```
**********************************************************
     625 Wed May 20 11:53:23 2015
new/usr/src/cmd/make/Makefile.com
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 # Copyright 2015, Richard Lowe.

  14 MAKE_INCLUDE= $(SRC)/cmd/make/include
  15 MAKE_DEFS= -DSYSV -DINTER -DTEAMWARE_MAKE_CMN
  15 MAKE_DEFS= -DSYSV -DINTER
  16 $(RELEASE_BUILD)MAKE_DEFS += -DNDEBUG
  17 CFLAGS += $(CCVERBOSE)
  18 CPPFLAGS += -I$(MAKE_INCLUDE) $(MAKE_DEFS)
```

```
**********************************************************
    1740 Wed May 20 11:53:24 2015
new/usr/src/cmd/make/bin/Makefile
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

# Copyright 2015, Richard Lowe.

PROG=   make
OBJS=   ar.o            \
        depvar.o        \
        dist.o          \
        doname.o        \
        dosys.o         \
        files.o         \
        globals.o       \
        implicit.o      \
        macro.o         \
        main.o          \
        make.o          \
        misc.o          \
        nse_printdep.o  \
        parallel.o      \
        pmake.o         \
        read.o          \
        read2.o         \
        rep.o           \
        state.o

include ../../Makefile.cmd
include ../Makefile.com

LDLIBS += ../lib/mksh/libmksh.a ../lib/mksdmsi18n/libmksdmsi18n.a ../lib/vroot/l
LDLIBS += ../lib/bsd/libbsd.a -lc -lnsl -lumem
LDLIBS += ../lib/bsd/libbsd.a -lc

CPPFLAGS += -D_FILE_OFFSET_BITS=64

ROOTLINKS = $(ROOTCCSBIN)/make $(ROOTXPG4BIN)/make $(ROOTBIN)/dmake $(ROOTCCSLIB
        $(ROOTLIB)/svr4.make

ROOTRULES = $(ROOTSHLIB)/make/make.rules $(ROOTSHLIB)/make/svr4.make.rules

all:    $(PROG)

install: all $(ROOTPROG) $(ROOTLINKS) $(ROOTRULES)

$(PROG):        $(OBJS)
        $(LINK.cc) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

$(ROOTCCSBIN)/make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@

$(ROOTCCSLIB)/svr4.make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@
```



```
$(ROOTLIB)/svr4.make:
        -$(RM) $@; $(SYMLINK) ../bin/make $@

$(ROOTXPG4BIN)/make:
        -$(RM) $@; $(SYMLINK) ../../bin/make $@

$(ROOTBIN)/dmake:
        -$(RM) $@; $(SYMLINK) ./make $@

$(ROOTRULES) := FILEMODE = 0444

$(ROOTRULES): $(ROOTSHLIB)/make

$(ROOTSHLIB)/make: FRC
        $(INS.dir)

$(ROOTSHLIB)/make/%: %.file
        $(INS.rename)

lint:

clean:
        $(RM) $(OBJS)

FRC:

include ../../Makefile.targ
```

```
**********************************************************
  103949 Wed May 20 11:53:24 2015
new/usr/src/cmd/make/bin/doname.cc
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
```

```
/*
 * 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 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      doname.c
 *
 *      Figure out which targets are out of date and rebuild them
 */

/*
 * Included files
 */
#include <alloca.h>             /* alloca() */
#if defined(TEAMWARE_MAKE_CMN)
#include <avo/util.h>           /* avo_get_user(), avo_hostname() */
#endif

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <avo/strings.h> /* AVO_STRDUP() */
#       include <dm/Avo_MToolJobResultMsg.h>
#       include <dm/Avo_MToolJobStartMsg.h>
#       include <dm/Avo_MToolRsrcInfoMsg.h>
#       include <dm/Avo_macro_defs.h> /* AVO_BLOCK_INTERUPTS & AVO_UNBLOCK_INTER
#       include <dmthread/Avo_ServerState.h>
#       include <rw/pstream.h>
#       include <rw/xdrstrea.h>
#endif

#include <fcntl.h>
#include <mk/defs.h>
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/macro.h>         /* getvar(), expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <poll.h>


#include <signal.h>

#       include <stropts.h>
```



```
#include <sys/errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>        /* uname() */
#include <sys/wait.h>
#include <unistd.h>             /* close() */

/*
 * Defined macros
 */
#       define LOCALHOST "localhost"

#define MAXRULES 100

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#define SEND_MTOOL_MSG(cmds) \
        if (send_mtool_msgs) { \
                cmds \
        }
#else
#define SEND_MTOOL_MSG(cmds)
#endif

// Sleep for .1 seconds between stat()'s
const int       STAT_RETRY_SLEEP_TIME = 100000;

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static char     hostName[MAXNAMELEN] = "";
static char     userName[MAXNAMELEN] = "";

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
        static FILE     *mtool_msgs_fp;
        static XDR      xdrs;
        static int      sent_rsrc_info_msg = 0;
#endif

static int      second_pass = 0;

/*
 * File table of contents
 */
extern  Doname          doname_check(register Name target, register Boolean do_g
extern  Doname          doname(register Name target, register Boolean do_get, re
static  Boolean         check_dependencies(Doname *result, Property line, Boolea
void            dynamic_dependencies(Name target);
static  Doname          run_command(register Property line, Boolean print_machin
extern  Doname          execute_serial(Property line);
extern  Name            vpath_translation(register Name cmd);
extern  void            check_state(Name temp_file_name);
static  void            read_dependency_file(register Name filename);
static  void            check_read_state_file(void);
static  void            do_assign(register Name line, register Name target);
static  void            build_command_strings(Name target, register Property lin
static  Doname          touch_command(register Property line, register Name targ
extern  void            update_target(Property line, Doname result);
static  Doname          sccs_get(register Name target, register Property *comman
extern  void            read_directory_of_file(register Name file);
static  void            add_pattern_conditionals(register Name target);
extern  void            set_locals(register Name target, register Property old_l
```

```
125 extern  void            reset_locals(register Name target, register Property old
126 extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
127 static  void            delete_query_chain(Chain ch);

129 // From read2.cc
130 extern  Name            normalize_name(register wchar_t *name_string, register i


133 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
134         static void             append_job_result_msg(Avo_MToolJobResultMsg *job
135         static int              pollResults(char *outFn, char *errFn, char *host
136         static void             pollResultsAction(char *outFn, char *errFn);
137         static void             rxmGetNextResultsBlock(int fd);
138         static int              us_sleep(unsigned int nusecs);
139         extern "C" void         Avo_PollResultsAction_Sigusr1Handler(int foo);
140 #endif

142 /*
143  * DONE.
144  *
145  *      doname_check(target, do_get, implicit, automatic)
146  *
147  *      Will call doname() and then inspect the return value
148  *
149  *      Return value:
150  *                              Indication if the build failed or not
151  *
152  *      Parameters:
153  *              target          The target to build
154  *              do_get          Passed thru to doname()
155  *              implicit        Passed thru to doname()
156  *              automatic       Are we building a hidden dependency?
157  *
158  *      Global variables used:
159  *              build_failed_seen       Set if -k is on and error occurs
160  *              continue_after_error    Indicates that -k is on
161  *              report_dependencies     No error msg if -P is on
162  */
163 Doname
164 doname_check(register Name target, register Boolean do_get, register Boolean imp
165 {
166         int first_time = 1;
167         (void) fflush(stdout);
168 try_again:
169         switch (doname(target, do_get, implicit, automatic)) {
170         case build_ok:
171                 second_pass = 0;
172                 return build_ok;
173         case build_running:
174                 second_pass = 0;
175                 return build_running;
176         case build_failed:
177                 if (!continue_after_error) {
178                         fatal(catgets(catd, 1, 13, "Target '%s' not remade becau
179                               target->string_mb);
180                 }
181                 build_failed_seen = true;
182                 second_pass = 0;
183                 return build_failed;
184         case build_dont_know:
185                 /*
186                  * If we can't figure out how to build an automatic
187                  * (hidden) dependency, we just ignore it.
188                  * We later declare the target to be out of date just in
189                  * case something changed.
190                  * Also, don't complain if just reporting the dependencies
```



```
191                  * and not building anything.
192                  */
193                 if (automatic || (report_dependencies_level > 0)) {
194                         second_pass = 0;
195                         return build_dont_know;
196                 }
197                 if(first_time) {
198                         first_time = 0;
199                         second_pass = 1;
200                         goto try_again;
201                 }
202                 second_pass = 0;
203                 if (continue_after_error && !svr4) {
204                         warning(catgets(catd, 1, 14, "Don't know how to make tar
205                                 target->string_mb);
206                         build_failed_seen = true;
207                         return build_failed;
208                 }
209                 fatal(catgets(catd, 1, 15, "Don't know how to make target '%s'")
210                 break;
211         }
212 #ifdef lint
213         return build_failed;
214 #endif
215 }
______unchanged_portion_omitted_

3277 /*
3278  *      set_locals(target, old_locals)
3279  *
3280  *      Sets any conditional macros for the target.
3281  *      Each target carries a possibly empty set of conditional properties.
3282  *
3283  *      Parameters:
3284  *              target          The target to set conditional macros for
3285  *              old_locals      Space to store old values in
3286  *
3287  *      Global variables used:
3288  *              debug_level     Should we trace activity?
3289  *              is_conditional  We need to preserve this value
3290  *              recursion_level Used for tracing
3291  */
3292 void
3293 set_locals(register Name target, register Property old_locals)
3294 {
3295         register Property       conditional;
3296         register int            i;
3297         register Boolean        saved_conditional_macro_used;
3298         Chain                   cond_name;
3299         Chain                   cond_chain;

3304 #ifdef DISTRIBUTED
3301         if (target->dont_activate_cond_values) {
3302                 return;
3303         }
3308 #endif

3305         saved_conditional_macro_used = conditional_macro_used;

3307         /* Scan the list of conditional properties and apply each one */
3308         for (conditional = get_prop(target->prop, conditional_prop), i = 0;
3309              conditional != NULL;
3310              conditional = get_prop(conditional->next, conditional_prop),
3311              i++) {
3312                 /* Save the old value */
3313                 old_locals[i].body.macro =
```

```
3314                    maybe_append_prop(conditional->body.conditional.name,
3315                                      macro_prop)->body.macro;
3316                if (debug_level > 1) {
3317                        (void) printf(catgets(catd, 1, 38, "%*sActivating condit
3318                                      recursion_level,
3319                                      "");
3320                }
3321                /* Set the conditional value. Macros are expanded when the */
3322                /* macro is refd as usual */
3323                if ((conditional->body.conditional.name != virtual_root) ||
3324                    (conditional->body.conditional.value != virtual_root)) {
3325                        (void) SETVAR(conditional->body.conditional.name,
3326                                      conditional->body.conditional.value,
3327                                      (Boolean) conditional->body.conditional.ap
3328                }
3329                cond_name = ALLOC(Chain);
3330                cond_name->name = conditional->body.conditional.name;
3331        }
3332        /* Put this target on the front of the chain of conditional targets */
3333        cond_chain = ALLOC(Chain);
3334        cond_chain->name = target;
3335        cond_chain->next = conditional_targets;
3336        conditional_targets = cond_chain;
3337        conditional_macro_used = saved_conditional_macro_used;
3338 }

3340 /*
3341  *      reset_locals(target, old_locals, conditional, index)
3342  *
3343  *      Removes any conditional macros for the target.
3344  *
3345  *      Parameters:
3346  *              target          The target we are retoring values for
3347  *              old_locals      The values to restore
3348  *              conditional     The first conditional block for the target
3349  *              index           into the old_locals vector
3350  *      Global variables used:
3351  *              debug_level     Should we trace activities?
3352  *              recursion_level Used for tracing
3353  */
3354 void
3355 reset_locals(register Name target, register Property old_locals, register Proper
3356 {
3357        register Property       this_conditional;
3358        Chain                   cond_chain;

3365 #ifdef DISTRIBUTED
3360        if (target->dont_activate_cond_values) {
3361                return;
3362        }
3369 #endif

3364        /* Scan the list of conditional properties and restore the old value */
3365        /* to each one Reverse the order relative to when we assigned macros */
3366        this_conditional = get_prop(conditional->next, conditional_prop);
3367        if (this_conditional != NULL) {
3368                reset_locals(target, old_locals, this_conditional, index+1);
3369        } else {
3370                /* Remove conditional target from chain */
3371                if (conditional_targets == NULL ||
3372                    conditional_targets->name != target) {
3373                        warning(catgets(catd, 1, 39, "Internal error: reset targ
3374                } else {
3375                        cond_chain = conditional_targets->next;
3376                        retmem_mb((caddr_t) conditional_targets);
3377                        conditional_targets = cond_chain;
```



```
3378                }
3379        }
3380        get_prop(conditional->body.conditional.name->prop,
3381                 macro_prop)->body.macro = old_locals[index].body.macro;
3382        if (conditional->body.conditional.name == virtual_root) {
3383                (void) SETVAR(virtual_root, getvar(virtual_root), false);
3384        }
3385        if (debug_level > 1) {
3386                if (old_locals[index].body.macro.value != NULL) {
3387                        (void) printf(catgets(catd, 1, 40, "%*sdeactivating cond
3388                                      recursion_level,
3389                                      "",
3390                                      conditional->body.conditional.name->
3391                                      string_mb,
3392                                      old_locals[index].body.macro.value->
3393                                      string_mb);
3394                } else {
3395                        (void) printf(catgets(catd, 1, 41, "%*sdeactivating cond
3396                                      recursion_level,
3397                                      "",
3398                                      conditional->body.conditional.name->
3399                                      string_mb);
3400                }
3401        }
3402 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   97949 Wed May 20 11:53:25 2015
new/usr/src/cmd/make/bin/main.cc
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
```

```
   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 2006 Sun Microsystems, Inc. All rights reserved.
  23  * Use is subject to license terms.
  24  */

  26 /*
  27  *      main.cc
  28  *
  29  *      make program main routine plus some helper routines
  30  */
  31
  32 /*
  33  * Included files
  34  */
  35 #if defined(TEAMWARE_MAKE_CMN)
  36 #       include <avo/intl.h>
  37 #       include <avo/libcli.h>          /* libcli_init() */
  38 #       include <avo/cli_license.h>     /* avo_cli_get_license() */
  39 #       include <avo/find_dir.h>        /* avo_find_run_dir() */
  40 #       include <avo/version_string.h>
  41 #       include <avo/util.h>            /* avo_init() */
  42 #       include <avo/cleanup.h>
  37 #endif

  39 #include <bsd/bsd.h>            /* bsd_signal() */

  41 #ifdef DISTRIBUTED
  42 #       include <dm/Avo_AcknowledgeMsg.h>
  43 #       include <rw/xdrstrea.h>
  44 #       include <dmrc/dmrc.h> /* dmakerc file processing */
  45 #endif

  47 #include <locale.h>             /* setlocale() */
  48 #include <mk/defs.h>
  49 #include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */
  50 #include <mksh/macro.h>         /* getvar() */
  51 #include <mksh/misc.h>          /* getmem(), setup_char_semantics() */

  53 #if defined(TEAMWARE_MAKE_CMN)
  54 #endif
```



```
  56 #include <pwd.h>                /* getpwnam() */
  57 #include <setjmp.h>
  58 #include <signal.h>
  59 #include <stdlib.h>
  60 #include <sys/errno.h>          /* ENOENT */
  61 #include <sys/stat.h>           /* fstat() */
  62 #include <fcntl.h>              /* open() */

  64 #       include <sys/systeminfo.h>      /* sysinfo() */

  66 #include <sys/types.h>          /* stat() */
  67 #include <sys/wait.h>           /* wait() */
  68 #include <unistd.h>             /* execv(), unlink(), access() */
  69 #include <vroot/report.h>       /* report_dependency(), get_report_file() */

  71 // From read2.cc
  72 extern  Name            normalize_name(register wchar_t *name_string, register i

  74 // From parallel.cc
  75 #if defined(TEAMWARE_MAKE_CMN)
  76 #define MAXJOBS_ADJUST_RFE4694000

  78 #ifdef MAXJOBS_ADJUST_RFE4694000
  79 extern void job_adjust_fini();
  80 #endif /* MAXJOBS_ADJUST_RFE4694000 */
  81 #endif /* TEAMWARE_MAKE_CMN */


  84 /*
  85  * Defined macros
  86  */
  87 #define MAKE_PREFIX             NOCATGETS("/usr")
  88 #define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT_32")
  89 #define LD_SUPPORT_ENV_VAR_32   NOCATGETS("SGS_SUPPORT_32")
  90 #define LD_SUPPORT_ENV_VAR_64   NOCATGETS("SGS_SUPPORT_64")
  93 #define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT")
  91 #define LD_SUPPORT_MAKE_LIB     NOCATGETS("libmakestate.so.1")
  92 #define LD_SUPPORT_MAKE_LIB_DIR NOCATGETS("/lib")
  93 #define LD_SUPPORT_MAKE_LIB_DIR_64      NOCATGETS("/64")
  94 #endif /* ! codereview */

  96 /*
  97  * typedefs & structs
  98  */

 100 /*
 101  * Static variables
 102  */
 103 static  char            *argv_zero_string;
 104 static  Boolean         build_failed_ever_seen;
 105 static  Boolean         continue_after_error_ever_seen; /* '-k' */
 106 static  Boolean         dmake_group_specified;          /* '-g' */
 107 static  Boolean         dmake_max_jobs_specified;       /* '-j' */
 108 static  Boolean         dmake_mode_specified;           /* '-m' */
 109 static  Boolean         dmake_add_mode_specified;       /* '-x' */
 110 static  Boolean         dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE
 111 static  Boolean         dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M
 112 static  Boolean         dmake_odir_specified;           /* '-o' */
 113 static  Boolean         dmake_rcfile_specified;         /* '-c' */
 114 static  Boolean         env_wins;                       /* '-e' */
 115 static  Boolean         ignore_default_mk;              /* '-r' */
 116 static  Boolean         list_all_targets;               /* '-T' */
 117 static  int             mf_argc;
 118 static  char            **mf_argv;
 119 static  Dependency_rec  not_auto_depen_struct;
 120 static  Dependency      not_auto_depen = &not_auto_depen_struct;
```

```
121 static  Boolean         pmake_cap_r_specified;          /* '-R' */
122 static  Boolean         pmake_machinesfile_specified;   /* '-M' */
123 static  Boolean         stop_after_error_ever_seen;     /* '-S' */
124 static  Boolean         trace_status;                   /* '-p' */

126 #ifdef DMAKE_STATISTICS
127 static  Boolean         getname_stat = false;
128 #endif

130 #if defined(TEAMWARE_MAKE_CMN)
131         static  time_t          start_time;
132         static  int             g_argc;
133         static  char            **g_argv;
 95         static  Avo_cleanup     *cleanup = NULL;
134 #endif

136 /*
137  * File table of contents
138  */
139         extern "C" void         cleanup_after_exit(void);

141 #ifdef TEAMWARE_MAKE_CMN
142 extern "C" {
143         extern  void            dmake_exit_callback(void);
144         extern  void            dmake_message_callback(char *);
145 }
146 #endif

148 extern  Name            normalize_name(register wchar_t *name_string, register i

150 extern  int             main(int, char * []);

152 static  void            append_makeflags_string(Name, String);
153 static  void            doalarm(int);
154 static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
155 static  void            make_targets(int, char **, Boolean);
156 static  int             parse_command_option(char);
157 static  void            read_command_options(int, char **);
158 static  void            read_environment(Boolean);
159 static  void            read_files_and_state(int, char **);
160 static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
161 static  void            report_recursion(Name);
162 static  void            set_sgs_support(void);
163 static  void            setup_for_projectdir(void);
164 static  void            setup_makeflags_argv(void);
165 static  void            report_dir_enter_leave(Boolean entering);

167 extern void expand_value(Name, register String , Boolean);

169 #ifdef DISTRIBUTED
170         extern  int             dmake_ofd;
171         extern  FILE*           dmake_ofp;
172         extern  int             rxmPid;
173         extern  XDR             xdrs_out;
174 #endif
175 #ifdef TEAMWARE_MAKE_CMN
176         static const char       verstring[] = "illumos make";
138         extern  char            verstring[];
177 #endif

179 jmp_buf jmpbuffer;
180 extern nl_catd catd;

182 /*
183  *      main(argc, argv)
184  *
```



```
185  *      Parameters:
186  *              argc                    You know what this is
187  *              argv                    You know what this is
188  *
189  *      Static variables used:
190  *              list_all_targets        make -T seen
191  *              trace_status            make -p seen
192  *
193  *      Global variables used:
194  *              debug_level             Should we trace make actions?
