

```

*****
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_____

3433 /*
3434  * fullname is dynamically allocated to be able to hold the
3435  * maximum size string that can be constructed from name.
3436  * path is exactly like the shell PATH variable.
3437  */
3438 struct _buf *
3439 kobj_open_path(char *name, int use_path, int use_moddir_suffix)
3440 {
3441     char *p, *q;
3442     char *pathp;
3443     char *pathpsave;
3444     char *fullname;
3445     int maxpathlen;
3446     struct _buf *file;

3448 #if !defined(MODDIR_SUFFIX)
3449     use_moddir_suffix = B_FALSE;
3450 #endif

3452     if (!use_path)
3453         pathp = "";          /* use name as specified */
3454     else
3455         pathp = kobj_module_path;
3456                             /* use configured default path */

3458     pathpsave = pathp;     /* keep this for error reporting */

3460     /*
3461     * Allocate enough space for the largest possible fullname.
3462     * since path is of the form <directory> : <directory> : ...
3463     * we're potentially allocating a little more than we need to
3464     * but we'll allocate the exact amount when we find the right directory.
3465     * (The + 3 below is one for NULL terminator and one for the '/'
3466     * we might have to add at the beginning of path and one for
3467     * the '/' between path and name.)
3468     */
3469     maxpathlen = strlen(pathp) + strlen(name) + 3;
3470     /* sizeof includes null */
3471     maxpathlen += sizeof (slash_moddir_suffix_slash) - 1;
3472     fullname = kobj_zalloc(maxpathlen, KM_WAIT);

3474     for (;;) {
3475         p = fullname;
3476         if (*pathp != '\\0' && *pathp != '/')
3477             *p++ = '/';          /* path must start with '/' */
3478         while (*pathp && *pathp != ':' && *pathp != ' ')
3479             *p++ = *pathp++;
3480         if (p != fullname && p[-1] != '/')
3481             *p++ = '/';
3482         if (use_moddir_suffix) {
3483             char *b = basename(name);
3484             char *s;

3486             /* copy everything up to the base name */
3487             q = name;
3488             while (q != b && *q)
3489                 *p++ = *q++;
3490             s = slash_moddir_suffix_slash;
3491             while (*s)

```

```

3492                 *p++ = *s++;
3493                 /* copy the rest */
3494                 while (*b)
3495                     *p++ = *b++;
3496             } else {
3497                 q = name;
3498                 while (*q)
3499                     *p++ = *q++;
3500             }
3501             *p = 0;
3502             if ((file = kobj_open_file(fullname)) != (struct _buf *)-1) {
3503                 kobj_free(fullname, maxpathlen);
3504                 return (file);
3505             }
3506             while (*pathp == ' ' || *pathp == ':')
3507                 pathp++;
3508             if (*pathp == 0)
3509                 break;

3511         }
3512         kobj_free(fullname, maxpathlen);
3513         if (_moddebug & MODDEBUG_ERRMSG) {
3514             _kobj_printf(ops, "can't open %s,", name);
3515             _kobj_printf(ops, " path is %s\n", pathpsave);
3516         }
3517         return ((struct _buf *)-1);
3518     }
_____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
*****

```

```

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 */
21 /*
22 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

```

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

```

26 #include <sys/types.h>
27 #include <sys/inttypes.h>
28 #include <sys/param.h>
29 #include <sys/system.h>
30 #include <sys/user.h>
31 #include <sys/disp.h>
32 #include <sys/conf.h>
33 #include <sys/bootconf.h>
34 #include <sys/sysconf.h>
35 #include <sys/sunddi.h>
36 #include <sys/esunddi.h>
37 #include <sys/ddi_impldefs.h>
38 #include <sys/kmem.h>
39 #include <sys/vmem.h>
40 #include <sys/fs/ufs_fsdir.h>
41 #include <sys/hwconf.h>
42 #include <sys/modctl.h>
43 #include <sys/cmn_err.h>
44 #include <sys/kobj.h>
45 #include <sys/kobj_lex.h>
46 #include <sys/errno.h>
47 #include <sys/debug.h>
48 #include <sys/autoconf.h>
49 #include <sys/callb.h>
50 #include <sys/sysmacros.h>
51 #include <sys/dacf.h>
52 #include <vm/seg_kmem.h>

```

```

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

```

```

57 #define DAFILE      "/etc/driver_aliases"
58 #define CLASSFILE   "/etc/driver_classes"
59 #define DACFFILE    "/etc/dacf.conf"

