation.
 *
 * mdi_path_discovery_boot
 *      Number of times path discovery will be attempted during early boot.
 *      Probably there is no reason to ever set this value to greater than one.
 *
 * mdi_path_discovery_postboot
 *      Number of times path discovery will be attempted after early boot.
 *      Set it to a minimum of two to allow for discovery of iscsi paths which
 *      may happen very late during booting.
 *
 * mdi_path_discovery_interval
 *      Minimum number of seconds MDI will wait between successive discovery
 *      of all paths. Set it to -1 to disable discovery of all paths.
 */
static int mdi_path_discovery_boot = 1;
static int mdi_path_discovery_postboot = 2;
static int mdi_path_discovery_interval = 10;

/*
 * number of seconds the asynchronous configuration thread will sleep idle
 * before exiting.
 */
static int mdi_async_config_idle_time = 600;

static int mdi_bus_config_cache_hash_size = 256;

/* turns off multithreaded configuration for certain operations */
static int mdi_mtc_off = 0;

/*
 * The "path" to a pathinfo node is identical to the /devices path to a
 * devinfo node had the device been enumerated under a pHCI instead of
 * a vHCI.  This pathinfo "path" is associated with a 'path_instance'.
 * This association persists across create/delete of the pathinfo nodes,
 * but not across reboot.
 */
static uint_t           mdi_pathmap_instance = 1;       /* 0 -> any path */
static int              mdi_pathmap_hash_size = 256;
static kmutex_t         mdi_pathmap_mutex;
static mod_hash_t       *mdi_pathmap_bypath;            /* "path"->instance */
static mod_hash_t       *mdi_pathmap_byinstance;        /* instance->"path" */
static mod_hash_t       *mdi_pathmap_sbyinstance;       /* inst->shortpath */

/*
 * MDI component property name/value string definitions
 */
const char              *mdi_component_prop = "mpxio-component";
const char              *mdi_component_prop_vhci = "vhci";
const char              *mdi_component_prop_phci = "phci";
const char              *mdi_component_prop_client = "client";

/*
 * MDI client global unique identifier property name
 */
const char              *mdi_client_guid_prop = "client-guid";

/*
 * MDI client load balancing property name/value string definitions
 */
const char              *mdi_load_balance = "load-balance";
```

```
const char              *mdi_load_balance_none = "none";
const char              *mdi_load_balance_rr = "round-robin";
const char              *mdi_load_balance_lba = "logical-block";

/*
 * Obsolete vHCI class definition; to be removed after Leadville update
 */
const char *mdi_vhci_class_scsi = MDI_HCI_CLASS_SCSI;

static char vhci_greeting[] =
        "\tThere already exists one vHCI driver for class %s\n"
        "\tOnly one vHCI driver for each class is allowed\n";

/*
 * Static function prototypes
 */
static int              i_mdi_phci_offline(dev_info_t *, uint_t);
static int              i_mdi_client_offline(dev_info_t *, uint_t);
static int              i_mdi_phci_pre_detach(dev_info_t *, ddi_detach_cmd_t);
static void             i_mdi_phci_post_detach(dev_info_t *,
                            ddi_detach_cmd_t, int);
static int              i_mdi_client_pre_detach(dev_info_t *,
                            ddi_detach_cmd_t);
static void             i_mdi_client_post_detach(dev_info_t *,
                            ddi_detach_cmd_t, int);
static void             i_mdi_pm_hold_pip(mdi_pathinfo_t *);
static void             i_mdi_pm_rele_pip(mdi_pathinfo_t *);
static int              i_mdi_lba_lb(mdi_client_t *ct,
                            mdi_pathinfo_t **ret_pip, struct buf *buf);
static void             i_mdi_pm_hold_client(mdi_client_t *, int);
static void             i_mdi_pm_rele_client(mdi_client_t *, int);
static void             i_mdi_pm_reset_client(mdi_client_t *);
static int              i_mdi_power_all_phci(mdi_client_t *);
static void             i_mdi_log_sysevent(dev_info_t *, char *, char *);