195  *              keep_state              Set if .KEEP_STATE seen
196  *              makeflags               The Name "MAKEFLAGS", used to get macro
197  *              remote_command_name     Name of remote invocation cmd ("on")
198  *              running_list            List of parallel running processes
199  *              stdout_stderr_same      true if stdout and stderr are the same
200  *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
201  *              temp_file_directory     Set to the dir where we create tmp file
202  *              trace_reader            Set to reflect tracing status
203  *              working_on_targets      Set when building user targets
204  */
205 int
206 main(int argc, char *argv[])
207 {
208         /*
209          * cp is a -> to the value of the MAKEFLAGS env var,
210          * which has to be regular chars.
211          */
212         register char           *cp;
213         char                    make_state_dir[MAXPATHLEN];
214         Boolean                 parallel_flag = false;
215         char                    *prognameptr;
216         char                    *slash_ptr;
217         mode_t                  um;
218         int                     i;
219 #ifdef TEAMWARE_MAKE_CMN
220         struct itimerval        value;
221         char                    def_dmakerc_path[MAXPATHLEN];
222         Name                    dmake_name, dmake_name2;
223         Name                    dmake_value, dmake_value2;
224         Property                prop, prop2;
225         struct stat             statbuf;
226         int                     statval;
227 #endif

229         struct stat             out_stat, err_stat;
230         hostid = gethostid();
193 #ifdef TEAMWARE_MAKE_CMN
194         avo_get_user(NULL, NULL); // Initialize user name
195 #endif
231         bsd_signals();

233         (void) setlocale(LC_ALL, "");


236 #ifdef DMAKE_STATISTICS
237         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
238                 getname_stat = true;
239         }
240 #endif


208         /*
209          * avo_init() sets the umask to 0.  Save it here and restore
210          * it after the avo_init() call.
211          */
212 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
```

```
213         um = umask(0);
214         avo_init(argv[0]);
215         umask(um);

217         cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
218 #endif

242 #if defined(TEAMWARE_MAKE_CMN)
243         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
222         libcli_init();
244 #endif

246 // ---> fprintf(stderr, catgets(catd, 15, 666, "--- SUN make ---\n"));


249 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
250 /*
251  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
252  * from avo_util library.
253  */
254         libmksdmsi18n_init();
255 #endif


258 #ifndef TEAMWARE_MAKE_CMN
259         textdomain(NOCATGETS("SUNW_SPRO_MAKE"));
260 #endif /* TEAMWARE_MAKE_CMN */

262 #ifdef TEAMWARE_MAKE_CMN
263         g_argc = argc;
264         g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
265         for (i = 0; i < argc; i++) {
266                 g_argv[i] = argv[i];
267         }
268         g_argv[i] = NULL;
269 #endif /* TEAMWARE_MAKE_CMN */

271         /*
272          * Set argv_zero_string to some form of argv[0] for
273          * recursive MAKE builds.
274          */

276         if (*argv[0] == (int) slash_char) {
277                 /* argv[0] starts with a slash */
278                 argv_zero_string = strdup(argv[0]);
279         } else if (strchr(argv[0], (int) slash_char) == NULL) {
280                 /* argv[0] contains no slashes */
281                 argv_zero_string = strdup(argv[0]);
282         } else {
283                 /*
284                  * argv[0] contains at least one slash,
285                  * but doesn't start with a slash
286                  */
287                 char    *tmp_current_path;
288                 char    *tmp_string;

290                 tmp_current_path = get_current_path();
291                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
292                                 strlen(argv[0]) + 1);
293                 (void) sprintf(tmp_string,
294                                 "%s/%s",
295                                 tmp_current_path,
296                                 argv[0]);
297                 argv_zero_string = strdup(tmp_string);
298                 retmem_mb(tmp_string);
299         }
```



```
301         /*
302          * The following flags are reset if we don't have the
303          * (.nse_depinfo or .make.state) files locked and only set
304          * AFTER the file has been locked. This ensures that if the user
305          * interrupts the program while file_lock() is waiting to lock
306          * the file, the interrupt handler doesn't remove a lock
307          * that doesn't belong to us.
308          */
309         make_state_lockfile = NULL;
310         make_state_locked = false;


313         /*
314          * look for last slash char in the path to look at the binary
315          * name. This is to resolve the hard link and invoke make
316          * in svr4 mode.
317          */

319         /* Sun OS make standart */
320         svr4 = false;
321         posix = false;
322         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
323                 svr4 = false;
324                 posix = true;
325         } else {
326                 prognameptr = strrchr(argv[0], '/');
327                 if(prognameptr) {
328                         prognameptr++;
329                 } else {
330                         prognameptr = argv[0];
331                 }
332                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
333                         svr4 = true;
334                         posix = false;
335                 }
336         }
337         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
338            svr4 = true;
339            posix = false;
340         }

342         /*
343          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
344          */
345         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
346         if (dmake_compat_mode_var != NULL) {
347                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
348                         gnu_style = true;
349                 }
350                 //svr4 = false;
351                 //posix = false;
352         }

354         /*
355          * Temporary directory set up.
356          */
357         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
358         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
359                 strcpy(mbs_buffer, tmpdir_var);
360                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
361                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
362                         *tmpdir_var = '\0');
363                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
364                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
365                                 mbs_buffer, getpid());
```

```
366                         int fd = mkstemp(mbs_buffer2);
367                         if (fd >= 0) {
368                                 close(fd);
369                                 unlink(mbs_buffer2);
370                                 tmpdir = strdup(mbs_buffer);
371                         }
372                 }
373         }

375         /* find out if stdout and stderr point to the same place */
376         if (fstat(1, &out_stat) < 0) {
377                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
378         }
379         if (fstat(2, &err_stat) < 0) {
380                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
381         }
382         if ((out_stat.st_dev == err_stat.st_dev) &&
383             (out_stat.st_ino == err_stat.st_ino)) {
384                 stdout_stderr_same = true;
385         } else {
386                 stdout_stderr_same = false;
387         }
388         /* Make the vroot package scan the path using shell semantics */
389         set_path_style(0);

391         setup_char_semantics();

393         setup_for_projectdir();

395         /*
396          * If running with .KEEP_STATE, curdir will be set with
397          * the connected directory.
398          */
399         (void) atexit(cleanup_after_exit);

401         load_cached_names();

403 /*
404  *      Set command line flags
405  */
406         setup_makeflags_argv();
407         read_command_options(mf_argc, mf_argv);
408         read_command_options(argc, argv);
409         if (debug_level > 0) {
410                 cp = getenv(makeflags->string_mb);
411                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
412         }

414         setup_interrupt(handle_interrupt);

416         read_files_and_state(argc, argv);

418 #ifdef TEAMWARE_MAKE_CMN
419         /*
420          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
421          */
422         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
423         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
424         prop2 = get_prop(dmake_name2->prop, macro_prop);
425         if (prop2 == NULL) {
426                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
427                 output_mode = txt1_mode;
428         } else {
429                 dmake_value2 = prop2->body.macro.value;
430                 if ((dmake_value2 == NULL) ||
431                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
```



```
432                         output_mode = txt1_mode;
433                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
434                         output_mode = txt2_mode;
435                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
436                         output_mode = html1_mode;
437                 } else {
438                         warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
439                               dmake_value2->string_mb);
440                 }
441         }
442         /*
443          * Find the dmake_mode: distributed, parallel, or serial.
444          */
445     if ((!pmake_cap_r_specified) &&
446         (!pmake_machinesfile_specified)) {
447         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
448         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
449         prop2 = get_prop(dmake_name2->prop, macro_prop);
450         if (prop2 == NULL) {
451                 /* DMAKE_MODE not defined, default to distributed mode */
452                 dmake_mode_type = distributed_mode;
453                 no_parallel = false;
454         } else {
455                 dmake_value2 = prop2->body.macro.value;
456                 if ((dmake_value2 == NULL) ||
457                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
458                         dmake_mode_type = distributed_mode;
459                         no_parallel = false;
460                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
461                         dmake_mode_type = parallel_mode;
462                         no_parallel = false;
463                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
464                         dmake_mode_type = serial_mode;
465                         no_parallel = true;
466                 } else {
467                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
468                 }
469         }

471         if ((!list_all_targets) &&
472             (report_dependencies_level == 0)) {
473                 /*
474                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
475                  * They could be defined in the env, in the makefile, or on the
476                  * command line.
477                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
478                  * then print a message, and default to parallel mode.
479                  */
480 #ifdef DISTRIBUTED
481                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
482                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
483                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
484                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
485                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
486                      ((dmake_value = prop->body.macro.value) == NULL)) &&
487                     (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
488                      ((dmake_value2 = prop2->body.macro.value) == NULL))) {
489                         Boolean empty_dmakerc = true;
490                         char *homedir = getenv(NOCATGETS("HOME"));
491                         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
492                                 sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
493                                 if ((((statval = stat(def_dmakerc_path, &statbuf
494                                       ((statval == 0) && (statbuf.st_size == 0
495                                 } else {
496                                       Avo_dmakerc     *rcfile = new Avo_dmaker
497                                       Avo_err         *err = rcfile->read(def_
```

```
498                                         if (err) {
499                                                 fatal(err->str);
500                                         }
501                                         empty_dmakerc = rcfile->was_empty();
502                                         delete rcfile;
503                                 }
504                         }
505                         if (empty_dmakerc) {
506                                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
507                                         putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
508                                         (void) fprintf(stdout, catgets(catd, 1,
509                                         (void) fprintf(stdout, catgets(catd, 1,
510                                 }
511                                 dmake_mode_type = parallel_mode;
512                                 no_parallel = false;
513                         }
514                 }
515 #else
516                 if(dmake_mode_type == distributed_mode) {
517                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
518                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
519                         dmake_mode_type = parallel_mode;
520                         no_parallel = false;
521                 }
522 #endif  /* DISTRIBUTED */
523         }
524     }
525 #endif

526 #ifdef TEAMWARE_MAKE_CMN
527         parallel_flag = true;
508         putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));
509         /* XXX - This is a major hack for DMake/Licensing. */
510         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
511                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
512                         /*
513                          * If the user can not get a TeamWare license,
514                          * default to serial mode.
515                          */
516                         dmake_mode_type = serial_mode;
517                         no_parallel = true;
518                 } else {
519                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
520                 }
521                 start_time = time(NULL);
522                 /*
523                  * XXX - Hack to disable SIGALRM's from licensing library's
524                  *       setitimer().
525                  */
526                 value.it_interval.tv_sec = 0;
527                 value.it_interval.tv_usec = 0;
528                 value.it_value.tv_sec = 0;
529                 value.it_value.tv_usec = 0;
530                 (void) setitimer(ITIMER_REAL, &value, NULL);
        }

529 //
530 // If dmake is running with -t option, set dmake_mode_type to serial.
531 // This is done because doname() calls touch_command() that runs serially.
532 // If we do not do that, maketool will have problems.
533 //
534         if(touch) {
535                 dmake_mode_type = serial_mode;
536                 no_parallel = true;
537         }
538 #else
```



```
539         parallel_flag = false;
540 #endif

542 #if defined (TEAMWARE_MAKE_CMN)
543         /*
544          * Check whether stdout and stderr are physically same.
545          * This is in order to decide whether we need to redirect
546          * stderr separately from stdout.
547          * This check is performed only if __DMAKE_SEPARATE_STDERR
548          * is not set. This variable may be used in order to preserve
549          * the 'old' behaviour.
550          */
551         out_err_same = true;
552         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
553         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
554                 struct stat stdout_stat;
555                 struct stat stderr_stat;
556                 if( (fstat(1, &stdout_stat) == 0)
557                  && (fstat(2, &stderr_stat) == 0) )
558                 {
559                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
560                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
561                         {
562                                 out_err_same = false;
563                         }
564                 }
565         }
566 #endif

568 #ifdef DISTRIBUTED
569         /*
570          * At this point, DMake should startup an rxm with any and all
571          * DMake command line options. Rxm will, among other things,
572          * read the rc file.
573          */
574         if ((!list_all_targets) &&
575             (report_dependencies_level == 0) &&
576             (dmake_mode_type == distributed_mode)) {
577                 startup_rxm();
578         }
579 #endif
580
581 /*
582  *      Enable interrupt handler for alarms
583  */
584         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

586 /*
587  *      Check if make should report
588  */
589         if (getenv(sunpro_dependencies->string_mb) != NULL) {
590                 FILE    *report_file;

592                 report_dependency("");
593                 report_file = get_report_file();
594                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
595                         (void) fprintf(report_file, "\n");
596                 }
597         }

599 /*
600  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
601  */
602         if (keep_state) {
603                 maybe_append_prop(sunpro_dependencies, macro_prop)->
604                   body.macro.exported = true;
```

```
605         } else {
606                 maybe_append_prop(sunpro_dependencies, macro_prop)->
607                   body.macro.exported = false;
608         }

610         working_on_targets = true;
611         if (trace_status) {
612                 dump_make_state();
613                 fclose(stdout);
614                 fclose(stderr);
615                 exit_status = 0;
616                 exit(0);
617         }
618         if (list_all_targets) {
619                 dump_target_list();
620                 fclose(stdout);
621                 fclose(stderr);
622                 exit_status = 0;
623                 exit(0);
624         }
625         trace_reader = false;

627         /*
628          * Set temp_file_directory to the directory the .make.state
629          * file is written to.
630          */
631         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
632                 temp_file_directory = strdup(get_current_path());
633         } else {
634                 *slash_ptr = (int) nul_char;
635                 (void) strcpy(make_state_dir, make_state->string_mb);
636                 *slash_ptr = (int) slash_char;
637                    /* when there is only one slash and it's the first
638                    ** character, make_state_dir should point to '/'.