```

```

61 static char class_file[] = CLASSFILE;
62 static char dafile[] = DAFILE;
63 static char dacffile[] = DACFFILE;

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

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

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

74 #if defined(_PSM_MODULES)

76 struct psm_mach {
77     struct psm_mach *m_next;
78     char *m_machname;
79 };

```

_____ unchanged_portion_omitted _____

```

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

499 static struct sysparam *
500 do_sysfile_cmd(struct _buf *file, const char *cmd)
501 {
502     struct sysparam *sysp;
503     struct modcmd *mcp;
504     token_t token, op;
505     char *cp;
506     int ch;
507     char tok1[MOD_MAXPATH + 1]; /* used to read the path set by 'moddir' */
508     char tok2[64];

510     for (mcp = modcmd; mcp->mc_cmdname != NULL; mcp++) {
511         if (strcmp(mcp->mc_cmdname, cmd) == 0)
512             break;
513     }
514     sysp = vmem_alloc(mod_sysfile_arena, sizeof (struct sysparam),
515                     VM_SLEEP);
516     bzero(sysp, sizeof (struct sysparam));
517     sysp->sys_op = SETOP_NONE; /* set op to noop initially */

519     switch (sysp->sys_type = mcp->mc_type) {
520     case MOD_INCLUDE:
521     case MOD_EXCLUDE:
522     case MOD_FORCELOAD:
523         /*
524          * Are followed by colon.
525          */
526     case MOD_ROOTFS:
527     case MOD_SWAPFS:
528         if ((token = kobj_lex(file, tok1, sizeof (tok1))) == COLON) {
529             token = kobj_lex(file, tok1, sizeof (tok1));
530         } else {
531             kobj_file_err(CE_WARN, file, colon_err, cmd);
532         }
533         if (token != NAME) {
534             kobj_file_err(CE_WARN, file, "value expected");
535             goto bad;

```

```

536     }
537
538     cp = tok1 + strlen(tok1);
539     while ((ch = kobj_getc(file)) != -1 && !iswhite(ch) &&
540           !isnewline(ch)) {
541         if (cp - tok1 >= sizeof (tok1) - 1) {
542             kobj_file_err(CE_WARN, file, oversize_err);
543             goto bad;
544         }
545         *cp++ = (char)ch;
546     }
547     *cp = '\0';
548
549     if (ch != -1)
550         (void) kobj_ungetc(file);
551     if (sysp->sys_type == MOD_INCLUDE)
552         return (NULL);
553     sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(tok1) + 1,
554                               VM_SLEEP);
555     (void) strcpy(sysp->sys_ptr, tok1);
556     break;
557 case MOD_SET:
558 case MOD_SET64:
559 case MOD_SET32:
560 {
561     char *var;
562     token_t tok3;
563
564     if (kobj_lex(file, tok1, sizeof (tok1)) != NAME) {
565         kobj_file_err(CE_WARN, file, "value expected");
566         goto bad;
567     }
568
569     /*
570      * If the next token is a colon (:),
571      * we have the <modname>:<variable> construct.
572      */
573     if ((token = kobj_lex(file, tok2, sizeof (tok2))) == COLON) {
574         if ((token = kobj_lex(file, tok2,
575                               sizeof (tok2))) == NAME) {
576             var = tok2;
577             /*
578              * Save the module name.
579              */
580             sysp->sys_modnam = vmem_alloc(mod_sysfile_arena,
581                                           strlen(tok1) + 1, VM_SLEEP);
582             (void) strcpy(sysp->sys_modnam, tok1);
583             op = kobj_lex(file, tok1, sizeof (tok1));
584         } else {
585             kobj_file_err(CE_WARN, file, "value expected");
586             goto bad;
587         }
588     } else {
589         /* otherwise, it was the op */
590         var = tok1;
591         op = token;
592     }
593     /*
594      * kernel param - place variable name in sys_ptr.
595      */
596     sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(var) + 1,
597                               VM_SLEEP);
598     (void) strcpy(sysp->sys_ptr, var);
599     /* set operation */
600     switch (op) {
601     case EQUALS:

