****
111823 Mon Sep 9 20:46:53 2013
new/usr/src/uts/common/krtld/kobj.c
4122 do_sysfile_cmd colon-separates the module path, and then we can't parse it
****
______***unchanged_portion_omitted_***

```
/*
 * fullname is dynamically allocated to be able to hold the
 * maximum size string that can be constructed from name.
 * path is exactly like the shell PATH variable.
 */
struct _buf *
kobj_open_path(char *name, int use_path, int use_moddir_suffix)
{
        char *p, *q;
        char *pathp;
        char *pathpsave;
        char *fullname;
        int maxpathlen;
        struct _buf *file;

#if !defined(MODDIR_SUFFIX)
        use_moddir_suffix = B_FALSE;
#endif

        if (!use_path)
                pathp = "";             /* use name as specified */
        else
                pathp = kobj_module_path;
                                        /* use configured default path */

        pathpsave = pathp;              /* keep this for error reporting */

        /*
         * Allocate enough space for the largest possible fullname.
         * since path is of the form <directory> : <directory> : ...
         * we're potentially allocating a little more than we need to
         * but we'll allocate the exact amount when we find the right directory.
         * (The + 3 below is one for NULL terminator and one for the '/'
         * we might have to add at the beginning of path and one for
         * the '/' between path and name.)
         */
        maxpathlen = strlen(pathp) + strlen(name) + 3;
        /* sizeof includes null */
        maxpathlen += sizeof (slash_moddir_suffix_slash) - 1;
        fullname = kobj_zalloc(maxpathlen, KM_WAIT);

        for (;;) {
                p = fullname;
                if (*pathp != '\0' && *pathp != '/')
                        *p++ = '/';     /* path must start with '/' */
                while (*pathp && *pathp != ':' && *pathp != ' ')
                        *p++ = *pathp++;
                if (p != fullname && p[-1] != '/')
                        *p++ = '/';
                if (use_moddir_suffix) {
                        char *b = basename(name);
                        char *s;

                        /* copy everything up to the base name */
                        q = name;
                        while (q != b && *q)
                                *p++ = *q++;
                        s = slash_moddir_suffix_slash;
                        while (*s)
```

```
                                *p++ = *s++;
                        /* copy the rest */
                        while (*b)
                                *p++ = *b++;
                } else {
                        q = name;
                        while (*q)
                                *p++ = *q++;
                }
                *p = 0;
                if ((file = kobj_open_file(fullname)) != (struct _buf *)-1) {
                        kobj_free(fullname, maxpathlen);
                        return (file);
                }
                while (*pathp == ' ' || *pathp == ':')
                while (*pathp == ' ')
                        pathp++;
                if (*pathp == 0)
                        break;

        }
        kobj_free(fullname, maxpathlen);
        if (_moddebug & MODDEBUG_ERRMSG) {
                _kobj_printf(ops, "can't open %s,", name);
                _kobj_printf(ops, " path is %s\n", pathpsave);
        }
        return ((struct _buf *)-1);
}
```
______***unchanged_portion_omitted_***

****
   74163 Mon Sep 9 20:46:54 2013
new/usr/src/uts/common/os/modsysfile.c
4122 do_sysfile_cmd colon-separates the module path, and then we can't parse it
****

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <sys/types.h>
#include <sys/inttypes.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/user.h>
#include <sys/disp.h>
#include <sys/conf.h>
#include <sys/bootconf.h>
#include <sys/sysconf.h>
#include <sys/sunddi.h>
#include <sys/esunddi.h>
#include <sys/ddi_impldefs.h>
#include <sys/kmem.h>
#include <sys/vmem.h>
#include <sys/fs/ufs_fsdir.h>
#include <sys/hwconf.h>
#include <sys/modctl.h>
#include <sys/cmn_err.h>
#include <sys/kobj.h>
#include <sys/kobj_lex.h>
#include <sys/errno.h>
#include <sys/debug.h>
#include <sys/autoconf.h>
#include <sys/callb.h>
#include <sys/sysmacros.h>
#include <sys/dacf.h>
#include <vm/seg_kmem.h>

struct hwc_class *hcl_head;     /* head of list of classes */
static kmutex_t hcl_lock;       /* for accessing list of classes */

#define DAFILE          "/etc/driver_aliases"
#define CLASSFILE       "/etc/driver_classes"
#define DACFFILE        "/etc/dacf.conf"
```