639                    */
640                 if(make_state_dir[0] == '\0') {
641                    make_state_dir[0] = '/';
642                    make_state_dir[1] = '\0';
643                 }
644                 if (make_state_dir[0] == (int) slash_char) {
645                         temp_file_directory = strdup(make_state_dir);
646                 } else {
647                         char    tmp_current_path2[MAXPATHLEN];
648
649                         (void) sprintf(tmp_current_path2,
650                                         "%s/%s",
651                                         get_current_path(),
652                                         make_state_dir);
653                         temp_file_directory = strdup(tmp_current_path2);
654                 }
655         }

657 #ifdef DISTRIBUTED
658         building_serial = false;
659 #endif

661         report_dir_enter_leave(true);

663         make_targets(argc, argv, parallel_flag);

665         report_dir_enter_leave(false);

667         if (build_failed_ever_seen) {
668                 if (posix) {
669                         exit_status = 1;
670                 }
```



```
671                 exit(1);
672         }
673         exit_status = 0;
674         exit(0);
675         /* NOTREACHED */
676 }
_____unchanged_portion_omitted_
746 #endif

748 /*
749 #ifdef DISTRIBUTED
750     if (get_parent() == TRUE) {
751 #endif
752  */

754         parallel = false;
755         /* If we used the SVR4_MAKE, don't build .DONE or .FAILED */
756         if (!getenv(USE_SVR4_MAKE)){
757             /* Build the target .DONE or .FAILED if we caught an error */
758             if (!quest && !list_all_targets) {
759                 Name            failed_name;

761                 MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
762                 failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
763                 if ((exit_status != 0) && (failed_name->prop != NULL)) {
764 #ifdef TEAMWARE_MAKE_CMN
765                         /*
766                          * [tolik] switch DMake to serial mode
767                          */
768                         dmake_mode_type = serial_mode;
769                         no_parallel = true;
770 #endif
771                         (void) doname(failed_name, false, true);
772                 } else {
773                     if (!trace_status) {
774 #ifdef TEAMWARE_MAKE_CMN
775                         /*
776                          * Switch DMake to serial mode
777                          */
778                         dmake_mode_type = serial_mode;
779                         no_parallel = true;
780 #endif
781                         (void) doname(done, false, true);
782                     }
783                 }
784             }
785         }
786         /*
787          * Remove the temp file utilities report dependencies thru if it
788          * is still around
789          */
790         if (temp_file_name != NULL) {
791                 (void) unlink(temp_file_name->string_mb);
792         }
793         /*
794          * Do not save the current command in .make.state if make
795          * was interrupted.
796          */
797         if (current_line != NULL) {
798                 command_changed = true;
799                 current_line->body.line.command_used = NULL;
800         }
801         /*
802          * For each parallel build process running, remove the temp files
803          * and zap the command line so it won't be put in .make.state
804          */
```

```
805         for (rp = running_list; rp != NULL; rp = rp->next) {
806                 if (rp->temp_file != NULL) {
807                         (void) unlink(rp->temp_file->string_mb);
808                 }
809                 if (rp->stdout_file != NULL) {
810                         (void) unlink(rp->stdout_file);
811                         retmem_mb(rp->stdout_file);
812                         rp->stdout_file = NULL;
813                 }
814                 if (rp->stderr_file != NULL) {
815                         (void) unlink(rp->stderr_file);
816                         retmem_mb(rp->stderr_file);
817                         rp->stderr_file = NULL;
818                 }
819                 command_changed = true;
820 /*
821                 line = get_prop(rp->target->prop, line_prop);
822                 if (line != NULL) {
823                         line->body.line.command_used = NULL;
824                 }
825  */
826         }
827         /* Remove the statefile lock file if the file has been locked */
828         if ((make_state_lockfile != NULL) && (make_state_locked)) {
829                 (void) unlink(make_state_lockfile);
830                 make_state_lockfile = NULL;
831                 make_state_locked = false;
832         }
833         /* Write .make.state */
834         write_state_file(1, (Boolean) 1);

839 #ifdef TEAMWARE_MAKE_CMN
840         // Deleting the usage tracking object sends the usage mail
841         cleanup->set_exit_status(exit_status);
842         delete cleanup;
843 #endif

835 /*
836 #ifdef DISTRIBUTED
837     }
838 #endif
839  */

841 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
842         job_adjust_fini();
843 #endif

845 #ifdef TEAMWARE_MAKE_CMN
846         catclose(catd);
847 #endif
848 #ifdef DISTRIBUTED
849         if (rxmPid > 0) {
850                 // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
851                 Avo_AcknowledgeMsg acknowledgeMsg;
852                 RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

854                 int xdrResult = xdr(&xdrs_out, msg);

856                 if (xdrResult) {
857                         fflush(dmake_ofp);
858                 } else {
859 /*
860                         fatal(catgets(catd, 1, 266, "couldn't tell rxm to exit")
861  */
862                         kill(rxmPid, SIGTERM);
863                 }
```



```
865                 waitpid(rxmPid, NULL, 0);
866                 rxmPid = 0;
867         }
868 #endif
869 }
```

_____***unchanged_portion_omitted_***

```
1816 /*
1817  *      set_sgs_support()
1818  *
1819  *      Add the libmakestate.so.1 lib to the env var SGS_SUPPORT
1820  *        if it's not already in there.
1821  *      The SGS_SUPPORT env var and libmakestate.so.1 is used by
1822  *        the linker ld to report .make.state info back to make.
1823  *
1824  * In the new world we always will set the 32-bit and 64-bit versions of this
1825  * variable explicitly so that we can take into account the correct isa and our
1826  * prefix. So say that the prefix was /opt/local. Then we would want to search
1827  * /opt/local/lib/libmakestate.so.1:libmakestate.so.1. We still want to search
1828  * the original location just as a safety measure.
1829 #endif /* ! codereview */
1830  */
1831 static void
1832 set_sgs_support()
1833 {
1834         int             len;
1835         char            *newpath, *newpath64;
1836         char            *oldpath, *oldpath64;
1837         static char     *prev_path, *prev_path64;
1833         char            *newpath;
1834         char            *oldpath;
1835         static char     *prev_path;

1839         oldpath = getenv(LD_SUPPORT_ENV_VAR_32);
1837         oldpath = getenv(LD_SUPPORT_ENV_VAR);
1840         if (oldpath == NULL) {
1841                 len = snprintf(NULL, 0, "%s=%s/%s/%s:%s",
1842                     LD_SUPPORT_ENV_VAR_32,
1843                     MAKE_PREFIX,
1844                     LD_SUPPORT_MAKE_LIB_DIR,
1845                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB) + 1;
1839                 len = strlen(LD_SUPPORT_ENV_VAR) + 1 +
1840                         strlen(LD_SUPPORT_MAKE_LIB) + 1;
1846                 newpath = (char *) malloc(len);
1847                 sprintf(newpath, "%s=%s/%s/%s:%s",
1848                     LD_SUPPORT_ENV_VAR_32,
1849                     MAKE_PREFIX,
1850                     LD_SUPPORT_MAKE_LIB_DIR,
1851                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB);
1852         } else {
1853                 len = snprintf(NULL, 0, "%s=%s:%s/%s/%s:%s",
1854                     LD_SUPPORT_ENV_VAR_32, oldpath, MAKE_PREFIX,
1855                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB,
1856                     LD_SUPPORT_MAKE_LIB) + 1;
1842                 sprintf(newpath, "%s=", LD_SUPPORT_ENV_VAR);
1843         } else {
1844                 len = strlen(LD_SUPPORT_ENV_VAR) + 1 + strlen(oldpath) + 1 +
1845                         strlen(LD_SUPPORT_MAKE_LIB) + 1;
1857                 newpath = (char *) malloc(len);
1858                 sprintf(newpath, "%s=%s:%s/%s/%s:%s",
1859                     LD_SUPPORT_ENV_VAR_32, oldpath, MAKE_PREFIX,
1860                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB,
1861                     LD_SUPPORT_MAKE_LIB);
1862         }
```

```
1864         oldpath64 = getenv(LD_SUPPORT_ENV_VAR_64);
1865         if (oldpath64 == NULL) {
1866                 len = snprintf(NULL, 0, "%s=%s/%s/%s/%s:%s",
1867                     LD_SUPPORT_ENV_VAR_64, MAKE_PREFIX, LD_SUPPORT_MAKE_LIB_DIR,
1868                     LD_SUPPORT_MAKE_LIB_DIR_64, LD_SUPPORT_MAKE_LIB,
1869                     LD_SUPPORT_MAKE_LIB) + 1;
1870                 newpath64 = (char *) malloc(len);
1871                 sprintf(newpath64, "%s=%s/%s/%s/%s:%s",
1872                     LD_SUPPORT_ENV_VAR_64, MAKE_PREFIX, LD_SUPPORT_MAKE_LIB_DIR,
1873                     LD_SUPPORT_MAKE_LIB_DIR_64, LD_SUPPORT_MAKE_LIB,
1874                     LD_SUPPORT_MAKE_LIB);
1875         } else {
1876                 len = snprintf(NULL, 0, "%s=%s:%s/%s/%s/%s:%s",
1877                     LD_SUPPORT_ENV_VAR_64, oldpath64, MAKE_PREFIX,
1878                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB_DIR_64,
1879                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB) + 1;
1880                 newpath64 = (char *) malloc(len);
1881                 sprintf(newpath64, "%s=%s:%s/%s/%s/%s:%s",
1882                     LD_SUPPORT_ENV_VAR_64, oldpath64, MAKE_PREFIX,
1883                     LD_SUPPORT_MAKE_LIB_DIR, LD_SUPPORT_MAKE_LIB_DIR_64,
1884                     LD_SUPPORT_MAKE_LIB, LD_SUPPORT_MAKE_LIB);
1847                 sprintf(newpath, "%s=%s", LD_SUPPORT_ENV_VAR, oldpath);
1885         }

1850 #if defined(TEAMWARE_MAKE_CMN)

1852         /* function maybe_append_str_to_env_var() is defined in avo_util library
1853          * Serial make should not use this library !!!
1854          */
1855         maybe_append_str_to_env_var(newpath, LD_SUPPORT_MAKE_LIB);
1856 #else
1857         if (oldpath == NULL) {
1858                 sprintf(newpath, "%s%s", newpath, LD_SUPPORT_MAKE_LIB);
1859         } else {
1860                 sprintf(newpath, "%s:%s", newpath, LD_SUPPORT_MAKE_LIB);
1861         }
1862 #endif
1887         putenv(newpath);
1888         if (prev_path) {
1889                 free(prev_path);
1890         }
1891         prev_path = newpath;

1893         putenv(newpath64);
1894         if (prev_path64) {
1895                 free(prev_path64);
1896         }
1897         prev_path64 = newpath64;
1898 #endif /* ! codereview */
1899 }

1901 /*
1902  *      read_files_and_state(argc, argv)
1903  *
1904  *      Read the makefiles we care about and the environment
1905  *      Also read the = style command line options
1906  *
1907  *      Parameters:
1908  *              argc            You know what this is
1909  *              argv            You know what this is
1910  *
1911  *      Static variables used:
1912  *              env_wins        make -e, determines if env vars are RO
1913  *              ignore_default_mk make -r, determines if make.rules is read
1914  *              not_auto_depen  dwight
1915  *
```



```
1916  *      Global variables used:
1917  *              default_target_to_build Set to first proper target from file
1918  *              do_not_exec_rule Set to false when makfile is made
1919  *              dot             The Name ".", used to read current dir
1920  *              empty_name      The Name "", use as macro value
1921  *              keep_state      Set if KEEP_STATE is in environment
1922  *              make_state      The Name ".make.state", used to read file
1923  *              makefile_type   Set to type of file being read
1924  *              makeflags       The Name "MAKEFLAGS", used to set macro value
1925  *              not_auto        dwight
1926  *              read_trace_level Checked to se if the reader should trace
1927  *              report_dependencies If -P is on we do not read .make.state
1928  *              trace_reader    Set if reader should trace
1929  *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
1930  */
1931 static void
1932 read_files_and_state(int argc, char **argv)
1933 {
1934         wchar_t                 buffer[1000];
1935         wchar_t                 buffer_posix[1000];
1936         register char           ch;
1937         register char           *cp;
1938         Property                def_make_macro = NULL;
1939         Name                    def_make_name;
1940         Name                    default_makefile;
1941         String_rec              dest;
1942         wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
1943         register int            i;
1944         register int            j;
1945         Name                    keep_state_name;
1946         int                     length;
1947         Name                    Makefile;
1948         register Property       macro;
1949         struct stat             make_state_stat;
1950         Name                    makefile_name;
1951         register int            makefile_next = 0;
1952         register Boolean        makefile_read = false;
1953         String_rec              makeflags_string;
1954         String_rec              makeflags_string_posix;
1955         String_rec *            makeflags_string_current;
1956         Name                    makeflags_value_saved;
1957         register Name           name;
1958         Name                    new_make_value;
1959         Boolean                 save_do_not_exec_rule;
1960         Name                    sdotMakefile;
1961         Name                    sdotmakefile_name;
1962         static wchar_t          state_file_str;
1963         static char             state_file_str_mb[MAXPATHLEN];
1964         static struct _Name     state_filename;
1965         Boolean                 temp;
1966         char                    tmp_char;
1967         wchar_t                 *tmp_wcs_buffer;
1968         register Name           value;
1969         ASCII_Dyn_Array         makeflags_and_macro;
1970         Boolean                 is_xpg4;

1972 /*
1973  *      Remember current mode. It may be changed after reading makefile
1974  *      and we will have to correct MAKEFLAGS variable.
1975  */
1976         is_xpg4 = posix;

1978         MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
1979         keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
1980         MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
1981         Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
```

```
1982         MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
1983         makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1984         MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
1985         sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
1986         MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
1987         sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

1989 /*
1990  *      Set flag if NSE is active
1991  */

1993 /*
1994  *      initialize global dependency entry for .NOT_AUTO
1995  */
1996         not_auto_depen->next = NULL;
1997         not_auto_depen->name = not_auto;
1998         not_auto_depen->automatic = not_auto_depen->stale = false;

2000 /*
2001  *      Read internal definitions and rules.
2002  */
2003         if (read_trace_level > 1) {
2004                 trace_reader = true;
2005         }
2006         if (!ignore_default_mk) {
2007                 if (svr4) {
2008                         MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
2009                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2010                 } else {
2011                         MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
2012                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2013                 }
2014                 default_makefile->stat.is_file = true;

2016                 (void) read_makefile(default_makefile,
2017                                      true,
2018                                      false,
2019                                      true);
2020         }

2022         /*
2023          * If the user did not redefine the MAKE macro in the
2024          * default makefile (make.rules), then we'd like to
2025          * change the macro value of MAKE to be some form
2026          * of argv[0] for recursive MAKE builds.
2027          */
2028         MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
2029         def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2030         def_make_macro = get_prop(def_make_name->prop, macro_prop);
2031         if ((def_make_macro != NULL) &&
2032             (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
2033                       NOCATGETS("make")))) {
2034                 MBSTOWCS(wcs_buffer, argv_zero_string);
2035                 new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2036                 (void) SETVAR(def_make_name,
2037                               new_make_value,
2038                               false);
2039         }

2041         default_target_to_build = NULL;
2042         trace_reader = false;

2044 /*
2045  *      Read environment args. Let file args which follow override unless
2046  *      -e option seen. If -e option is not mentioned.
2047  */
```



```
2048         read_environment(env_wins);
2049         if (getvar(virtual_root)->hash.length == 0) {
2050                 maybe_append_prop(virtual_root, macro_prop)
2051                   ->body.macro.exported = true;
2052                 MBSTOWCS(wcs_buffer, "/");
2053                 (void) SETVAR(virtual_root,
2054                               GETNAME(wcs_buffer, FIND_LENGTH),
2055                               false);
2056         }

2058 /*
2059  * We now scan mf_argv and argv to see if we need to set
2060  * any of the DMake-added options/variables in MAKEFLAGS.
2061  */

2063         makeflags_and_macro.start = 0;
2064         makeflags_and_macro.size = 0;
2065         enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
2066         enter_argv_values(argc, argv, &makeflags_and_macro);

2068 /*
2069  *      Set MFLAGS and MAKEFLAGS
2070  *
2071  *      Before reading makefile we do not know exactly which mode
2072  *      (posix or not) is used. So prepare two MAKEFLAGS strings
2073  *      for both posix and solaris modes because they are different.