/*
 * Internal mdi_pathinfo node functions
 */
static void             i_mdi_pi_kstat_destroy(mdi_pathinfo_t *);

static mdi_vhci_t       *i_mdi_vhci_class2vhci(char *);
static mdi_vhci_t       *i_devi_get_vhci(dev_info_t *);
static mdi_phci_t       *i_devi_get_phci(dev_info_t *);
static void             i_mdi_phci_lock(mdi_phci_t *, mdi_pathinfo_t *);
static void             i_mdi_phci_unlock(mdi_phci_t *);
static mdi_pathinfo_t   *i_mdi_pi_alloc(mdi_phci_t *, char *, mdi_client_t *);
static void             i_mdi_phci_add_path(mdi_phci_t *, mdi_pathinfo_t *);
static void             i_mdi_client_add_path(mdi_client_t *, mdi_pathinfo_t *);
static void             i_mdi_pi_free(mdi_phci_t *ph, mdi_pathinfo_t *,
                            mdi_client_t *);
static void             i_mdi_phci_remove_path(mdi_phci_t *, mdi_pathinfo_t *);
static void             i_mdi_client_remove_path(mdi_client_t *,
                            mdi_pathinfo_t *);

static int              i_mdi_pi_state_change(mdi_pathinfo_t *,
                            mdi_pathinfo_state_t, int);
static int              i_mdi_pi_offline(mdi_pathinfo_t *, int);
static dev_info_t       *i_mdi_devinfo_create(mdi_vhci_t *, char *, char *,
                            char **, int);
static dev_info_t       *i_mdi_devinfo_find(mdi_vhci_t *, char *, char *);
static int              i_mdi_devinfo_remove(dev_info_t *, dev_info_t *, int);
static int              i_mdi_is_child_present(dev_info_t *, dev_info_t *);
static mdi_client_t     *i_mdi_client_alloc(mdi_vhci_t *, char *, char *);
static void             i_mdi_client_enlist_table(mdi_vhci_t *, mdi_client_t *);
static void             i_mdi_client_delist_table(mdi_vhci_t *, mdi_client_t *);
static mdi_client_t     *i_mdi_client_find(mdi_vhci_t *, char *, char *);
static void             i_mdi_client_update_state(mdi_client_t *);
static int              i_mdi_client_compute_state(mdi_client_t *,
                            mdi_phci_t *);
static void             i_mdi_client_lock(mdi_client_t *, mdi_pathinfo_t *);
static void             i_mdi_client_unlock(mdi_client_t *);
static int              i_mdi_client_free(mdi_vhci_t *, mdi_client_t *);
static mdi_client_t     *i_devi_get_client(dev_info_t *);
/*
 * NOTE: this will be removed once the NWS files are changed to use the new
 * mdi_{enable,disable}_path interfaces
 */
static int              i_mdi_pi_enable_disable(dev_info_t *, dev_info_t *,
                                int, int);
static mdi_pathinfo_t   *i_mdi_enable_disable_path(mdi_pathinfo_t *pip,
                                mdi_vhci_t *vh, int flags, int op);
/*
 * Failover related function prototypes
 */
static int              i_mdi_failover(void *);

/*
 * misc internal functions
 */
static int              i_mdi_get_hash_key(char *);
static int              i_map_nvlist_error_to_mdi(int);
static void             i_mdi_report_path_state(mdi_client_t *,
                            mdi_pathinfo_t *);

static void             setup_vhci_cache(mdi_vhci_t *);
static int              destroy_vhci_cache(mdi_vhci_t *);
static int              stop_vhcache_async_threads(mdi_vhci_config_t *);
static boolean_t        stop_vhcache_flush_thread(void *, int);
static void             free_string_array(char **, int);
static void             free_vhcache_phci(mdi_vhcache_phci_t *);
static void             free_vhcache_pathinfo(mdi_vhcache_pathinfo_t *);
static void             free_vhcache_client(mdi_vhcache_client_t *);
static int              mainnvl_to_vhcache(mdi_vhci_cache_t *, nvlist_t *);
static nvlist_t         *vhcache_to_mainnvl(mdi_vhci_cache_t *);
static void             vhcache_phci_add(mdi_vhci_config_t *, mdi_phci_t *);
static void             vhcache_phci_remove(mdi_vhci_config_t *, mdi_phci_t *);
static void             vhcache_pi_add(mdi_vhci_config_t *,
                            struct mdi_pathinfo *);
static void             vhcache_pi_remove(mdi_vhci_config_t *,
                            struct mdi_pathinfo *);
static void             free_phclient_path_list(mdi_phys_path_t *);
static void             sort_vhcache_paths(mdi_vhcache_client_t *);
static int              flush_vhcache(mdi_vhci_config_t *, int);
static void             vhcache_dirty(mdi_vhci_config_t *);
static void             free_async_client_config(mdi_async_client_config_t *);
static void             single_threaded_vhconfig_enter(mdi_vhci_config_t *);
static void             single_threaded_vhconfig_exit(mdi_vhci_config_t *);
static nvlist_t         *read_on_disk_vhci_cache(char *);
extern int              fread_nvlist(char *, nvlist_t **);
extern int              fwrite_nvlist(char *, nvlist_t *);