```

```

602         /* simple assignment */
603         sysp->sys_op = SETOP_ASSIGN;
604         break;
605     case AMPERSAND:
606         /* bitwise AND */
607         sysp->sys_op = SETOP_AND;
608         break;
609     case BIT_OR:
610         /* bitwise OR */
611         sysp->sys_op = SETOP_OR;
612         break;
613     default:
614         /* unsupported operation */
615         kobj_file_err(CE_WARN, file,
616                     "unsupported operator %s", tok2);
617         goto bad;
618     }
619
620     switch ((tok3 = kobj_lex(file, tok1, sizeof (tok1)))) {
621     case STRING:
622         /* string variable */
623         if (sysp->sys_op != SETOP_ASSIGN) {
624             kobj_file_err(CE_WARN, file, bad_op, tok1);
625             goto bad;
626         }
627         if (kobj_get_string(&sysp->sys_info, tok1) == 0) {
628             kobj_file_err(CE_WARN, file, "string garbled");
629             goto bad;
630         }
631         /*
632          * Set SYSPARAM_STR_TOKEN in sys_flags to notify
633          * sysparam_print_warning() that this is a string
634          * token.
635          */
636         sysp->sys_flags |= SYSPARAM_STR_TOKEN;
637         break;
638     case HEXVAL:
639     case DECVAl:
640         if (kobj_getvalue(tok1, &sysp->sys_info) == -1) {
641             kobj_file_err(CE_WARN, file,
642                         "invalid number '%s'", tok1);
643             goto bad;
644         }
645
646         /*
647          * Set the appropriate flag (hexadecimal or decimal)
648          * in sys_flags for sysparam_print_warning() to be
649          * able to print the number with the correct format.
650          */
651         if (tok3 == HEXVAL) {
652             sysp->sys_flags |= SYSPARAM_HEX_TOKEN;
653         } else {
654             sysp->sys_flags |= SYSPARAM_DEC_TOKEN;
655         }
656         break;
657     default:
658         kobj_file_err(CE_WARN, file, "bad rvalue '%s'", tok1);
659         goto bad;
660     } /* end switch */
661
662     /*
663      * Now that we've parsed it to check the syntax, consider
664      * discarding it (because it -doesn't- apply to this flavor
665      * of the kernel)
666      */
667     #ifdef _LP64

```

```

668         if (sysp->sys_type == MOD_SET32)
669             return (NULL);
670 #else
671         if (sysp->sys_type == MOD_SET64)
672             return (NULL);
673 #endif
674         sysp->sys_type = MOD_SET;
675         break;
676     }
677 case MOD_MODDIR:
678     if ((token = kobj_lex(file, tok1, sizeof (tok1))) != COLON) {
679         kobj_file_err(CE_WARN, file, colon_err, cmd);
680         goto bad;
681     }
682
683     cp = tok1;
684     while ((token = kobj_lex(file, cp,
685         sizeof (tok1) - (cp - tok1))) != NEWLINE && token != EOF) {
686         if (token == -1) {
687             kobj_file_err(CE_WARN, file, oversize_err);
688             goto bad;
689         }
690         cp += strlen(cp);
691         while ((ch = kobj_getc(file)) != -1 && !iswhite(ch) &&
692             !isnewline(ch) && ch != ':') {
693             if (cp - tok1 >= sizeof (tok1) - 1) {
694                 kobj_file_err(CE_WARN, file,
695                     oversize_err);
696                 goto bad;
697             }
698             *cp++ = (char)ch;
699         }
700         *cp++ = ' ';
701         *cp++ = ':';
702         if (isnewline(ch)) {
703             cp--;
704             (void) kobj_ungetc(file);
705         }
706         (void) kobj_ungetc(file);
707         *cp = '\0';
708         sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(tok1) + 1,
709             VM_SLEEP);
710         (void) strcpy(sysp->sys_ptr, tok1);
711         break;
712
713 case MOD_SWAPDEV:
714 case MOD_ROOTDEV:
715     if ((token = kobj_lex(file, tok1, sizeof (tok1))) != COLON) {
716         kobj_file_err(CE_WARN, file, colon_err, cmd);
717         goto bad;
718     }
719     while ((ch = kobj_getc(file)) == ' ' || ch == '\t')
720         ;
721     cp = tok1;
722     while (!iswhite(ch) && !isnewline(ch) && ch != -1) {
723         if (cp - tok1 >= sizeof (tok1) - 1) {
724             kobj_file_err(CE_WARN, file, oversize_err);
725             goto bad;
726         }
727
728         *cp++ = (char)ch;
729         ch = kobj_getc(file);
730     }
731     if (ch != -1)
732         (void) kobj_ungetc(file);

```

```

733         *cp = '\0';
734
735         sysp->sys_ptr = vmem_alloc(mod_sysfile_arena, strlen(tok1) + 1,
736             VM_SLEEP);
737         (void) strcpy(sysp->sys_ptr, tok1);
738         break;
739
740 case MOD_UNKNOWN:
741     default:
742         kobj_file_err(CE_WARN, file, "unknown command '%s'", cmd);
743         goto bad;
744     }
745
746     return (sysp);
747
748 bad:
749     kobj_find_eol(file);
750     return (NULL);
751 }

```

unchanged portion omitted