2074  */
2075         INIT_STRING_FROM_STACK(makeflags_string, buffer);
2076         INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
2077         append_char((int) hyphen_char, &makeflags_string);
2078         append_char((int) hyphen_char, &makeflags_string_posix);

2080         switch (read_trace_level) {
2081         case 2:
2082                 append_char('D', &makeflags_string);
2083                 append_char('D', &makeflags_string_posix);
2084         case 1:
2085                 append_char('D', &makeflags_string);
2086                 append_char('D', &makeflags_string_posix);
2087         }
2088         switch (debug_level) {
2089         case 2:
2090                 append_char('d', &makeflags_string);
2091                 append_char('d', &makeflags_string_posix);
2092         case 1:
2093                 append_char('d', &makeflags_string);
2094                 append_char('d', &makeflags_string_posix);
2095         }
2096         if (env_wins) {
2097                 append_char('e', &makeflags_string);
2098                 append_char('e', &makeflags_string_posix);
2099         }
2100         if (ignore_errors_all) {
2101                 append_char('i', &makeflags_string);
2102                 append_char('i', &makeflags_string_posix);
2103         }
2104         if (continue_after_error) {
2105                 if (stop_after_error_ever_seen) {
2106                         append_char('S', &makeflags_string_posix);
2107                         append_char((int) space_char, &makeflags_string_posix);
2108                         append_char((int) hyphen_char, &makeflags_string_posix);
2109                 }
2110                 append_char('k', &makeflags_string);
2111                 append_char('k', &makeflags_string_posix);
2112         } else {
2113                 if (stop_after_error_ever_seen
```

```
2114                    && continue_after_error_ever_seen) {
2115                        append_char('k', &makeflags_string_posix);
2116                        append_char((int) space_char, &makeflags_string_posix);
2117                        append_char((int) hyphen_char, &makeflags_string_posix);
2118                        append_char('S', &makeflags_string_posix);
2119                }
2120        }
2121        if (do_not_exec_rule) {
2122                append_char('n', &makeflags_string);
2123                append_char('n', &makeflags_string_posix);
2124        }
2125        switch (report_dependencies_level) {
2126        case 4:
2127                append_char('P', &makeflags_string);
2128                append_char('P', &makeflags_string_posix);
2129        case 3:
2130                append_char('P', &makeflags_string);
2131                append_char('P', &makeflags_string_posix);
2132        case 2:
2133                append_char('P', &makeflags_string);
2134                append_char('P', &makeflags_string_posix);
2135        case 1:
2136                append_char('P', &makeflags_string);
2137                append_char('P', &makeflags_string_posix);
2138        }
2139        if (trace_status) {
2140                append_char('p', &makeflags_string);
2141                append_char('p', &makeflags_string_posix);
2142        }
2143        if (quest) {
2144                append_char('q', &makeflags_string);
2145                append_char('q', &makeflags_string_posix);
2146        }
2147        if (silent_all) {
2148                append_char('s', &makeflags_string);
2149                append_char('s', &makeflags_string_posix);
2150        }
2151        if (touch) {
2152                append_char('t', &makeflags_string);
2153                append_char('t', &makeflags_string_posix);
2154        }
2155        if (build_unconditional) {
2156                append_char('u', &makeflags_string);
2157                append_char('u', &makeflags_string_posix);
2158        }
2159        if (report_cwd) {
2160                append_char('w', &makeflags_string);
2161                append_char('w', &makeflags_string_posix);
2162        }
2163        /* -c dmake_rcfile */
2164        if (dmake_rcfile_specified) {
2165                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2166                dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
2167                append_makeflags_string(dmake_rcfile, &makeflags_string);
2168                append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
2169        }
2170        /* -g dmake_group */
2171        if (dmake_group_specified) {
2172                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2173                dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
2174                append_makeflags_string(dmake_group, &makeflags_string);
2175                append_makeflags_string(dmake_group, &makeflags_string_posix);
2176        }
2177        /* -j dmake_max_jobs */
2178        if (dmake_max_jobs_specified) {
2179                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
```



```
2180                dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
2181                append_makeflags_string(dmake_max_jobs, &makeflags_string);
2182                append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
2183        }
2184        /* -m dmake_mode */
2185        if (dmake_mode_specified) {
2186                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2187                dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2188                append_makeflags_string(dmake_mode, &makeflags_string);
2189                append_makeflags_string(dmake_mode, &makeflags_string_posix);
2190        }
2191        /* -x dmake_compat_mode */
2192 //     if (dmake_compat_mode_specified) {
2193 //             MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2194 //             dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2195 //             append_makeflags_string(dmake_compat_mode, &makeflags_string);
2196 //             append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
2197 //     }
2198        /* -x dmake_output_mode */
2199        if (dmake_output_mode_specified) {
2200                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2201                dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2202                append_makeflags_string(dmake_output_mode, &makeflags_string);
2203                append_makeflags_string(dmake_output_mode, &makeflags_string_pos
2204        }
2205        /* -o dmake_odir */
2206        if (dmake_odir_specified) {
2207                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2208                dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2209                append_makeflags_string(dmake_odir, &makeflags_string);
2210                append_makeflags_string(dmake_odir, &makeflags_string_posix);
2211        }
2212        /* -M pmake_machinesfile */
2213        if (pmake_machinesfile_specified) {
2214                MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2215                pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2216                append_makeflags_string(pmake_machinesfile, &makeflags_string);
2217                append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2218        }
2219        /* -R */
2220        if (pmake_cap_r_specified) {
2221                append_char((int) space_char, &makeflags_string);
2222                append_char((int) hyphen_char, &makeflags_string);
2223                append_char('R', &makeflags_string);
2224                append_char((int) space_char, &makeflags_string_posix);
2225                append_char((int) hyphen_char, &makeflags_string_posix);
2226                append_char('R', &makeflags_string_posix);
2227        }

2229 /*
2230  *      Make sure MAKEFLAGS is exported
2231  */
2232        maybe_append_prop(makeflags, macro_prop)->
2233          body.macro.exported = true;

2235        if (makeflags_string.buffer.start[1] != (int) nul_char) {
2236                if (makeflags_string.buffer.start[1] != (int) space_char) {
2237                        MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2238                        (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2239                                      GETNAME(makeflags_string.buffer.start,
2240                                              FIND_LENGTH),
2241                                      false);
2242                } else {
2243                        MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2244                        (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2245                                      GETNAME(makeflags_string.buffer.start + 2,
```

```
2246                                            FIND_LENGTH),
2247                                    false);
2248                }
2249        }

2251 /*
2252  *      Add command line macro to POSIX makeflags_string
2253  */
2254        if (makeflags_and_macro.start) {
2255                tmp_char = (char) space_char;
2256                cp = makeflags_and_macro.start;
2257                do {
2258                        append_char(tmp_char, &makeflags_string_posix);
2259                } while ( tmp_char = *cp++ );
2260                retmem_mb(makeflags_and_macro.start);
2261        }

2263 /*
2264  *      Now set the value of MAKEFLAGS macro in accordance
2265  *      with current mode.
2266  */
2267        macro = maybe_append_prop(makeflags, macro_prop);
2268        temp = (Boolean) macro->body.macro.read_only;
2269        macro->body.macro.read_only = false;
2270        if(posix || gnu_style) {
2271                makeflags_string_current = &makeflags_string_posix;
2272        } else {
2273                makeflags_string_current = &makeflags_string;
2274        }
2275        if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2276                makeflags_value_saved =
2277                        GETNAME( makeflags_string_current->buffer.start + 1
2278                               , FIND_LENGTH
2279                               );
2280        } else {
2281                if (makeflags_string_current->buffer.start[1] != (int) space_cha
2282                        makeflags_value_saved =
2283                                GETNAME( makeflags_string_current->buffer.start
2284                                       , FIND_LENGTH
2285                                       );
2286                } else {
2287                        makeflags_value_saved =
2288                                GETNAME( makeflags_string_current->buffer.start
2289                                       , FIND_LENGTH
2290                                       );
2291                }
2292        }
2293        (void) SETVAR( makeflags
2294                     , makeflags_value_saved
2295                     , false
2296                     );
2297        macro->body.macro.read_only = temp;

2299 /*
2300  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2301  */
2302        save_do_not_exec_rule = do_not_exec_rule;
2303        do_not_exec_rule = false;
2304        if (read_trace_level > 0) {
2305                trace_reader = true;
2306        }

2308        for (i = 1; i < argc; i++) {
2309                if (argv[i] &&
2310                    (argv[i][0] == (int) hyphen_char) &&
2311                    (argv[i][1] == 'f') &&
```



```
2312                    (argv[i][2] == (int) nul_char)) {
2313                        argv[i] = NULL;         /* Remove -f */
2314                        if (i >= argc - 1) {
2315                                fatal(catgets(catd, 1, 190, "No filename argumen
2316                        }
2317                        MBSTOWCS(wcs_buffer, argv[++i]);
2318                        primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2319                        (void) read_makefile(primary_makefile, true, true, true)
2320                        argv[i] = NULL;         /* Remove filename */
2321                        makefile_read = true;
2322                } else if (argv[i] &&
2323                           (argv[i][0] == (int) hyphen_char) &&
2324                           (argv[i][1] == 'c' ||
2325                            argv[i][1] == 'g' ||
2326                            argv[i][1] == 'j' ||
2327                            argv[i][1] == 'K' ||
2328                            argv[i][1] == 'M' ||
2329                            argv[i][1] == 'm' ||
2330                            argv[i][1] == 'O' ||
2331                            argv[i][1] == 'o') &&
2332                           (argv[i][2] == (int) nul_char)) {
2333                        argv[i] = NULL;
2334                        argv[++i] = NULL;
2335                }
2336        }

2338 /*
2339  *      If no command line "-f" args then look for "makefile", and then for
2340  *      "Makefile" if "makefile" isn't found.
2341  */
2342        if (!makefile_read) {
2343                (void) read_dir(dot,
2344                                (wchar_t *) NULL,
2345                                (Property) NULL,
2346                                (wchar_t *) NULL);
2347            if (!posix) {
2348                if (makefile_name->stat.is_file) {
2349                        if (Makefile->stat.is_file) {
2350                                warning(catgets(catd, 1, 310, "Both 'makefile' a
2351                        }
2352                        primary_makefile = makefile_name;
2353                        makefile_read = read_makefile(makefile_name,
2354                                                      false,
2355                                                      false,
2356                                                      true);
2357                }
2358                if (!makefile_read &&
2359                    Makefile->stat.is_file) {
2360                        primary_makefile = Makefile;
2361                        makefile_read = read_makefile(Makefile,
2362                                                      false,
2363                                                      false,
2364                                                      true);
2365                }
2366            } else {

2368                enum sccs_stat save_m_has_sccs = NO_SCCS;
2369                enum sccs_stat save_M_has_sccs = NO_SCCS;

2371                if (makefile_name->stat.is_file) {
2372                        if (Makefile->stat.is_file) {
2373                                warning(catgets(catd, 1, 191, "Both 'makefile' a
2374                        }
2375                }
2376                if (makefile_name->stat.is_file) {
2377                        if (makefile_name->stat.has_sccs == NO_SCCS) {
```

```
2378                            primary_makefile = makefile_name;
2379                            makefile_read = read_makefile(makefile_name,
2380                                                          false,
2381                                                          false,
2382                                                          true);
2383                          } else {
2384                            save_m_has_sccs = makefile_name->stat.has_sccs;
2385                            makefile_name->stat.has_sccs = NO_SCCS;
2386                            primary_makefile = makefile_name;
2387                            makefile_read = read_makefile(makefile_name,
2388                                                          false,
2389                                                          false,
2390                                                          true);
2391                          }
2392                    }
2393                    if (!makefile_read &&
2394                        Makefile->stat.is_file) {
2395                            if (Makefile->stat.has_sccs == NO_SCCS) {
2396                               primary_makefile = Makefile;
2397                               makefile_read = read_makefile(Makefile,
2398                                                             false,
2399                                                             false,
2400                                                             true);
2401                            } else {
2402                              save_M_has_sccs = Makefile->stat.has_sccs;
2403                              Makefile->stat.has_sccs = NO_SCCS;
2404                              primary_makefile = Makefile;
2405                              makefile_read = read_makefile(Makefile,
2406                                                            false,
2407                                                            false,
2408                                                            true);
2409                            }
2410                    }
2411                    if (!makefile_read &&
2412                          makefile_name->stat.is_file) {
2413                             makefile_name->stat.has_sccs = save_m_has_sccs;
2414                             primary_makefile = makefile_name;
2415                             makefile_read = read_makefile(makefile_name,
2416                                                           false,
2417                                                           false,
2418                                                           true);
2419                    }
2420                    if (!makefile_read &&
2421                        Makefile->stat.is_file) {
2422                             Makefile->stat.has_sccs = save_M_has_sccs;
2423                             primary_makefile = Makefile;
2424                             makefile_read = read_makefile(Makefile,
2425                                                           false,
2426                                                           false,
2427                                                           true);
2428                    }
2429                }
2430            }
2431            do_not_exec_rule = save_do_not_exec_rule;
2432            allrules_read = makefile_read;
2433            trace_reader = false;

2435     /*
2436      *      Now get current value of MAKEFLAGS and compare it with
2437      *      the saved value we set before reading makefile.
2438      *      If they are different then MAKEFLAGS is subsequently set by
2439      *      makefile, just leave it there. Otherwise, if make mode
2440      *      is changed by using .POSIX target in makefile we need
2441      *      to correct MAKEFLAGS value.
2442      */
2443            Name mf_val = getvar(makeflags);
```



```
2444            if( (posix != is_xpg4)
2445             && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2446            {
2447                    if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2448                            (void) SETVAR(makeflags,
2449                                          GETNAME(makeflags_string_posix.buffer.star
2450                                                  FIND_LENGTH),
2451                                          false);
2452                    } else {
2453                            if (makeflags_string_posix.buffer.start[1] != (int) spac
2454                                    (void) SETVAR(makeflags,
2455                                                  GETNAME(makeflags_string_posix.buf
2456                                                          FIND_LENGTH),
2457                                                  false);
2458                            } else {
2459                                    (void) SETVAR(makeflags,
2460                                                  GETNAME(makeflags_string_posix.buf
2461                                                          FIND_LENGTH),
2462                                                  false);
2463                            }
2464                    }
2465            }

2467            if (makeflags_string.free_after_use) {
2468                    retmem(makeflags_string.buffer.start);
2469            }
2470            if (makeflags_string_posix.free_after_use) {
2471                    retmem(makeflags_string_posix.buffer.start);
2472            }
2473            makeflags_string.buffer.start = NULL;
2474            makeflags_string_posix.buffer.start = NULL;

2476            if (posix) {
2477                    /*
2478                     * If the user did not redefine the ARFLAGS macro in the
2479                     * default makefile (make.rules), then we'd like to
2480                     * change the macro value of ARFLAGS to be in accordance
2481                     * with "POSIX" requirements.
2482                     */
2483                    MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2484                    name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2485                    macro = get_prop(name->prop, macro_prop);
2486                    if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2487                        (IS_EQUAL(macro->body.macro.value->string_mb,
2488                                  NOCATGETS("rv")))) {
2489                            MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2490                            value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2491                            (void) SETVAR(name,
2492                                          value,
2493                                          false);
2494                    }
2495            }

2497            if (!posix && !svr4) {
2498                    set_sgs_support();
2499            }


2502     /*
2503      *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2504      */
2505            macro = get_prop(keep_state_name->prop, macro_prop);
2506            if ((macro != NULL) &&
2507                macro->body.macro.exported) {
2508                    keep_state = true;
2509            }
```

```
2510         if (keep_state) {
2511                 if (macro == NULL) {
2512                         macro = maybe_append_prop(keep_state_name,
2513                                                   macro_prop);
2514                 }
2515                 macro->body.macro.exported = true;
2516                 (void) SETVAR(keep_state_name,
2517                               empty_name,
2518                               false);

2520                 /*
2521                  *      Read state file
2522                  */

2524                 /* Before we read state, let's make sure we have
2525                 ** right state file.
2526                 */
2527                 /* just in case macro references are used in make_state file
2528                 ** name, we better expand them at this stage using expand_value.
2529                 */
2530                 INIT_STRING_FROM_STACK(dest, destbuffer);
2531                 expand_value(make_state, &dest, false);

2533                 make_state = GETNAME(dest.buffer.start, FIND_LENGTH);

2535                 if(!stat(make_state->string_mb, &make_state_stat)) {
2536                    if(!(make_state_stat.st_mode & S_IFREG) ) {
2537                         /* copy the make_state structure to the other
2538                         ** and then let make_state point to the new
2539                         ** one.
2540                         */
2541                        memcpy(&state_filename, make_state,sizeof(state_filename))
2542                        state_filename.string_mb = state_file_str_mb;
2543                 /* Just a kludge to avoid two slashes back to back */
2544                        if((make_state->hash.length == 1)&&
2545                                (make_state->string_mb[0] == '/')) {
2546                           make_state->hash.length = 0;
2547                           make_state->string_mb[0] = '\0';
2548                        }
2549                        sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2550                         make_state->string_mb,NOCATGETS("/.make.state"));
2551                        make_state = &state_filename;
2552                          /* adjust the length to reflect the appended string */
2553                        make_state->hash.length += 12;
2554                    }
2555                 } else { /* the file doesn't exist or no permission */
2556                    char tmp_path[MAXPATHLEN];
2557                    char *slashp;

2559                    if (slashp = strrchr(make_state->string_mb, '/')) {
2560                       strncpy(tmp_path, make_state->string_mb,
2561                                (slashp - make_state->string_mb));
2562                         tmp_path[slashp - make_state->string_mb]=0;
2563                       if(strlen(tmp_path)) {
2564                         if(stat(tmp_path, &make_state_stat)) {
2565                           warning(catgets(catd, 1, 192, "directory %s for .KEEP_
2566                         }
2567                         if (access(tmp_path, F_OK) != 0) {
2568                           warning(catgets(catd, 1, 193, "can't access dir %s"),t
2569                         }
2570                       }
2571                    }
2572                 }
2573                 if (report_dependencies_level != 1) {
2574                         Makefile_type   makefile_type_temp = makefile_type;
2575                         makefile_type = reading_statefile;
```



```
2576                         if (read_trace_level > 1) {
2577                                 trace_reader = true;
2578                         }
2579                         (void) read_simple_file(make_state,
2580                                                 false,
2581                                                 false,
2582                                                 false,
2583                                                 false,
2584                                                 false,
2585                                                 true);
2586                         trace_reader = false;
2587                         makefile_type = makefile_type_temp;
2588                 }
2589         }
2590 }

2592 /*
2593  * Scan the argv for options and "=" type args and make them readonly.
