```
**********************************************************
  106839 Thu Jan 17 15:01:41 2019
new/usr/src/lib/libbe/common/be_utils.c
10117 libbe needs smatch fixes
**********************************************************
```

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

  22 /*
  23  * Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
  24  * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
  25  * Copyright 2016 Toomas Soome <tsoome@me.com>
  26  * Copyright (c) 2015 by Delphix. All rights reserved.
  27  * Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
  28  * Copyright (c) 2018, Joyent, Inc.
  29  */


  32 /*
  33  * System includes
  34  */
  35 #include <assert.h>
  36 #include <errno.h>
  37 #include <libgen.h>
  38 #include <libintl.h>
  39 #include <libnvpair.h>
  40 #include <libzfs.h>
  41 #include <libgen.h>
  42 #include <stdio.h>
  43 #include <stdlib.h>
  44 #include <string.h>
  45 #include <sys/stat.h>
  46 #include <sys/types.h>
  47 #include <sys/vfstab.h>
  48 #include <sys/param.h>
  49 #include <sys/systeminfo.h>
  50 #include <ctype.h>
  51 #include <time.h>
  52 #include <unistd.h>
  53 #include <fcntl.h>
  54 #include <deflt.h>
  55 #include <wait.h>
  56 #include <libdevinfo.h>
  57 #include <libgen.h>

  59 #include <libbe.h>
  60 #include <libbe_priv.h>
  61 #include <boot_utils.h>
```



```
  62 #include <ficl.h>
  63 #include <ficlplatform/emu.h>

  65 /* Private function prototypes */
  66 static int update_dataset(char *, int, char *, char *, char *);
  67 static int _update_vfstab(char *, char *, char *, char *, be_fs_list_data_t *);
  68 static int be_open_menu(char *, char *, FILE **, char *, boolean_t);
  69 static int be_create_menu(char *, char *, FILE **, char *);
  70 static char *be_get_auto_name(char *, char *, boolean_t);

  72 /*
  73  * Global error printing
  74  */
  75 boolean_t do_print = B_FALSE;

  77 /*
  78  * Private datatypes
  79  */
  80 typedef struct zone_be_name_cb_data {
  81         char *base_be_name;
  82         int num;
  83 } zone_be_name_cb_data_t;
_____unchanged_portion_omitted_

 101 /*
 102  * Function:    be_get_boot_args
 103  * Description: Returns the fast boot argument string for enumerated BE.
 104  * Parameters:
 105  *              fbarg - pointer to argument string.
 106  *              entry - index of BE.
 107  * Returns:
 108  *              fast boot argument string.
 109  * Scope:
 110  *              Public
 111  */
 112 int
 113 be_get_boot_args(char **fbarg, int entry)
 114 {
 115         be_node_list_t *node, *be_nodes = NULL;
 116         be_transaction_data_t bt = {0};
 117         char *mountpoint = NULL;
 118         boolean_t be_mounted = B_FALSE;
 119         int ret = BE_SUCCESS;
 120         int index;
 121         ficlVm *vm;

 123         *fbarg = NULL;
 124         if (!be_zfs_init())
 125                 return (BE_ERR_INIT);

 127         /*
 128          * need pool name, menu.lst has entries from our pool only
 129          */
 130         ret = be_find_current_be(&bt);
 131         if (ret != BE_SUCCESS) {
 132                 be_zfs_fini();
 133                 return (ret);
 134         }

 136         /*
 137          * be_get_boot_args() is for loader, fail with grub will trigger
 138          * normal boot.
 139          */
 140         if (be_has_grub()) {
 141                 ret = BE_ERR_INIT;
 142                 goto done;
```

```
143         }
144
145         ret = _be_list(NULL, &be_nodes, BE_LIST_DEFAULT);
146         if (ret != BE_SUCCESS)
147                 goto done;
148
149         /*
150          * iterate through be_nodes,
151          * if entry == -1, stop if be_active_on_boot,
152          * else stop if index == entry.
153          */
154         index = 0;
155         for (node = be_nodes; node != NULL; node = node->be_next_node) {
156                 if (strcmp(node->be_rpool, bt.obe_zpool) != 0)
157                         continue;
158                 if (entry == BE_ENTRY_DEFAULT &&
159                     node->be_active_on_boot == B_TRUE)
160                         break;
161                 if (index == entry)
162                         break;
163                 index++;
164         }
165         if (node == NULL) {
166                 be_free_list(be_nodes);
167                 ret = BE_ERR_NOENT;
168                 goto done;
169         }
170
171         /* try to mount inactive be */
172         if (node->be_active == B_FALSE) {
173                 ret = _be_mount(node->be_node_name, &mountpoint,
174                     BE_MOUNT_FLAG_NO_ZONES);
175                 if (ret != BE_SUCCESS && ret != BE_ERR_MOUNTED) {
176                         be_free_list(be_nodes);
177                         goto done;
178                 } else
179                         be_mounted = B_TRUE;
180         }
181
182         vm = bf_init("", ficlSuppressTextOutput);
183         if (vm != NULL) {
184                 /*
185                  * zfs MAXNAMELEN is 256, so we need to pick buf large enough
186                  * to contain such names.
187                  */
188                 char buf[MAXNAMELEN * 2];
189                 char *kernel_options = NULL;
190                 char *kernel = NULL;
191                 char *tmp;
192                 zpool_handle_t *zph;
193
194                 /*
195                  * just try to interpret following words. on error
196                  * we will be missing kernelname, and will get out.
197                  */
198                 (void) snprintf(buf, sizeof (buf), "set currdev=zfs:%s:",
199                     node->be_root_ds);
200                 ret = ficlVmEvaluate(vm, buf);
201                 if (ret != FICL_VM_STATUS_OUT_OF_TEXT) {
202                         be_print_err(gettext("be_get_boot_args: error "
203                             "interpreting boot config: %d\n"), ret);
204                         bf_fini();
205                         ret = BE_ERR_NO_MENU;
206                         goto cleanup;
207                 }
208                 (void) snprintf(buf, sizeof (buf),
```