/* called once when first vhci registers with mdi */
static void
i_mdi_init()
{
        static int initialized = 0;

        if (initialized)
                return;
        initialized = 1;
```

```
        mutex_init(&mdi_mutex, NULL, MUTEX_DEFAULT, NULL);

        /* Create our taskq resources */
        mdi_taskq = taskq_create("mdi_taskq", mdi_taskq_n_threads,
            MDI_TASKQ_PRI, MDI_TASKQ_MINALLOC, MDI_TASKQ_MAXALLOC,
            TASKQ_PREPOPULATE | TASKQ_CPR_SAFE);
        ASSERT(mdi_taskq != NULL);      /* taskq_create never fails */

        /* Allocate ['path_instance' <-> "path"] maps */
        mutex_init(&mdi_pathmap_mutex, NULL, MUTEX_DRIVER, NULL);
        mdi_pathmap_bypath = mod_hash_create_strhash(
            "mdi_pathmap_bypath", mdi_pathmap_hash_size,
            mod_hash_null_valdtor);
        mdi_pathmap_byinstance = mod_hash_create_idhash(
            "mdi_pathmap_byinstance", mdi_pathmap_hash_size,
            mod_hash_null_valdtor);
        mdi_pathmap_sbyinstance = mod_hash_create_idhash(
            "mdi_pathmap_sbyinstance", mdi_pathmap_hash_size,
            mod_hash_null_valdtor);
}
```

______***unchanged_portion_omitted_***

```
/*
 * i_mdi_client_free():
 *              Free a client component
 */
static int
i_mdi_client_free(mdi_vhci_t *vh, mdi_client_t *ct)
{
        int             rv = MDI_SUCCESS;
        int             flags = ct->ct_flags;
        dev_info_t      *cdip;
        dev_info_t      *vdip;

        ASSERT(MDI_VHCI_CLIENT_LOCKED(vh));

        vdip = vh->vh_dip;
        cdip = ct->ct_dip;

        (void) ndi_prop_remove(DDI_DEV_T_NONE, cdip, MDI_CLIENT_GUID_PROP);
        DEVI(cdip)->devi_mdi_component &= ~MDI_COMPONENT_CLIENT;
        DEVI(cdip)->devi_mdi_client = NULL;

        /*
         * Clear out back ref. to dev_info_t node
         */
        ct->ct_dip = NULL;

        /*
         * Remove this client from our hash queue
         */
        i_mdi_client_delist_table(vh, ct);

        /*
         * Uninitialize and free the component
         */
        kmem_free(ct->ct_drvname, strlen(ct->ct_drvname) + 1);
        kmem_free(ct->ct_guid, strlen(ct->ct_guid) + 1);
        kmem_free(ct->ct_lb_args, sizeof (client_lb_args_t));
        cv_destroy(&ct->ct_failover_cv);
        cv_destroy(&ct->ct_unstable_cv);
        cv_destroy(&ct->ct_powerchange_cv);
        mutex_destroy(&ct->ct_mutex);
        kmem_free(ct, sizeof (*ct));
        if (cdip != NULL) {
        MDI_VHCI_CLIENT_UNLOCK(vh);
        (void) i_mdi_devinfo_remove(vdip, cdip, flags);
        MDI_VHCI_CLIENT_LOCK(vh);

        }
        return (rv);
}
```
______***unchanged_portion_omitted_***