2594  */
2595 static void
2596 enter_argv_values(int argc, char *argv[], ASCII_Dyn_Array *makeflags_and_macro)
2597 {
2598         register char           *cp;
2599         register int            i;
2600         int                     length;
2601         register Name           name;
2602         int                     opt_separator = argc;
2603         char                    tmp_char;
2604         wchar_t                 *tmp_wcs_buffer;
2605         register Name           value;
2606         Boolean                 append = false;
2607         Property                macro;
2608         struct stat             statbuf;


2611         /* Read argv options and "=" type args and make them readonly. */
2612         makefile_type = reading_nothing;
2613         for (i = 1; i < argc; ++i) {
2614                 append = false;
2615                 if (argv[i] == NULL) {
2616                         continue;
2617                 } else if (((argv[i][0] == '-') && (argv[i][1] == '-')) ||
2618                            ((argv[i][0] == (int) ' ') &&
2619                             (argv[i][1] == (int) '-') &&
2620                             (argv[i][2] == (int) ' ') &&
2621                             (argv[i][3] == (int) '-'))) {
2622                         argv[i] = NULL;
2623                         opt_separator = i;
2624                         continue;
2625                 } else if ((i < opt_separator) && (argv[i][0] == (int) hyphen_ch
2626                         switch (parse_command_option(argv[i][1])) {
2627                         case 1: /* -f seen */
2628                                 ++i;
2629                                 continue;
2630                         case 2: /* -c seen */
2631                                 if (argv[i+1] == NULL) {
2632                                         fatal(catgets(catd, 1, 194, "No dmake rc
2633                                 }
2634                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2635                                 name = GETNAME(wcs_buffer, FIND_LENGTH);
2636                                 break;
2637                         case 4: /* -g seen */
2638                                 if (argv[i+1] == NULL) {
2639                                         fatal(catgets(catd, 1, 195, "No dmake gr
2640                                 }
2641                                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
```

```
2642                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2643                                    break;
2644                            case 8: /* -j seen */
2645                                    if (argv[i+1] == NULL) {
2646                                            fatal(catgets(catd, 1, 196, "No dmake ma
2647                                    }
2648                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS")
2649                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2650                                    break;
2651                            case 16: /* -M seen */
2652                                    if (argv[i+1] == NULL) {
2653                                            fatal(catgets(catd, 1, 323, "No pmake ma
2654                                    }
2655                                    MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFI
2656                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2657                                    break;
2658                            case 32: /* -m seen */
2659                                    if (argv[i+1] == NULL) {
2660                                            fatal(catgets(catd, 1, 197, "No dmake mo
2661                                    }
2662                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2663                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2664                                    break;
2665                            case 128: /* -O seen */
2666                                    if (argv[i+1] == NULL) {
2667                                            fatal(catgets(catd, 1, 287, "No file des
2668                                    }
2669                                    mtool_msgs_fd = atoi(argv[i+1]);
2670                                    /* find out if mtool_msgs_fd is a valid file des
2671                                    if (fstat(mtool_msgs_fd, &statbuf) < 0) {
2672                                            fatal(catgets(catd, 1, 355, "Invalid fil
2673                                    }
2674                                    argv[i] = NULL;
2675                                    argv[i+1] = NULL;
2676                                    continue;
2677                            case 256: /* -K seen */
2678                                    if (argv[i+1] == NULL) {
2679                                            fatal(catgets(catd, 1, 288, "No makestat
2680                                    }
2681                                    MBSTOWCS(wcs_buffer, argv[i+1]);
2682                                    make_state = GETNAME(wcs_buffer, FIND_LENGTH);
2683                                    keep_state = true;
2684                                    argv[i] = NULL;
2685                                    argv[i+1] = NULL;
2686                                    continue;
2687                            case 512:       /* -o seen */
2688                                    if (argv[i+1] == NULL) {
2689                                            fatal(catgets(catd, 1, 312, "No dmake ou
2690                                    }
2691                                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2692                                    name = GETNAME(wcs_buffer, FIND_LENGTH);
2693                                    break;
2694                            case 1024: /* -x seen */
2695                                    if (argv[i+1] == NULL) {
2696                                            fatal(catgets(catd, 1, 351, "No argument
2697                                    }
2698                                    length = strlen( NOCATGETS("SUN_MAKE_COMPAT_MODE
2699                                    if (strncmp(argv[i+1], NOCATGETS("SUN_MAKE_COMPA
2700                                            argv[i+1] = &argv[i+1][length];
2701                                            MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE
2702                                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2703                                            dmake_compat_mode_specified = dmake_add_
2704                                            break;
2705                                    }
2706                                    length = strlen( NOCATGETS("DMAKE_OUTPUT_MODE=")
2707                                    if (strncmp(argv[i+1], NOCATGETS("DMAKE_OUTPUT_M
```



```
2708                                            argv[i+1] = &argv[i+1][length];
2709                                            MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OU
2710                                            name = GETNAME(wcs_buffer, FIND_LENGTH);
2711                                            dmake_output_mode_specified = dmake_add_
2712                                    } else {
2713                                            warning(catgets(catd, 1, 354, "Unknown a
2714                                                  argv[i+1]);
2715                                            argv[i] = argv[i + 1] = NULL;
2716                                            continue;
2717                                    }
2718                                    break;
2719                            default: /* Shouldn't reach here */
2720                                    argv[i] = NULL;
2721                                    continue;
2722                            }
2723                            argv[i] = NULL;
2724                            if (i == (argc - 1)) {
2725                                    break;
2726                            }
2727                            if ((length = strlen(argv[i+1])) >= MAXPATHLEN) {
2728                                    tmp_wcs_buffer = ALLOC_WC(length + 1);
2729                                    (void) mbstowcs(tmp_wcs_buffer, argv[i+1], lengt
2730                                    value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2731                                    retmem(tmp_wcs_buffer);
2732                            } else {
2733                                    MBSTOWCS(wcs_buffer, argv[i+1]);
2734                                    value = GETNAME(wcs_buffer, FIND_LENGTH);
2735                            }
2736                            argv[i+1] = NULL;
2737                    } else if ((cp = strchr(argv[i], (int) equal_char)) != NULL) {
2738 /*
2739  * Combine all macro in dynamic array
2740  */
2741                            if(*(cp-1) == (int) plus_char)
2742                            {
2743                                    if(isspace(*(cp-2))) {
2744                                            append = true;
2745                                            cp--;
2746                                    }
2747                            }
2748                            if(!append)
2749                                    append_or_replace_macro_in_dyn_array(makeflags_a
2750
2751                            while (isspace(*(cp-1))) {
2752                                    cp--;
2753                            }
2754                            tmp_char = *cp;
2755                            *cp = (int) nul_char;
2756                            MBSTOWCS(wcs_buffer, argv[i]);
2757                            *cp = tmp_char;
2758                            name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2759                            while (*cp != (int) equal_char) {
2760                                    cp++;
2761                            }
2762                            cp++;
2763                            while (isspace(*cp) && (*cp != (int) nul_char)) {
2764                                    cp++;
2765                            }
2766                            if ((length = strlen(cp)) >= MAXPATHLEN) {
2767                                    tmp_wcs_buffer = ALLOC_WC(length + 1);
2768                                    (void) mbstowcs(tmp_wcs_buffer, cp, length + 1);
2769                                    value = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2770                                    retmem(tmp_wcs_buffer);
2771                            } else {
2772                                    MBSTOWCS(wcs_buffer, cp);
2773                                    value = GETNAME(wcs_buffer, FIND_LENGTH);
```

```
2774                        }
2775                        argv[i] = NULL;
2776                } else {
2777                        /* Illegal MAKEFLAGS argument */
2778                        continue;
2779                }
2780                if(append) {
2781                        setvar_append(name, value);
2782                        append = false;
2783                } else {
2784                        macro = maybe_append_prop(name, macro_prop);
2785                        macro->body.macro.exported = true;
2786                        SETVAR(name, value, false)->body.macro.read_only = true;
2787                }
2788        }
2789 }

2791 /*
2792  * Append the DMake option and value to the MAKEFLAGS string.
2793  */
2794 static void
2795 append_makeflags_string(Name name, register String makeflags_string)
2796 {
2797         const char      *option;

2799         if (strcmp(name->string_mb, NOCATGETS("DMAKE_GROUP")) == 0) {
2800                 option = NOCATGETS(" -g ");
2801         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MAX_JOBS")) == 0) {
2802                 option = NOCATGETS(" -j ");
2803         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_MODE")) == 0) {
2804                 option = NOCATGETS(" -m ");
2805         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_ODIR")) == 0) {
2806                 option = NOCATGETS(" -o ");
2807         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_RCFILE")) == 0) {
2808                 option = NOCATGETS(" -c ");
2809         } else if (strcmp(name->string_mb, NOCATGETS("PMAKE_MACHINESFILE")) == 0
2810                 option = NOCATGETS(" -M ");
2811         } else if (strcmp(name->string_mb, NOCATGETS("DMAKE_OUTPUT_MODE")) == 0)
2812                 option = NOCATGETS(" -x DMAKE_OUTPUT_MODE=");
2813         } else if (strcmp(name->string_mb, NOCATGETS("SUN_MAKE_COMPAT_MODE")) ==
2814                 option = NOCATGETS(" -x SUN_MAKE_COMPAT_MODE=");
2815         } else {
2816                 fatal(catgets(catd, 1, 289, "Internal error: name not recognized
2817         }
2818         Property prop = maybe_append_prop(name, macro_prop);
2819         if( prop == 0 || prop->body.macro.value == 0 ||
2820             prop->body.macro.value->string_mb == 0 ) {
2821                 return;
2822         }
2823         char mbs_value[MAXPATHLEN + 100];
2824         strcpy(mbs_value, option);
2825         strcat(mbs_value, prop->body.macro.value->string_mb);
2826         MBSTOWCS(wcs_buffer, mbs_value);
2827         append_string(wcs_buffer, makeflags_string, FIND_LENGTH);
2828 }

2830 /*
2831  *      read_environment(read_only)
2832  *
2833  *      This routine reads the process environment when make starts and enters
2834  *      it as make macros. The environment variable SHELL is ignored.
2835  *
2836  *      Parameters:
2837  *              read_only       Should we make env vars read only?
2838  *
2839  *      Global variables used:
```



```
2840  *              report_pwd      Set if this make was started by other make
2841  */
2842 static void
2843 read_environment(Boolean read_only)
2844 {
2845         register char           **environment;
2846         int                     length;
2847         wchar_t                 *tmp_wcs_buffer;
2848         Boolean                 alloced_tmp_wcs_buffer = false;
2849         register wchar_t        *name;
2850         register wchar_t        *value;
2851         register Name           macro;
2852         Property                val;
2853         Boolean                 read_only_saved;

2855         reading_environment = true;
2856         environment = environ;
2857         for (; *environment; environment++) {
2858                 read_only_saved = read_only;
2859                 if ((length = strlen(*environment)) >= MAXPATHLEN) {
2860                         tmp_wcs_buffer = ALLOC_WC(length + 1);
2861                         alloced_tmp_wcs_buffer = true;
2862                         (void) mbstowcs(tmp_wcs_buffer, *environment, length + 1
2863                         name = tmp_wcs_buffer;
2864                 } else {
2865                         MBSTOWCS(wcs_buffer, *environment);
2866                         name = wcs_buffer;
2867                 }
2868                 value = (wchar_t *) wschr(name, (int) equal_char);

2870                 /*
2871                  * Looks like there's a bug in the system, but sometimes
2872                  * you can get blank lines in *environment.
2873                  */
2874                 if (!value) {
2875                         continue;
2876                 }
2877                 MBSTOWCS(wcs_buffer2, NOCATGETS("SHELL="));
2878                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2879                         continue;
2880                 }
2881                 MBSTOWCS(wcs_buffer2, NOCATGETS("MAKEFLAGS="));
2882                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
2883                         report_pwd = true;
2884                         /*
2885                          * In POSIX mode we do not want MAKEFLAGS to be readonly
2886                          * If the MAKEFLAGS macro is subsequently set by the mak
2887                          * it replaces the MAKEFLAGS variable currently found in
2888                          * environment.
2889                          * See Assertion 50 in section 6.2.5.3 of standard P1003
2890                          */
2891                         if(posix) {
2892                                 read_only_saved = false;
2893                         }
2894                 }

2896                 /*
2897                  * We ignore SUNPRO_DEPENDENCIES. This environment variable is
2898                  * set by make and read by cpp which then writes info to
2899                  * .make.dependency.xxx.  When make is invoked by another make
2900                  * (recursive make), we don't want to read this because then
2901                  * the child make will end up writing to the parent
2902                  * directory's .make.state and clobbering them.
2903                  */
2904                 MBSTOWCS(wcs_buffer2, NOCATGETS("SUNPRO_DEPENDENCIES"));
2905                 if (IS_WEQUALN(name, wcs_buffer2, wslen(wcs_buffer2))) {
```

```
2906                            continue;
2907                    }
2908
2909                    macro = GETNAME(name, value - name);
2910                    maybe_append_prop(macro, macro_prop)->body.macro.exported =
2911                      true;
2912                    if ((value == NULL) || ((value + 1)[0] == (int) nul_char)) {
2913                            val = setvar_daemon(macro,
2914                                                (Name) NULL,
2915                                                false, no_daemon, false, debug_level
2916                    } else {
2917                            val = setvar_daemon(macro,
2918                                                GETNAME(value + 1, FIND_LENGTH),
2919                                                false, no_daemon, false, debug_level
2920                    }
2921                    val->body.macro.read_only = read_only_saved;
2922                    if (alloced_tmp_wcs_buffer) {
2923                            retmem(tmp_wcs_buffer);
2924                            alloced_tmp_wcs_buffer = false;
2925                    }
2926            }
2927            reading_environment = false;
2928    }
2929
2930    /*
2931     *      read_makefile(makefile, complain, must_exist, report_file)
2932     *
2933     *      Read one makefile and check the result
2934     *
2935     *      Return value:
2936     *                              false is the read failed
2937     *
2938     *      Parameters:
2939     *              makefile        The file to read
2940     *              complain        Passed thru to read_simple_file()
2941     *              must_exist      Passed thru to read_simple_file()
2942     *              report_file     Passed thru to read_simple_file()
2943     *
2944     *      Global variables used:
2945     *              makefile_type   Set to indicate we are reading main file
2946     *              recursion_level Initialized
2947     */
2948    static Boolean
2949    read_makefile(register Name makefile, Boolean complain, Boolean must_exist, Bool
2950    {
2951            Boolean                 b;
2952
2953            makefile_type = reading_makefile;
2954            recursion_level = 0;
2955            reading_dependencies = true;
2956            b = read_simple_file(makefile, true, true, complain,
2957                                 must_exist, report_file, false);
2958            reading_dependencies = false;
2959            return b;
2960    }
2961
2962    /*
2963     *      make_targets(argc, argv, parallel_flag)
2964     *
2965     *      Call doname on the specified targets
2966     *
2967     *      Parameters:
2968     *              argc            You know what this is
2969     *              argv            You know what this is
2970     *              parallel_flag   True if building in parallel
2971     *
```



```
2972     *      Global variables used:
2973     *              build_failed_seen Used to generated message after failed -k
2974     *              commands_done   Used to generate message "Up to date"
2975     *              default_target_to_build First proper target in makefile
2976     *              init            The Name ".INIT", use to run command
2977     *              parallel        Global parallel building flag
2978     *              quest           make -q, suppresses messages
2979     *              recursion_level Initialized, used for tracing
2980     *              report_dependencies make -P, regroves whole process
2981     */
2982    static void
2983    make_targets(int argc, char **argv, Boolean parallel_flag)
2984    {
2985            int                     i;
2986            char                    *cp;
2987            Doname                  result;
2988            register Boolean        target_to_make_found = false;
2989
2990            (void) doname(init, true, true);
2991            recursion_level = 1;
2992            parallel = parallel_flag;
2993    /*
2994     *      make remaining args
2995     */
2996    #ifdef TEAMWARE_MAKE_CMN
2997    /*
2998            if ((report_dependencies_level == 0) && parallel) {
2999     */
3000            if (parallel) {
3001                    /*
3002                     * If building targets in parallel, start all of the
3003                     * remaining args to build in parallel.
3004                     */
3005                    for (i = 1; i < argc; i++) {
3006                            if ((cp = argv[i]) != NULL) {
3007                                    commands_done = false;
3008                                    if ((cp[0] == (int) period_char) &&
3009                                        (cp[1] == (int) slash_char)) {
3010                                            cp += 2;
3011                                    }
3012                                     if((cp[0] == (int) ' ') &&
3013                                        (cp[1] == (int) '-') &&
3014                                        (cp[2] == (int) ' ') &&
3015                                        (cp[3] == (int) '-')) {
3016                                        argv[i] = NULL;
3017                                            continue;
3018                                    }
3019                                    MBSTOWCS(wcs_buffer, cp);
3020                                    //default_target_to_build = GETNAME(wcs_buffer,
3021                                    //                                  FIND_LENGTH);
3022                                    default_target_to_build = normalize_name(wcs_buf
3023                                                                             wslen(wcs_buff
3024                                    if (default_target_to_build == wait_name) {
3025                                            if (parallel_process_cnt > 0) {
3026                                                    finish_running();
3027                                            }
3028                                            continue;
3029                                    }
3030                                    top_level_target = get_wstring(default_target_to
3031                                    /*
3032                                     * If we can't execute the current target in
3033                                     * parallel, hold off the target processing
3034                                     * to preserve the order of the targets as they
3035                                     * in command line.