```
static char class_file[] = CLASSFILE;
static char dafile[] = DAFILE;
static char dacffile[] = DACFFILE;

char *systemfile = "/etc/system";       /* name of ascii system file */

static struct sysparam *sysparam_hd;    /* head of parameters list */
static struct sysparam *sysparam_tl;    /* tail of parameters list */
static vmem_t *mod_sysfile_arena;       /* parser memory */

char obp_bootpath[BO_MAXOBJNAME];       /* bootpath from obp */
char svm_bootpath[BO_MAXOBJNAME];       /* bootpath redirected via rootdev */

#if defined(_PSM_MODULES)

struct psm_mach {
        struct psm_mach *m_next;
        char            *m_machname;
};
```
______***unchanged_portion_omitted***_

```
static char bad_op[] = "illegal operator '%s' used on a string";
static char colon_err[] = "A colon (:) must follow the '%s' command";
static char tok_err[] = "Unexpected token '%s'";
static char extra_err[] = "extraneous input ignored starting at '%s'";
static char oversize_err[] = "value too long";

static struct sysparam *
do_sysfile_cmd(struct _buf *file, const char *cmd)
{
        struct sysparam *sysp;
        struct modcmd *mcp;
        token_t token, op;
        char *cp;
        int ch;
        char tok1[MOD_MAXPATH + 1]; /* used to read the path set by 'moddir' */
        char tok2[64];

        for (mcp = modcmd; mcp->mc_cmdname != NULL; mcp++) {
                if (strcmp(mcp->mc_cmdname, cmd) == 0)
                        break;
        }
        sysp = vmem_alloc(mod_sysfile_arena, sizeof (struct sysparam),
            VM_SLEEP);
        bzero(sysp, sizeof (struct sysparam));
        sysp->sys_op = SETOP_NONE; /* set op to noop initially */

        switch (sysp->sys_type = mcp->mc_type) {
        case MOD_INCLUDE:
        case MOD_EXCLUDE:
        case MOD_FORCELOAD:
                /*
                 * Are followed by colon.
                 */
        case MOD_ROOTFS:
        case MOD_SWAPFS:
                if ((token = kobj_lex(file, tok1, sizeof (tok1))) == COLON) {
                        token = kobj_lex(file, tok1, sizeof (tok1));
                } else {
                        kobj_file_err(CE_WARN, file, colon_err, cmd);
                }
                if (token != NAME) {
                        kobj_file_err(CE_WARN, file, "value expected");
                        goto bad;
```

```
		}

		cp = tok1 + strlen(tok1);
		while ((ch = kobj_getc(file)) != -1 && !iswhite(ch) &&
		    !isnewline(ch)) {
			if (cp - tok1 >= sizeof (tok1) - 1) {
				kobj_file_err(CE_WARN, file, oversize_err);
				goto bad;
			}
			*cp++ = (char)ch;
		}
		*cp = '\0';

		if (ch != -1)
			(void) kobj_ungetc(file);
		if (sysp->sys_type == MOD_INCLUDE)
			return (NULL);
		sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(tok1) + 1,
		    VM_SLEEP);
		(void) strcpy(sysp->sys_ptr, tok1);
		break;
	case MOD_SET:
	case MOD_SET64:
	case MOD_SET32:
	{
		char *var;
		token_t tok3;

		if (kobj_lex(file, tok1, sizeof (tok1)) != NAME) {
			kobj_file_err(CE_WARN, file, "value expected");
			goto bad;
		}

		/*
		 * If the next token is a colon (:),
		 * we have the <modname>:<variable> construct.
		 */
		if ((token = kobj_lex(file, tok2, sizeof (tok2))) == COLON) {
			if ((token = kobj_lex(file, tok2,
			    sizeof (tok2))) == NAME) {
				var = tok2;
				/*
				 * Save the module name.
				 */
				sysp->sys_modnam = vmem_alloc(mod_sysfile_arena,
				    strlen(tok1) + 1, VM_SLEEP);
				(void) strcpy(sysp->sys_modnam, tok1);
				op = kobj_lex(file, tok1, sizeof (tok1));
			} else {
				kobj_file_err(CE_WARN, file, "value expected");
				goto bad;
			}
		} else {
			/* otherwise, it was the op */
			var = tok1;
			op = token;
		}
		/*
		 * kernel param - place variable name in sys_ptr.
		 */
		sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(var) + 1,
		    VM_SLEEP);
		(void) strcpy(sysp->sys_ptr, var);
		/* set operation */
		switch (op) {
		case EQUALS:
```