```
209                     "include /boot/forth/loader.4th");
210                 ret = ficlVmEvaluate(vm, buf);
211                 if (ret != FICL_VM_STATUS_OUT_OF_TEXT) {
212                         be_print_err(gettext("be_get_boot_args: error "
213                             "interpreting boot config: %d\n"), ret);
214                         bf_fini();
215                         ret = BE_ERR_NO_MENU;
216                         goto cleanup;
217                 }
218                 (void) snprintf(buf, sizeof (buf), "start");
219                 ret = ficlVmEvaluate(vm, buf);
220                 if (ret != FICL_VM_STATUS_OUT_OF_TEXT) {
221                         be_print_err(gettext("be_get_boot_args: error "
222                             "interpreting boot config: %d\n"), ret);
223                         bf_fini();
224                         ret = BE_ERR_NO_MENU;
225                         goto cleanup;
226                 }
227                 (void) snprintf(buf, sizeof (buf), "boot");
228                 ret = ficlVmEvaluate(vm, buf);
229                 bf_fini();
230                 if (ret != FICL_VM_STATUS_OUT_OF_TEXT) {
231                         be_print_err(gettext("be_get_boot_args: error "
232                             "interpreting boot config: %d\n"), ret);
233                         ret = BE_ERR_NO_MENU;
234                         goto cleanup;
235                 }
236
237                 kernel_options = getenv("boot-args");
238                 kernel = getenv("kernelname");
239
240                 if (kernel == NULL) {
241                         be_print_err(gettext("be_get_boot_args: no kernel\n"));
242                         ret = BE_ERR_NOENT;
243                         goto cleanup;
244                 }
245
246                 if ((zph = zpool_open(g_zfs, node->be_rpool)) == NULL) {
247                         be_print_err(gettext("be_get_boot_args: failed to "
248                             "open root pool (%s): %s\n"), node->be_rpool,
249                             libzfs_error_description(g_zfs));
250                         ret = zfs_err_to_be_err(g_zfs);
251                         goto cleanup;
252                 }
253                 ret = zpool_get_physpath(zph, buf, sizeof (buf));
254                 zpool_close(zph);
255                 if (ret != 0) {
256                         be_print_err(gettext("be_get_boot_args: failed to "
257                             "get physpath\n"));
258                         goto cleanup;
259                 }
260
261                 /* zpool_get_physpath() can return space separated list */
262                 tmp = buf;
263                 tmp = strsep(&tmp, " ");
264
265                 if (kernel_options == NULL || *kernel_options == '\0')
266                         (void) asprintf(fbarg, "/ %s "
267                             "-B zfs-bootfs=%s,bootpath=\"%s\"\n", kernel,
268                             node->be_root_ds, tmp);
269                 else
270                         (void) asprintf(fbarg, "/ %s %s "
271                             "-B zfs-bootfs=%s,bootpath=\"%s\"\n", kernel,
272                             kernel_options, node->be_root_ds, tmp);
273
274                 if (*fbarg == NULL)
```

```
 273                 if (fbarg == NULL)
 275                         ret = BE_ERR_NOMEM;
 276                 else
 277                         ret = 0;
 278         } else
 279                 ret = BE_ERR_NOMEM;
 280 cleanup:
 281         if (be_mounted == B_TRUE)
 282                 (void) _be_unmount(node->be_node_name, BE_UNMOUNT_FLAG_FORCE);
 283         be_free_list(be_nodes);
 284 done:
 285         free(mountpoint);
 286         free(bt.obe_name);
 287         free(bt.obe_root_ds);
 288         free(bt.obe_zpool);
 289         free(bt.obe_snap_name);
 290         free(bt.obe_altroot);
 291         be_zfs_fini();
 292         return (ret);
 293 }
```

unchanged_portion_omitted_