3036                                     */
3037                                    if (!parallel_ok(default_target_to_build, false)
```

```
3038                                            && parallel_process_cnt > 0) {
3039                                    finish_running();
3040                            }
3041                            result = doname_check(default_target_to_build,
3042                                                  true,
3043                                                  false,
3044                                                  false);
3045                            gather_recursive_deps();
3046                            if (/* !commands_done && */
3047                                (result == build_ok) &&
3048                                !quest &&
3049                                (report_dependencies_level == 0) /*  &&
3050                                (exists(default_target_to_build) > file_does
3051                                    if (posix) {
3052                                            if (!commands_done) {
3053                                                    (void) printf(catgets(ca
3054                                                                          default_ta
3055                                            } else {
3056                                                    if (no_action_was_taken)
3057                                                            (void) printf(ca
3058                                                                          de
3059                                                    }
3060                                            }
3061                                    } else {
3062                                            default_target_to_build->stat.ti
3063                                            if (!commands_done &&
3064                                                (exists(default_target_to_bu
3065                                                    (void) printf(catgets(ca
3066                                                                          default_ta
3067                                            }
3068                                    }
3069                            }
3070                    }
3071                    }
3072                    /* Now wait for all of the targets to finish running */
3073                    finish_running();
3074                    //              setjmp(jmpbuffer);
3075
3076            }
3077 #endif
3078            for (i = 1; i < argc; i++) {
3079                    if ((cp = argv[i]) != NULL) {
3080                            target_to_make_found = true;
3081                            if ((cp[0] == (int) period_char) &&
3082                                (cp[1] == (int) slash_char)) {
3083                                    cp += 2;
3084                            }
3085                                     if((cp[0] == (int) ' ') &&
3086                                        (cp[1] == (int) '-') &&
3087                                        (cp[2] == (int) ' ') &&
3088                                        (cp[3] == (int) '-')) {
3089                                        argv[i] = NULL;
3090                                            continue;
3091                                    }
3092                            MBSTOWCS(wcs_buffer, cp);
3093                            default_target_to_build = normalize_name(wcs_buffer, wsl
3094                            top_level_target = get_wstring(default_target_to_build->
3095                            report_recursion(default_target_to_build);
3096                            commands_done = false;
3097                            if (parallel) {
3098                                    result = (Doname) default_target_to_build->state
3099                            } else {
3100                                    result = doname_check(default_target_to_build,
3101                                                          true,
3102                                                          false,
3103                                                          false);
```



```
3104                            }
3105                            gather_recursive_deps();
3106                            if (build_failed_seen) {
3107                                    build_failed_ever_seen = true;
3108                                    warning(catgets(catd, 1, 200, "Target '%s' not r
3109                                            default_target_to_build->string_mb);
3110                            }
3111                            build_failed_seen = false;
3112                            if (report_dependencies_level > 0) {
3113                                    print_dependencies(default_target_to_build,
3114                                                       get_prop(default_target_to_bu
3115                                                                line_prop));
3116                            }
3117                            default_target_to_build->stat.time =
3118                              file_no_time;
3119                            if (default_target_to_build->colon_splits > 0) {
3120                                    default_target_to_build->state =
3121                                      build_dont_know;
3122                            }
3123                            if (!parallel &&
3124                                /* !commands_done && */
3125                                (result == build_ok) &&
3126                                !quest &&
3127                                (report_dependencies_level == 0) /*  &&
3128                                (exists(default_target_to_build) > file_doesnt_exist
3129                                    if (posix) {
3130                                            if (!commands_done) {
3131                                                    (void) printf(catgets(catd, 1, 2
3132                                                                          default_target_to_
3133                                            } else {
3134                                                    if (no_action_was_taken) {
3135                                                            (void) printf(catgets(ca
3136                                                                                  default_ta
3137                                                    }
3138                                            }
3139                                    } else {
3140                                            if (!commands_done &&
3141                                                (exists(default_target_to_build) > f
3142                                                    (void) printf(catgets(catd, 1, 2
3143                                                                          default_target_to_
3144                                            }
3145                                    }
3146                            }
3147                    }
3148            }

3150 /*
3151  *      If no file arguments have been encountered,
3152  *      make the first name encountered that doesnt start with a dot
3153  */
3154            if (!target_to_make_found) {
3155                    if (default_target_to_build == NULL) {
3156                            fatal(catgets(catd, 1, 202, "No arguments to build"));
3157                    }
3158                    commands_done = false;
3159                    top_level_target = get_wstring(default_target_to_build->string_m
3160                    report_recursion(default_target_to_build);


3163                    if (getenv(NOCATGETS("SPRO_EXPAND_ERRORS"))){
3164                            (void) printf(NOCATGETS("::(%s)\n"),
3165                                          default_target_to_build->string_mb);
3166                    }


3169 #ifdef TEAMWARE_MAKE_CMN
```

```
3170                 result = doname_parallel(default_target_to_build, true, false);
3171 #else
3172                 result = doname_check(default_target_to_build, true,
3173                                       false, false);
3174 #endif
3175                 gather_recursive_deps();
3176                 if (build_failed_seen) {
3177                         build_failed_ever_seen = true;
3178                         warning(catgets(catd, 1, 203, "Target `%s' not remade be
3179                                 default_target_to_build->string_mb);
3180                 }
3181                 build_failed_seen = false;
3182                 if (report_dependencies_level > 0) {
3183                         print_dependencies(default_target_to_build,
3184                                            get_prop(default_target_to_build->
3185                                                     prop,
3186                                                     line_prop));
3187                 }
3188                 default_target_to_build->stat.time = file_no_time;
3189                 if (default_target_to_build->colon_splits > 0) {
3190                         default_target_to_build->state = build_dont_know;
3191                 }
3192                 if (/* !commands_done && */
3193                     (result == build_ok) &&
3194                     !quest &&
3195                     (report_dependencies_level == 0) /*  &&
3196                     (exists(default_target_to_build) > file_doesnt_exist)  */) {
3197                         if (posix) {
3198                                 if (!commands_done) {
3199                                         (void) printf(catgets(catd, 1, 299, "`%s
3200                                                               default_target_to_build->s
3201                                 } else {
3202                                         if (no_action_was_taken) {
3203                                                 (void) printf(catgets(catd, 1, 3
3204                                                                       default_target_to_
3205                                         }
3206                                 }
3207                         } else {
3208                                 if (!commands_done &&
3209                                     (exists(default_target_to_build) > file_does
3210                                         (void) printf(catgets(catd, 1, 301, "`%s
3211                                                               default_target_to_build->s
3212                                 }
3213                         }
3214                 }
3215         }
3216 }

3218 /*
3219  *      report_recursion(target)
3220  *
3221  *      If this is a recursive make and the parent make has KEEP_STATE on
3222  *      this routine reports the dependency to the parent make
3223  *
3224  *      Parameters:
3225  *              target          Target to report
3226  *
3227  *      Global variables used:
3228  *              makefiles_used          List of makefiles read
3229  *              recursive_name          The Name ".RECURSIVE", printed
3230  *              report_dependency       dwight
3231  */
3232 static void
3233 report_recursion(register Name target)
3234 {
3235         register FILE           *report_file = get_report_file();
```



```
3237         if ((report_file == NULL) || (report_file == (FILE*)-1)) {
3238                 return;
3239         }
3240         if (primary_makefile == NULL) {
3241                 /*
3242                  * This can happen when there is no makefile and
3243                  * only implicit rules are being used.
3244                  */
3245                 return;
3246         }
3247         (void) fprintf(report_file,
3248                        "%s: %s ",
3249                        get_target_being_reported_for(),
3250                        recursive_name->string_mb);
3251         report_dependency(get_current_path());
3252         report_dependency(target->string_mb);
3253         report_dependency(primary_makefile->string_mb);
3254         (void) fprintf(report_file, "\n");
3255 }

3257 /* Next function "append_or_replace_macro_in_dyn_array" must be in "misc.cc". */
3258 /* NIKMOL */
3259 extern void
3260 append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar, char *macro)
3261 {
3262         register char   *cp0;   /* work pointer in macro */
3263         register char   *cp1;   /* work pointer in array */
3264         register char   *cp2;   /* work pointer in array */
3265         register char   *cp3;   /* work pointer in array */
3266         register char   *name;  /* macro name */
3267         register char   *value; /* macro value */
3268         register int    len_array;
3269         register int    len_macro;

3271         char * esc_value = NULL;
3272         int esc_len;

3274         if (!(len_macro = strlen(macro))) return;
3275         name = macro;
3276         while (isspace(*(name))) {
3277                 name++;
3278         }
3279         if (!(value = strchr(name, (int) equal_char))) {
3280                 /* no '=' in macro */
3281                 goto ERROR_MACRO;
3282         }
3283         cp0 = value;
3284         value++;
3285         while (isspace(*(value))) {
3286                 value++;
3287         }
3288         while (isspace(*(cp0-1))) {
3289                 cp0--;
3290         }
3291         if (cp0 <= name) goto ERROR_MACRO; /* no name */
3292         if (!(Ar->size)) goto ALLOC_ARRAY;
3293         cp1 = Ar->start;

3295 LOOK_FOR_NAME:
3296         if (!(cp1 = strchr(cp1, name[0]))) goto APPEND_MACRO;
3297         if (!(cp2 = strchr(cp1, (int) equal_char))) goto APPEND_MACRO;
3298         if (strncmp(cp1, name, (size_t)(cp0-name))) {
3299                 /* another name */
3300                 cp1++;
3301                 goto LOOK_FOR_NAME;
```

```
3302         }
3303         if (cp1 != Ar->start) {
3304                 if (!isspace(*(cp1-1))) {
3305                         /* another name */
3306                         cp1++;
3307                         goto LOOK_FOR_NAME;
3308                 }
3309         }
3310         for (cp3 = cp1 + (cp0-name); cp3 < cp2; cp3++) {
3311                 if (isspace(*cp3)) continue;
3312                 /* else: another name */
3313                 cp1++;
3314                 goto LOOK_FOR_NAME;
3315         }
3316         /* Look for the next macro name in array */
3317         cp3 = cp2+1;
3318         if (*cp3 != (int) doublequote_char) {
3319                 /* internal error */
3320                 goto ERROR_MACRO;
3321         }
3322         if (!(cp3 = strchr(cp3+1, (int) doublequote_char))) {
3323                 /* internal error */
3324                 goto ERROR_MACRO;
3325         }
3326         cp3++;
3327         while (isspace(*cp3)) {
3328                 cp3++;
3329         }
3330
3331         cp2 = cp1;  /* remove old macro */
3332         if ((*cp3) && (cp3 < Ar->start + Ar->size)) {
3333                 for (; cp3 < Ar->start + Ar->size; cp3++) {
3334                         *cp2++ = *cp3;
3335                 }
3336         }
3337         for (; cp2 < Ar->start + Ar->size; cp2++) {
3338                 *cp2 = 0;
3339         }
3340         if (*cp1) {
3341                 /* check next name */
3342                 goto LOOK_FOR_NAME;
3343         }
3344         goto APPEND_MACRO;

3346 ALLOC_ARRAY:
3347         if (Ar->size) {
3348                 cp1 = Ar->start;
3349         } else {
3350                 cp1 = 0;
3351         }
3352         Ar->size += 128;
3353         Ar->start = getmem(Ar->size);
3354         for (len_array=0; len_array < Ar->size; len_array++) {
3355                 Ar->start[len_array] = 0;
3356         }
3357         if (cp1) {
3358                 strcpy(Ar->start, cp1);
3359                 retmem((wchar_t *) cp1);
3360         }

3362 APPEND_MACRO:
3363         len_array = strlen(Ar->start);
3364         esc_value = (char*)malloc(strlen(value)*2 + 1);
3365         quote_str(value, esc_value);
3366         esc_len = strlen(esc_value) - strlen(value);
3367         if (len_array + len_macro + esc_len + 5 >= Ar->size) goto  ALLOC_ARRAY;
```



```
3368         strcat(Ar->start, " ");
3369         strncat(Ar->start, name, cp0-name);
3370         strcat(Ar->start, "=");
3371         strncat(Ar->start, esc_value, strlen(esc_value));
3372         free(esc_value);
3373         return;
3374 ERROR_MACRO:
3375         /* Macro without '=' or with invalid left/right part */
3376         return;
3377 }

3379 #ifdef TEAMWARE_MAKE_CMN
3380 /*
3381  * This function, if registered w/ avo_cli_get_license(), will be called
3382  * if the application is about to exit because:
3383  *   1) there has been certain unrecoverable error(s) that cause the
3384  *      application to exit immediately.
3385  *   2) the user has lost a license while the application is running.
3386  */
3387 extern "C" void
3388 dmake_exit_callback(void)
3389 {
3390         fatal(catgets(catd, 1, 306, "can not get a license, exiting..."));
3391         exit(1);
3392 }

3394 /*
3395  * This function, if registered w/ avo_cli_get_license(), will be called
3396  * if the application can not get a license.
3397  */
3398 extern "C" void
3399 dmake_message_callback(char *err_msg)
3400 {
3401         static Boolean  first = true;

3403         if (!first) {
3404                 return;
3405         }
3406         first = false;
3407         if ((!list_all_targets) &&
3408             (report_dependencies_level == 0) &&
3409             (dmake_mode_type != serial_mode)) {
3410                 warning(catgets(catd, 1, 313, "can not get a TeamWare license, d
3411         }
3412 }
3413 #endif

3415 #ifdef DISTRIBUTED
3416 /*
3417  * Returns whether -c is set or not.
3418  */
3419 Boolean
3420 get_dmake_rcfile_specified(void)
3421 {
3422         return(dmake_rcfile_specified);
3423 }

3425 /*
3426  * Returns whether -g is set or not.
3427  */
3428 Boolean
3429 get_dmake_group_specified(void)
3430 {
3431         return(dmake_group_specified);
3432 }
```

```
/*
 * Returns whether -j is set or not.
 */
Boolean
get_dmake_max_jobs_specified(void)
{
        return(dmake_max_jobs_specified);
}

/*
 * Returns whether -m is set or not.
 */
Boolean
get_dmake_mode_specified(void)
{
        return(dmake_mode_specified);
}

/*
 * Returns whether -o is set or not.
 */
Boolean
get_dmake_odir_specified(void)
{
        return(dmake_odir_specified);
}

#endif

static void
report_dir_enter_leave(Boolean entering)
{
        char    rcwd[MAXPATHLEN];
static  char *  mlev = NULL;
        char *  make_level_str = NULL;
        int     make_level_val = 0;

        make_level_str = getenv(NOCATGETS("MAKELEVEL"));
        if(make_level_str) {
                make_level_val = atoi(make_level_str);
        }
        if(mlev == NULL) {
                mlev = (char*) malloc(MAXPATHLEN);
        }
        if(entering) {
                sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
        } else {
                make_level_val--;
                sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
        }
        putenv(mlev);

        if(report_cwd) {
                if(make_level_val <= 0) {
                        if(entering) {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 329, "dmake: Entering
                                       , get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 330, "make: Entering d
                                       , get_current_path());
#endif
                        } else {
#ifdef TEAMWARE_MAKE_CMN
```



```
                                sprintf( rcwd
                                       , catgets(catd, 1, 331, "dmake: Leaving d
                                       , get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 332, "make: Leaving di
                                       , get_current_path());
#endif
                        }
                } else {
                        if(entering) {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 333, "dmake[%d]: Enter
                                       , make_level_val, get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 334, "make[%d]: Enteri
                                       , make_level_val, get_current_path());
#endif
                        } else {
#ifdef TEAMWARE_MAKE_CMN
                                sprintf( rcwd
                                       , catgets(catd, 1, 335, "dmake[%d]: Leavi
                                       , make_level_val, get_current_path());
#else
                                sprintf( rcwd
                                       , catgets(catd, 1, 336, "make[%d]: Leavin
                                       , make_level_val, get_current_path());
#endif
                        }
                }
                printf(NOCATGETS("%s"), rcwd);
        }
}
```

```
**********************************************************
   26174 Wed May 20 11:53:26 2015
new/usr/src/cmd/make/bin/misc.cc
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
______unchanged_portion_omitted_

 109 /*****************************************
 110  *
 111  *      String manipulation
 112  */

 114 /*****************************************
 115  *
 116  *      Nameblock property handling
 117  */

 119 /*****************************************
 120  *
 121  *      Error message handling
 122  */

 124 /*
 125  *      fatal(format, args...)
 126  *
 127  *      Print a message and die
 128  *
 129  *      Parameters:
 130  *              format          printf type format string
 131  *              args            Arguments to match the format
 132  *
 133  *      Global variables used:
 134  *              fatal_in_progress Indicates if this is a recursive call
 135  *              parallel_process_cnt Do we need to wait for anything?
 136  *              report_pwd      Should we report the current path?
 137  */
 138 /*VARARGS*/
 139 void
 140 fatal(const char *message, ...)
 140 fatal(char * message, ...)
 141 {
 142         va_list args;

 144         va_start(args, message);
 145         (void) fflush(stdout);
 146 #ifdef DISTRIBUTED
 147         (void) fprintf(stderr, catgets(catd, 1, 262, "dmake: Fatal error: "));
 148 #else
 149         (void) fprintf(stderr, catgets(catd, 1, 263, "make: Fatal error: "));
 150 #endif
 151         (void) vfprintf(stderr, message, args);
 152         (void) fprintf(stderr, "\n");
 153         va_end(args);
 154         if (report_pwd) {
 155                 (void) fprintf(stderr,
 156                                catgets(catd, 1, 156, "Current working directory
 157                                get_current_path());
 158         }
 159         (void) fflush(stderr);
 160         if (fatal_in_progress) {
 161                 exit_status = 1;
 162                 exit(1);
 163         }
 164         fatal_in_progress = true;
 165 #ifdef TEAMWARE_MAKE_CMN
 166         /* Let all parallel children finish */
```



```
 167         if ((dmake_mode_type == parallel_mode) &&
 168             (parallel_process_cnt > 0)) {
 169                 (void) fprintf(stderr,
 170                                catgets(catd, 1, 157, "Waiting for %d %s to finis
 171                                parallel_process_cnt,
 172                                parallel_process_cnt == 1 ?