```
			/* simple assignment */
			sysp->sys_op = SETOP_ASSIGN;
			break;
		case AMPERSAND:
			/* bitwise AND */
			sysp->sys_op = SETOP_AND;
			break;
		case BIT_OR:
			/* bitwise OR */
			sysp->sys_op = SETOP_OR;
			break;
		default:
			/* unsupported operation */
			kobj_file_err(CE_WARN, file,
			    "unsupported operator %s", tok2);
			goto bad;
		}

		switch ((tok3 = kobj_lex(file, tok1, sizeof (tok1)))) {
		case STRING:
			/* string variable */
			if (sysp->sys_op != SETOP_ASSIGN) {
				kobj_file_err(CE_WARN, file, bad_op, tok1);
				goto bad;
			}
			if (kobj_get_string(&sysp->sys_info, tok1) == 0) {
				kobj_file_err(CE_WARN, file, "string garbled");
				goto bad;
			}
			/*
			 * Set SYSPARAM_STR_TOKEN in sys_flags to notify
			 * sysparam_print_warning() that this is a string
			 * token.
			 */
			sysp->sys_flags |= SYSPARAM_STR_TOKEN;
			break;
		case HEXVAL:
		case DECVAL:
			if (kobj_getvalue(tok1, &sysp->sys_info) == -1) {
				kobj_file_err(CE_WARN, file,
				    "invalid number '%s'", tok1);
				goto bad;
			}

			/*
			 * Set the appropriate flag (hexadecimal or decimal)
			 * in sys_flags for sysparam_print_warning() to be
			 * able to print the number with the correct format.
			 */
			if (tok3 == HEXVAL) {
				sysp->sys_flags |= SYSPARAM_HEX_TOKEN;
			} else {
				sysp->sys_flags |= SYSPARAM_DEC_TOKEN;
			}
			break;
		default:
			kobj_file_err(CE_WARN, file, "bad rvalue '%s'", tok1);
			goto bad;
		} /* end switch */

		/*
		 * Now that we've parsed it to check the syntax, consider
		 * discarding it (because it -doesn't- apply to this flavor
		 * of the kernel)
		 */
#ifdef _LP64
```

```
                if (sysp->sys_type == MOD_SET32)
                        return (NULL);
#else
                if (sysp->sys_type == MOD_SET64)
                        return (NULL);
#endif
                sysp->sys_type = MOD_SET;
                break;
        }
        case MOD_MODDIR:
                if ((token = kobj_lex(file, tok1, sizeof (tok1))) != COLON) {
                        kobj_file_err(CE_WARN, file, colon_err, cmd);
                        goto bad;
                }

                cp = tok1;
                while ((token = kobj_lex(file, cp,
                    sizeof (tok1) - (cp - tok1))) != NEWLINE && token != EOF) {
                        if (token == -1) {
                                kobj_file_err(CE_WARN, file, oversize_err);
                                goto bad;
                        }
                        cp += strlen(cp);
                        while ((ch = kobj_getc(file)) != -1 && !iswhite(ch) &&
                            !isnewline(ch) && ch != ':') {
                                if (cp - tok1 >= sizeof (tok1) - 1) {
                                        kobj_file_err(CE_WARN, file,
                                            oversize_err);
                                        goto bad;
                                }
                                *cp++ = (char)ch;
                        }
                        *cp++ = ' ';
                        *cp++ = ':';
                        if (isnewline(ch)) {
                                cp--;
                                (void) kobj_ungetc(file);
                        }
                }
                (void) kobj_ungetc(file);
                *cp  = '\0';
                sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(tok1) + 1,
                    VM_SLEEP);
                (void) strcpy(sysp->sys_ptr, tok1);
                break;

        case MOD_SWAPDEV:
        case MOD_ROOTDEV:
                if ((token = kobj_lex(file, tok1, sizeof (tok1))) != COLON) {
                        kobj_file_err(CE_WARN, file, colon_err, cmd);
                        goto bad;
                }
                while ((ch = kobj_getc(file)) == ' ' || ch == '\t')
                        ;
                cp = tok1;
                while (!iswhite(ch) && !isnewline(ch) && ch != -1) {
                        if (cp - tok1 >= sizeof (tok1) - 1) {
                                kobj_file_err(CE_WARN, file, oversize_err);
                                goto bad;
                        }

                        *cp++ = (char)ch;
                        ch = kobj_getc(file);
                }
                if (ch != -1)
                        (void) kobj_ungetc(file);
```

```
                *cp = '\0';

                sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(tok1) + 1,
                    VM_SLEEP);
                (void) strcpy(sysp->sys_ptr, tok1);
                break;

        case MOD_UNKNOWN:
        default:
                kobj_file_err(CE_WARN, file, "unknown command '%s'", cmd);
                goto bad;
        }

        return (sysp);

bad:
        kobj_find_eol(file);
        return (NULL);
}
```

______***unchanged_portion_omitted_***