```
3863 /*
3864  * Function:    be_create_menu
3865  * Description:
3866  *              This function is used if no menu.lst file exists. In
3867  *              this case a new file is created and if needed default
3868  *              lines are added to the file.
3869  * Parameters:
3870  *              pool - The name of the pool the menu.lst file is on
3871  *              menu_file - The name of the file we're creating.
3872  *              menu_fp - A pointer to the file pointer of the file we
3873  *                        created. This is also used to pass back the file
3874  *                        pointer to the newly created file.
3875  *              mode - the original mode used for the failed attempt to
3876  *                     non-existent file.
3877  * Returns:
3878  *              BE_SUCCESS - Success
3879  *              be_errno_t - Failure
3880  * Scope:
3881  *              Private
3882  */
3883 static int
3884 be_create_menu(
3885         char *pool,
3886         char *menu_file,
3887         FILE **menu_fp,
3888         char *mode)
3889 {
3890         be_node_list_t  *be_nodes = NULL;
3891         char *menu_path = NULL;
3892         char *be_rpool = NULL;
3893         char *be_name = NULL;
3894         char *console = NULL;
3895         errno = 0;

3897         if (menu_file == NULL || menu_fp == NULL || mode == NULL)
3898                 return (BE_ERR_INVAL);

3900         menu_path = strdup(menu_file);
3901         if (menu_path == NULL)
3902                 return (BE_ERR_NOMEM);

3904         (void) dirname(menu_path);
3905         if (*menu_path == '.') {
3906                 free(menu_path);
```



```
3907                 return (BE_ERR_BAD_MENU_PATH);
3908         }
3909         if (mkdirp(menu_path,
3910             S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH) == -1 &&
3911             errno != EEXIST) {
3911                 free(menu_path);
3912                 be_print_err(gettext("be_create_menu: Failed to create the %s "
3913                     "directory: %s\n"), menu_path, strerror(errno));
3914                 free(menu_path);
3915                 return (errno_to_be_err(errno));
3916         }
3917         free(menu_path);

3919         /*
3920          * Check to see if this system supports grub
3921          */
3922         if (be_has_grub()) {
3923                 /*
3924                  * The grub menu is missing so we need to create it
3925                  * and fill in the first few lines.
3926                  */
3927                 FILE *temp_fp = fopen(menu_file, "a+");
3928                 if (temp_fp == NULL) {
3929                         *menu_fp = NULL;
3930                         return (errno_to_be_err(errno));
3931                 }

3933                 if ((console = be_get_console_prop()) != NULL) {

3935                         /*
3936                          * If console is redirected to serial line,
3937                          * GRUB splash screen will not be enabled.
3938                          */
3939                         if (strncmp(console, "text", strlen("text")) == 0 ||
3940                             strncmp(console, "graphics",
3941                             strlen("graphics")) == 0) {

3943                                 (void) fprintf(temp_fp, "%s\n", BE_GRUB_SPLASH);
3944                                 (void) fprintf(temp_fp, "%s\n",
3945                                     BE_GRUB_FOREGROUND);
3946                                 (void) fprintf(temp_fp, "%s\n",
3947                                     BE_GRUB_BACKGROUND);
3948                                 (void) fprintf(temp_fp, "%s\n",
3949                                     BE_GRUB_DEFAULT);
3950                         } else {
3951                                 be_print_err(gettext("be_create_menu: "
3952                                     "console on serial line, "
3953                                     "GRUB splash image will be disabled\n"));
3954                         }
3955                 }

3957                 (void) fprintf(temp_fp, "timeout 30\n");
3958                 (void) fclose(temp_fp);

3960         } else {
3961                 /*
3962                  * The menu file doesn't exist so we need to create a
3963                  * blank file.
3964                  */
3965                 FILE *temp_fp = fopen(menu_file, "w+");
3966                 if (temp_fp == NULL) {
3967                         *menu_fp = NULL;
3968                         return (errno_to_be_err(errno));
3969                 }
3970                 (void) fclose(temp_fp);
3971         }
```

```
3973         /*
3974          * Now we need to add all the BE's back into the the file.
3975          */
3976         if (_be_list(NULL, &be_nodes, BE_LIST_DEFAULT) == BE_SUCCESS) {
3977                 while (be_nodes != NULL) {
3978                         if (strcmp(pool, be_nodes->be_rpool) == 0) {
3979                                 (void) be_append_menu(be_nodes->be_node_name,
3980                                     be_nodes->be_rpool, NULL, NULL, NULL);
3981                         }
3982                         if (be_nodes->be_active_on_boot) {
3983                                 be_rpool = strdup(be_nodes->be_rpool);
3984                                 be_name = strdup(be_nodes->be_node_name);
3985                         }

3987                         be_nodes = be_nodes->be_next_node;
3988                 }
3989         }
3990         be_free_list(be_nodes);

3992         /*
3993          * Check to see if this system supports grub
3994          */
3995         if (be_has_grub()) {
3996                 int err = be_change_grub_default(be_name, be_rpool);
3997                 if (err != BE_SUCCESS)
3998                         return (err);
3999         }
4000         *menu_fp = fopen(menu_file, mode);
4001         if (*menu_fp == NULL)
4002                 return (errno_to_be_err(errno));

4004         return (BE_SUCCESS);
4005 }
```

_____***unchanged_portion_omitted_***