 173                                catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
 174                 (void) fflush(stderr);
 175         }

 177         while (parallel_process_cnt > 0) {
 178 #ifdef DISTRIBUTED
 179                 if (dmake_mode_type == distributed_mode) {
 180                         (void) await_dist(false);
 181                 } else {
 182                         await_parallel(true);
 183                 }
 184 #else
 185                 await_parallel(true);
 186 #endif
 187                 finish_children(false);
 188         }
 189 #endif

 191 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
 192         job_adjust_fini();
 193 #endif

 195         exit_status = 1;
 196         exit(1);
 197 }
______unchanged_portion_omitted_
```

```
**********************************************************
   52890 Wed May 20 11:53:27 2015
new/usr/src/cmd/make/bin/parallel.cc
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
```

```
/*
 * 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 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

#ifdef TEAMWARE_MAKE_CMN

/*
 *      parallel.cc
 *
 *      Deal with the parallel processing
 */

/*
 * Included files
 */
#ifdef DISTRIBUTED
#include <avo/strings.h>        /* AVO_STRDUP() */
#include <dm/Avo_DoJobMsg.h>
#include <dm/Avo_MToolJobResultMsg.h>
#endif
#include <errno.h>              /* errno */
#include <fcntl.h>
#include <avo/util.h>           /* avo_get_user(), avo_hostname() */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* redirect_io() */
#include <mksh/macro.h>         /* expand_value() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/signal.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/wait.h>
#include <unistd.h>
#include <netdb.h>
#endif /* ! codereview */



/*
 * Defined macros
 */
```



```
 */
#define MAXRULES                100

/*
 * This const should be in avo_dms/include/AvoDmakeCommand.h
 */
const int local_host_mask = 0x20;


/*
 * typedefs & structs
 */


/*
 * Static variables
 */
#ifdef TEAMWARE_MAKE_CMN
static  Boolean         just_did_subtree = false;
static  char            local_host[MAXNAMELEN] = "";
static  char            user_name[MAXNAMELEN] = "";
#endif
static  int             pmake_max_jobs = 0;
static  pid_t           process_running = -1;
static  Running         *running_tail = &running_list;
static  Name            subtree_conflict;
static  Name            subtree_conflict2;


/*
 * File table of contents
 */
#ifdef DISTRIBUTED
static  void            append_dmake_cmd(Avo_DoJobMsg *dmake_job_msg, char *orig
static  void            append_job_result_msg(Avo_MToolJobResultMsg *msg, char *
static  void            send_job_result_msg(Running rp);
#endif
static  void            delete_running_struct(Running rp);
static  Boolean         dependency_conflict(Name target);
static  Doname          distribute_process(char **commands, Property line);
static  void            doname_subtree(Name target, Boolean do_get, Boolean impl
static  void            dump_out_file(char *filename, Boolean err);
static  void            finish_doname(Running rp);
static  void            maybe_reread_make_state(void);
static  void            process_next(void);
static  void            reset_conditionals(int cnt, Name *targets, Property *loc
static  pid_t           run_rule_commands(char *host, char **commands);
static  Property        *set_conditionals(int cnt, Name *targets);
static  void            store_conditionals(Running rp);


/*
 *      execute_parallel(line, waitflg)
 *
 *      DMake 2.x:
 *      parallel mode: spawns a parallel process to execute the command group.
 *      distributed mode: sends the command group down the pipe to rxm.
 *
 *      Return value:
 *                              The result of the execution
 *
 *      Parameters:
 *              line            The command group to execute
 */
Doname
execute_parallel(Property line, Boolean waitflg, Boolean local)
```

```
127 {
128         int                     argcnt;
129         int                     cmd_options = 0;
130         char                    *commands[MAXRULES + 5];
131         char                    *cp;
132 #ifdef DISTRIBUTED
133         Avo_DoJobMsg            *dmake_job_msg = NULL;
134 #endif
135         Name                    dmake_name;
136         Name                    dmake_value;
137         int                     ignore;
138         Name                    make_machines_name;
139         char                    **p;
140         Property                prop;
141         Doname                  result = build_ok;
142         Cmd_line                rule;
143         Boolean                 silent_flag;
144         Name                    target = line->body.line.target;
145         Boolean                 wrote_state_file = false;

147         if ((pmake_max_jobs == 0) &&
148             (dmake_mode_type == parallel_mode)) {
 55                 if (user_name[0] == '\0') {
 56                         avo_get_user(user_name, NULL);
 57                 }
149                 if (local_host[0] == '\0') {
150                         (void) gethostname(local_host, MAXNAMELEN);
 59                         strcpy(local_host, avo_hostname());
151                 }
152                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
153                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
154                 if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
155                     ((dmake_value = prop->body.macro.value) != NULL)) {
156                         pmake_max_jobs = atoi(dmake_value->string_mb);
157                         if (pmake_max_jobs <= 0) {
158                                 warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
159                                 warning(catgets(catd, 1, 309, "setting DMAKE_MAX
160                                 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
161                         }
162                 } else {
163                         /*
164                          * For backwards compatibility w/ PMake 1.x, when
165                          * DMake 2.x is being run in parallel mode, DMake
166                          * should parse the PMake startup file
167                          * $(HOME)/.make.machines to get the pmake_max_jobs.
168                          */
169                         MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
170                         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
171                         if (((prop = get_prop(dmake_name->prop, macro_prop)) !=
172                             ((dmake_value = prop->body.macro.value) != NULL)) {
173                                 make_machines_name = dmake_value;
174                         } else {
175                                 make_machines_name = NULL;
176                         }
177                         if ((pmake_max_jobs = read_make_machines(make_machines_n
178                                 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
179                         }
180                 }
181 #ifdef DISTRIBUTED
182                 if (send_mtool_msgs) {
183                         send_rsrc_info_msg(pmake_max_jobs, local_host, user_name
184                 }
185 #endif
186         }

188         if ((dmake_mode_type == serial_mode) ||
```



```
189             ((dmake_mode_type == parallel_mode) && (waitflg))) {
190                 return (execute_serial(line));
191         }

193 #ifdef DISTRIBUTED
194         if (dmake_mode_type == distributed_mode) {
195                 if(local) {
196 //                      return (execute_serial(line));
197                         waitflg = true;
198                 }
199                 dmake_job_msg = new Avo_DoJobMsg();
200                 dmake_job_msg->setJobId(++job_msg_id);
201                 dmake_job_msg->setTarget(target->string_mb);
202                 dmake_job_msg->setImmediateOutput(0);
203                 called_make = false;
204         } else
205 #endif
206         {
207                 p = commands;
208         }

210         argcnt = 0;
211         for (rule = line->body.line.command_used;
212              rule != NULL;
213              rule = rule->next) {
214                 if (posix && (touch || quest) && !rule->always_exec) {
215                         continue;
216                 }
217                 if (vpath_defined) {
218                         rule->command_line =
219                           vpath_translation(rule->command_line);
220                 }
221                 if (dmake_mode_type == distributed_mode) {
222                         cmd_options = 0;
223                         if(local) {
224                                 cmd_options |= local_host_mask;
225                         }
226                 } else {
227                         silent_flag = false;
228                         ignore = 0;
229                 }
230                 if (rule->command_line->hash.length > 0) {
231                         if (++argcnt == MAXRULES) {
232                                 if (dmake_mode_type == distributed_mode) {
233                                         /* XXX - tell rxm to execute on local ho
234                                         /* I WAS HERE!!! */
235                                 } else {
236                                         /* Too many rules, run serially instead.
237                                         return build_serial;
238                                 }
239                         }
240 #ifdef DISTRIBUTED
241                         if (dmake_mode_type == distributed_mode) {
242                                 /*
243                                  * XXX - set assign_mask to tell rxm
244                                  * to do the following.
245                                  */
246 /* From execute_serial():
247                                 if (rule->assign) {
248                                         result = build_ok;
249                                         do_assign(rule->command_line, target);
250  */
251                                 if (0) {
252                                 } else if (report_dependencies_level == 0) {
253                                         if (rule->ignore_error) {
254                                                 cmd_options |= ignore_mask;
```

```
255                                     }
256                                     if (rule->silent) {
257                                             cmd_options |= silent_mask;
258                                     }
259                                     if (rule->command_line->meta) {
260                                             cmd_options |= meta_mask;
261                                     }
262                                     if (rule->make_refd) {
263                                             cmd_options |= make_refd_mask;
264                                     }
265                                     if (do_not_exec_rule) {
266                                             cmd_options |= do_not_exec_mask;
267                                     }
268                                     append_dmake_cmd(dmake_job_msg,
269                                                      rule->command_line->str
270                                                      cmd_options);
271                                     /* Copying dosys()... */
272                                     if (rule->make_refd) {
273                                             if (waitflg) {
274                                                     dmake_job_msg->setImmedi
275                                             }
276                                             called_make = true;
277                                             if (command_changed &&
278                                                 !wrote_state_file) {
279                                                     write_state_file(0, fals
280                                                     wrote_state_file = true;
281                                             }
282                                     }
283                             }
284                     } else
285 #endif
286                     {
287                             if (rule->silent && !silent) {
288                                     silent_flag = true;
289                             }
290                             if (rule->ignore_error) {
291                                     ignore++;
292                             }
293                             /* XXX - need to add support for + prefix */
294                             if (silent_flag || ignore) {
295                                     *p = getmem((silent_flag ? 1 : 0) +
296                                                 ignore +
297                                                 (strlen(rule->
298                                                         command_line->
299                                                         string_mb)) +
300                                                 1);
301                                     cp = *p++;
302                                     if (silent_flag) {
303                                             *cp++ = (int) at_char;
304                                     }
305                                     if (ignore) {
306                                             *cp++ = (int) hyphen_char;
307                                     }
308                                     (void) strcpy(cp, rule->command_line->st
309                             } else {
310                                     *p++ = rule->command_line->string_mb;
311                             }
312                     }
313             }
314         }
315         if ((argcnt == 0) ||
316             (report_dependencies_level > 0)) {
317 #ifdef DISTRIBUTED
318                 if (dmake_job_msg) {
319                         delete dmake_job_msg;
320                 }
```



```
321 #endif
322                 return build_ok;
323         }
324 #ifdef DISTRIBUTED
325         if (dmake_mode_type == distributed_mode) {
326                 // Send  a DoJob message to the rxm process.
327                 distribute_rxm(dmake_job_msg);
328
329                 // Wait for an acknowledgement.
330                 Avo_AcknowledgeMsg *ackMsg = getAcknowledgeMsg();
331                 if (ackMsg) {
332                         delete ackMsg;
333                 }
334
335                 if (waitflg) {
336                         // Wait for, and process a job result.
337                         result = await_dist(waitflg);
338                         if (called_make) {
339                                 maybe_reread_make_state();
340                         }
341                         check_state(temp_file_name);
342                         if (result == build_failed) {
343                                 if (!continue_after_error) {
344
345 #ifdef PRINT_EXIT_STATUS
346                                         warning(NOCATGETS("I'm in execute_parall
347 #endif
348
349                                         fatal(catgets(catd, 1, 252, "Command fai
350                                               target->string_mb);
351                                 }
352                                 /*
353                                  * Make sure a failing command is not
354                                  * saved in .make.state.
355                                  */
356                                 line->body.line.command_used = NULL;
357                         }
358                         if (temp_file_name != NULL) {
359                                 free_name(temp_file_name);
360                         }
361                         temp_file_name = NULL;
362                         Property spro = get_prop(sunpro_dependencies->prop, macr
363                         if(spro != NULL) {
364                                 Name val = spro->body.macro.value;
365                                 if(val != NULL) {
366                                         free_name(val);
367                                         spro->body.macro.value = NULL;
368                                 }
369                         }
370                         spro = get_prop(sunpro_dependencies->prop, env_mem_prop)
371                         if(spro) {
372                                 char *val = spro->body.env_mem.value;
373                                 if(val != NULL) {
374                                         retmem_mb(val);
375                                         spro->body.env_mem.value = NULL;
376                                 }
377                         }
378                         return result;
379                 } else {
380                         parallel_process_cnt++;
381                         return build_running;
382                 }
383         } else
384 #endif
385         {
386                 *p = NULL;
```

```
 388                 Doname res = distribute_process(commands, line);
 389                 if (res == build_running) {
 390                         parallel_process_cnt++;
 391                 }

 393                 /*
 394                  * Return only those memory that were specially allocated
 395                  * for part of commands.
 396                  */
 397                 for (int i = 0; commands[i] != NULL; i++) {
 398                         if ((commands[i][0] == (int) at_char) ||
 399                             (commands[i][0] == (int) hyphen_char)) {
 400                                 retmem_mb(commands[i]);
 401                         }
 402                 }
 403                 return res;
 404         }
 405 }
_____unchanged_portion_omitted_

 645 /*
 646  *  void job_adjust_error()
 647  *
 648  *  Description:
 649  *      Prints warning message, cleans up job adjust data, and disables job adju
 650  *
 651  *  Environment:
 652  *      DMAKE_ADJUST_MAX_JOBS
 653  *
 654  *  External functions:
 655  *      putenv()
 656  *
 657  *  Static variables:
 658  *      job_adjust_mode Current job adjust mode
 659  */
 660 static void
 661 job_adjust_error() {
 662         if (job_adjust_mode != ADJUST_NONE) {
 663                 /* cleanup internals */
 664                 job_adjust_fini();

 666                 /* warning message for the user */
 667                 warning(catgets(catd, 1, 339, "Encountered max jobs auto adjustm

 669                 /* switch off job adjustment for the children */
 670                 putenv(strdup(NOCATGETS("DMAKE_ADJUST_MAX_JOBS=NO")));
 579                 putenv(NOCATGETS("DMAKE_ADJUST_MAX_JOBS=NO"));

 672                 /* and for this dmake */
 673                 job_adjust_mode = ADJUST_NONE;
 674         }
 675 }
_____unchanged_portion_omitted_

1698 /*
1699  *      add_running(target, true_target, command, recursion_level, auto_count,
1700  *                                      automatics, do_get, implicit)
1701  *
1702  *      Adds a record on the running list for this target, which
1703  *      was just spawned and is running.
1704  *
1705  *      Parameters:
1706  *              target          Target being built
1707  *              true_target     True target for target
1708  *              command         Running command.
```



```
1709  *              recursion_level Debug indentation level
1710  *              auto_count      Count of automatic dependencies
1711  *              automatics      List of automatic dependencies
1712  *              do_get          Sccs get flag
1713  *              implicit        Implicit flag
1714  *
1715  *      Static variables used:
1716  *              running_tail    Tail of running list
1717  *              process_running PID of process
1718  *
1719  *      Global variables used:
1720  *              current_line    Current line for target
1721  *              current_target  Current target being built
1722  *              stderr_file     Temporary file for stdout
1723  *              stdout_file     Temporary file for stdout
1724  *              temp_file_name  Temporary file for auto dependencies
1725  */
1726 void
1727 add_running(Name target, Name true_target, Property command, int recursion_level
1728 {
1729         Running         rp;
1730         Name            *p;

1732         rp = new_running_struct();
1733         rp->state = build_running;
1734         rp->target = target;
1735         rp->true_target = true_target;
1736         rp->command = command;
1646         Property spro_val = get_prop(sunpro_dependencies->prop, macro_prop);
1647         if(spro_val) {
1648                 rp->sprodep_value = spro_val->body.macro.value;
1649                 spro_val->body.macro.value = NULL;
1650                 spro_val = get_prop(sunpro_dependencies->prop, env_mem_prop);
1651                 if(spro_val) {
1652                         rp->sprodep_env = spro_val->body.env_mem.value;
1653                         spro_val->body.env_mem.value = NULL;
1654                 }
1655         }
1737         rp->recursion_level = recursion_level;
1738         rp->do_get = do_get;
1739         rp->implicit = implicit;
1740         rp->auto_count = auto_count;
1741         if (auto_count > 0) {
1742                 rp->automatics = (Name *) getmem(auto_count * sizeof (Name));
1743                 for (p = rp->automatics; auto_count > 0; auto_count--) {
1744                         *p++ = *automatics++;
1745                 }
1746         } else {
1747                 rp->automatics = NULL;
1748         }
1749 #ifdef DISTRIBUTED
1750         if (dmake_mode_type == distributed_mode) {
1751                 rp->make_refd = called_make;
1752                 called_make = false;
1753         } else
1754 #endif
1755         {
1756                 rp->pid = process_running;
1757                 process_running = -1;
1758                 childPid = -1;
1759         }
1760         rp->job_msg_id = job_msg_id;
1761         rp->stdout_file = stdout_file;
1762         rp->stderr_file = stderr_file;
1763         rp->temp_file = temp_file_name;
1764         rp->redo = false;
```

```
1765         rp->next = NULL;
1766         store_conditionals(rp);
1767         stdout_file = NULL;
1768         stderr_file = NULL;
1769         temp_file_name = NULL;
1770         current_target = NULL;
1771         current_line = NULL;
1772         *running_tail = rp;
1773         running_tail = &rp->next;
1774 }
```

_____***unchanged_portion_omitted_***

***********************************************************
***15381 Wed May 20 11:53:27 2015***
***new/usr/src/cmd/make/include/mk/defs.h***
***make: ship the Joyent patch to enable parallel make (originally from rm)***
***********************************************************
_____***unchanged_portion_omitted_***

```
177 /*
178  * Typedefs for all structs
179  */
180 typedef struct _Cmd_line        *Cmd_line, Cmd_line_rec;
181 typedef struct _Dependency      *Dependency, Dependency_rec;
182 typedef struct _Macro           *Macro, Macro_rec;
183 typedef struct _Name_vector     *Name_vector, Name_vector_rec;
184 typedef struct _Percent         *Percent, Percent_rec;
185 typedef struct _Dyntarget       *Dyntarget;
186 typedef struct _Recursive_make  *Recursive_make, Recursive_make_rec;
187 typedef struct _Running         *Running, Running_rec;


190 /*
191  *      extern declarations for all global variables.
192  *      The actual declarations are in globals.cc
193  */
194 extern  Boolean         allrules_read;
195 extern  Name            posix_name;
196 extern  Name            svr4_name;
197 extern  Boolean         sdot_target;
198 extern  Boolean         all_parallel;
199 extern  Boolean         assign_done;
200 extern  Boolean         build_failed_seen;
201 #ifdef DISTRIBUTED
202 extern  Boolean         building_serial;
203 #endif
204 extern  Name            built_last_make_run;
205 extern  Name            c_at;
206 #ifdef DISTRIBUTED
207 extern  Boolean         called_make;
208 #endif
209 extern  Boolean         command_changed;
210 extern  Boolean         commands_done;
211 extern  Chain           conditional_targets;
212 extern  Name            conditionals;
213 extern  Boolean         continue_after_error;
214 extern  Property        current_line;
215 extern  Name            current_make_version;
216 extern  Name            current_target;
217 extern  short           debug_level;
218 extern  Cmd_line        default_rule;
219 extern  Name            default_rule_name;
220 extern  Name            default_target_to_build;
221 extern  Boolean         depinfo_already_read;
222 extern  Name            dmake_group;
223 extern  Name            dmake_max_jobs;
224 extern  Name            dmake_mode;
225 extern  DMake_mode      dmake_mode_type;
226 extern  Name            dmake_output_mode;
227 extern  DMake_output_mode       output_mode;
228 extern  Name            dmake_odir;
229 extern  Name            dmake_rcfile;
230 extern  Name            done;
231 extern  Name            dot;
232 extern  Name            dot_keep_state;
233 extern  Name            dot_keep_state_file;
234 extern  Name            empty_name;
```



```
235 extern  Boolean         fatal_in_progress;
236 extern  int             file_number;
237 extern  Name            force;
238 extern  Name            ignore_name;
239 extern  Boolean         ignore_errors;
240 extern  Boolean         ignore_errors_all;
241 extern  Name            init;
242 extern  int             job_msg_id;
243 extern  Boolean         keep_state;
244 extern  Name            make_state;
245 #ifdef TEAMWARE_MAKE_CMN
246 extern  timestruc_t     make_state_before;
247 #endif
248 extern  Boolean         make_state_locked;
249 extern  Dependency      makefiles_used;
250 extern  Name            makeflags;
251 extern  Name            make_version;
252 extern  char            mbs_buffer2[];
253 extern  char            *mbs_ptr;
254 extern  char            *mbs_ptr2;
255 extern  Boolean         no_action_was_taken;
256 extern  int             mtool_msgs_fd;
257 extern  Boolean         no_parallel;
258 extern  Name            no_parallel_name;
259 extern  Name            not_auto;
260 extern  Boolean         only_parallel;
261 extern  Boolean         parallel;
262 extern  Name            parallel_name;
263 extern  Name            localhost_name;
264 extern  int             parallel_process_cnt;
265 extern  Percent         percent_list;
266 extern  Dyntarget       dyntarget_list;
267 extern  Name            plus;
268 extern  Name            pmake_machinesfile;
269 extern  Name            precious;
270 extern  Name            primary_makefile;
271 extern  Boolean         quest;
272 extern  short           read_trace_level;
273 extern  Boolean         reading_dependencies;
274 extern  int             recursion_level;
275 extern  Name            recursive_name;
276 extern  short           report_dependencies_level;
277 extern  Boolean         report_pwd;
278 extern  Boolean         rewrite_statefile;
279 extern  Running         running_list;
280 extern  char            *sccs_dir_path;
281 extern  Name            sccs_get_name;
282 extern  Name            sccs_get_posix_name;
283 extern  Cmd_line        sccs_get_rule;
284 extern  Cmd_line        sccs_get_org_rule;
285 extern  Cmd_line        sccs_get_posix_rule;
286 extern  Name            get_name;
287 extern  Name            get_posix_name;
288 extern  Cmd_line        get_rule;
289 extern  Cmd_line        get_posix_rule;
290 extern  Boolean         send_mtool_msgs;
291 extern  Boolean         all_precious;
292 extern  Boolean         report_cwd;
293 extern  Boolean         silent_all;
294 extern  Boolean         silent;
295 extern  Name            silent_name;
296 extern  char            *stderr_file;
297 extern  char            *stdout_file;
298 extern  Boolean         stdout_stderr_same;
299 extern  Dependency      suffixes;
300 extern  Name            suffixes_name;
```

```
301 extern  Name            sunpro_dependencies;
302 extern  Boolean         target_variants;
303 extern  const char      *tmpdir;
304 extern  const char      *temp_file_directory;
305 extern  Name            temp_file_name;
306 extern  short           temp_file_number;
307 extern  wchar_t         *top_level_target;
308 extern  Boolean         touch;
309 extern  Boolean         trace_reader;
310 extern  Boolean         build_unconditional;
311 extern  pathpt          vroot_path;
312 extern  Name            wait_name;
313 extern  wchar_t         wcs_buffer2[];
314 extern  wchar_t         *wcs_ptr;
315 extern  wchar_t         *wcs_ptr2;
316 extern  nl_catd         catd;
317 extern  long int        hostid;

319 /*
320  * Declarations of system defined variables
321  */
322 /* On linux this variable is defined in 'signal.h' */
323 extern  char            *sys_siglist[];

325 /*
326  * Declarations of system supplied functions
327  */
328 extern  int             file_lock(char *, char *, int *, int);

330 /*
331  * Declarations of functions declared and used by make
332  */
333 extern  void            add_pending(Name target, int recursion_level, Boolean do
334 extern  void            add_running(Name target, Name true_target, Property comm
335 extern  void            add_serial(Name target, int recursion_level, Boolean do_
336 extern  void            add_subtree(Name target, int recursion_level, Boolean do
337 extern  void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
338 #ifdef DISTRIBUTED
339 extern  Doname          await_dist(Boolean waitflg);
340 #endif
341 #ifdef TEAMWARE_MAKE_CMN
342 extern  void            await_parallel(Boolean waitflg);
343 #endif
344 extern  void            build_suffix_list(Name target_suffix);
345 extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
346 extern  void            check_state(Name temp_file_name);
347 extern  void            cond_macros_into_string(Name np, String_rec *buffer);
348 extern  void            construct_target_string();
349 extern  void            create_xdrs_ptr(void);
350 extern  void            depvar_add_to_list (Name name, Boolean cmdline);
351 #ifdef DISTRIBUTED
352 extern  void            distribute_rxm(Avo_DoJobMsg *dmake_job_msg);
353 extern  int             getRxmMessage(void);
354 extern  Avo_JobResultMsg* getJobResultMsg(void);
355 extern  Avo_AcknowledgeMsg* getAcknowledgeMsg(void);
356 #endif
357 extern  Doname          doname(register Name target, register Boolean do_get, re
358 extern  Doname          doname_check(register Name target, register Boolean do_g
359 extern  Doname          doname_parallel(Name target, Boolean do_get, Boolean imp
360 extern  Doname          dosys(register Name command, register Boolean ignore_err
361 extern  void            dump_make_state(void);
362 extern  void            dump_target_list(void);
363 extern  void            enter_conditional(register Name target, Name name, Name
364 extern  void            enter_dependencies(register Name target, Chain target_gr
365 extern  void            enter_dependency(Property line, register Name depe, Bool
366 extern  void            enter_equal(Name name, Name value, register Boolean appe
```



```
367 extern  Percent         enter_percent(register Name target, Chain target_group,
368 extern  Dyntarget       enter_dyntarget(register Name target);
369 extern  Name_vector     enter_name(String string, Boolean tail_present, register
370 extern  Boolean         exec_vp(register char *name, register char **argv, char
371 extern  Doname          execute_parallel(Property line, Boolean waitflg, Boolean
372 extern  Doname          execute_serial(Property line);
373 extern  timestruc_t&    exists(register Name target);
374 extern  void            fatal(const char *, ...);
374 extern  void            fatal(char *, ...);
375 extern  void            fatal_reader(char *, ...);
376 extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
377 extern  Doname          find_double_suffix_rule(register Name target, Property *
378 extern  Doname          find_percent_rule(register Name target, Property *comman
379 extern  int             find_run_directory (char *cmd, char *cwd, char *dir, cha
380 extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
381 extern  Chain           find_target_groups(register Name_vector target_list, reg
382 extern  void            finish_children(Boolean docheck);
383 extern  void            finish_running(void);
384 extern  void            free_chain(Name_vector ptr);
385 extern  void            gather_recursive_deps(void);
386 extern  char            *get_current_path(void);
387 extern  int             get_job_msg_id(void);
388 extern  FILE            *get_mtool_msgs_fp(void);
389 #ifdef DISTRIBUTED
390 extern  Boolean         get_dmake_group_specified(void);
391 extern  Boolean         get_dmake_max_jobs_specified(void);
392 extern  Boolean         get_dmake_mode_specified(void);
393 extern  Boolean         get_dmake_odir_specified(void);
394 extern  Boolean         get_dmake_rcfile_specified(void);
395 extern  Boolean         get_pmake_machinesfile_specified(void);
396 #endif
397 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
398 extern  XDR             *get_xdrs_ptr(void);
399 #endif
400 extern  wchar_t         *getmem_wc(register int size);
401 /* On linux getwd(char *) is defined in 'unistd.h' */
402 #ifdef __cplusplus
403 extern "C" {
404 #endif
405 extern  char            *getwd(char *);
406 #ifdef __cplusplus
407 }
```

*unchanged_portion_omitted_*

***********************************************************
***1486 Wed May 20 11:53:28 2015***
***new/usr/src/cmd/make/lib/mksdmsi18n/libmksdmsi18n_init.cc***
***make: ship the Joyent patch to enable parallel make (originally from rm)***
***********************************************************

```
/*
 * 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 1996 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

#include <avo/intl.h>
#include <stdio.h>
#include <stdlib.h>

nl_catd libmksdmsi18n_catd;

/*
 * Open the catalog file for libmksdmsi18n.  Users of this library must set
 * NSLPATH first.  See avo_18n_init().
 */
int
libmksdmsi18n_init()
{
        char            name[20];

        if (getenv(NOCATGETS("NLSPATH")) == NULL) {
                fprintf(stderr, NOCATGETS("Internal error: Set NLSPATH before op
                return 1;
        }
        sprintf(name, NOCATGETS("libmksdmsi18n_%d"), I18N_VERSION);
        libmksdmsi18n_catd = catopen(name, NL_CAT_LOCALE);
        return 0;
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   20840 Wed May 20 11:53:29 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
```

```
/*
 * 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 2005 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


/*
 *      dosys.cc
 *
 *      Execute one commandline
 */

/*
 * Included files
 */
#include <sys/wait.h>                   /* WIFEXITED(status) */
#include <alloca.h>             /* alloca() */

#if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL) /* tolik */
#       include <avo/strings.h> /* AVO_STRDUP() */
#if defined(DISTRIBUTED)
#       include <dm/Avo_CmdOutput.h>
#       include <rw/xdrstrea.h>
#endif
#endif

#include <stdio.h>              /* errno */
#include <errno.h>              /* errno */
#include <fcntl.h>              /* open() */
#include <mksh/dosys.h>
#include <mksh/macro.h>         /* getvar() */
#include <mksh/misc.h>          /* getmem(), fatal_mksh(), errmsg() */
#include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */
#include <sys/signal.h>         /* SIG_DFL */
#include <sys/stat.h>           /* open() */
#include <sys/wait.h>           /* wait() */
#include <ulimit.h>             /* ulimit() */
#include <unistd.h>             /* close(), dup2() */
```



```
/*
 * Defined macros
 */
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#define SEND_MTOOL_MSG(cmds) \
        if (send_mtool_msgs) { \
                cmds \
        }
#else
#define SEND_MTOOL_MSG(cmds)
#endif

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
 */
static Boolean  exec_vp(register char *name, register char **argv, char **envp,

/*
 * Workaround for NFS bug. Sometimes, when running 'open' on a remote
 * dmake server, it fails with "Stale NFS file handle" error.
 * The second attempt seems to work.
 */
int
my_open(const char *path, int oflag, mode_t mode) {
        int res = open(path, oflag, mode);
        if (res < 0 && (errno == ESTALE || errno == EAGAIN)) {
                /* Stale NFS file handle. Try again */
                res = open(path, oflag, mode);
        }
        return res;
}
```

***unchanged_portion_omitted_***

```
/*
 *      await(ignore_error, silent_error, target, command, running_pid)
 *
 *      Wait for one child process and analyzes
 *      the returned status when the child process terminates.
 *
 *      Return value:
 *                              Returns true if commands ran OK
 *
 *      Parameters:
 *              ignore_error    Should we abort on error?
 *              silent_error    Should error messages be suppressed for dmake?
 *              target          The target we are building, for error msgs
 *              command         The command we ran, for error msgs
 *              running_pid     The pid of the process we are waiting for
 *
 *      Static variables used:
 *              filter_file     The fd for the filter file
 *              filter_file_name The name of the filter file
 *
 *      Global variables used:
 *              filter_stderr   Set if -X is on
 */
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
Boolean
```

```
577 await(register Boolean ignore_error, register Boolean silent_error, Name target,
578 #else
579 Boolean
580 await(register Boolean ignore_error, register Boolean silent_error, Name target,
581 #endif
582 {
583         int                     status;
584         char                    *buffer;
585         int                     core_dumped;
586         int                     exit_status;
587 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
588         Avo_CmdOutput           *make_output_msg;
589 #endif
590         FILE                    *outfp;
591         register pid_t          pid;
592         struct stat             stat_buff;
593         int                     termination_signal;
594         char                    tmp_buf[MAXPATHLEN];
595 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
596         RWCollectable           *xdr_msg;
597 #endif

599         while ((pid = wait(&status)) != running_pid) {
600                 if (pid == -1) {
601                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 98, "wait() fa
602                 }
603         }
604         (void) fflush(stdout);
605         (void) fflush(stderr);

607         if (status == 0) {

609 #ifdef PRINT_EXIT_STATUS
610                 warning_mksh(NOCATGETS("I'm in await(), and status is 0."));
611 #endif

613                 return succeeded;
614         }

616 #ifdef PRINT_EXIT_STATUS
617         warning_mksh(NOCATGETS("I'm in await(), and status is *NOT* 0."));
618 #endif


621         exit_status = WEXITSTATUS(status);

623 #ifdef PRINT_EXIT_STATUS
624         warning_mksh(NOCATGETS("I'm in await(), and exit_status is %d."), exit_s
625 #endif

627         termination_signal = WTERMSIG(status);
628         core_dumped = WCOREDUMP(status);

630         /*
631          * If the child returned an error, we now try to print a
632          * nice message about it.
633          */
634         SEND_MTOOL_MSG(
635                 make_output_msg = new Avo_CmdOutput();
636                 (void) sprintf(tmp_buf, "%d", job_msg_id);
637                 make_output_msg->appendOutput(strdup(tmp_buf));
638                 make_output_msg->appendOutput(AVO_STRDUP(tmp_buf));
638         );

640         tmp_buf[0] = (int) nul_char;
641         if (!silent_error) {
```



```
642                 if (exit_status != 0) {
643                         (void) fprintf(stdout,
644                                 catgets(libmksdmsi18n_catd, 1, 103, "***
645                                 exit_status);
646                         SEND_MTOOL_MSG(
647                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
648                                         catgets(libmksdmsi18n_catd, 1, 10
649                                         exit_status);
650                         );
651                 } else {
652                         (void) fprintf(stdout,
653                                         catgets(libmksdmsi18n_catd, 1, 10
654                                         termination_signal);
655                         SEND_MTOOL_MSG(
656                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)]
657                                         catgets(libmksdmsi18n_cat
658                                         termination_signal);
659                         );
660                         if (core_dumped) {
661                                 (void) fprintf(stdout,
662                                         catgets(libmksdmsi18n_catd, 1, 10
663                                 SEND_MTOOL_MSG(
664                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
665                                                 catgets(libmksdmsi18n_cat
666                                 );
667                         }
668                 }
669                 if (ignore_error) {
670                         (void) fprintf(stdout,
671                                 catgets(libmksdmsi18n_catd, 1, 109, " (ig
672                         SEND_MTOOL_MSG(
673                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
674                                         catgets(libmksdmsi18n_catd, 1, 11
675                         );
676                 }
677                 (void) fprintf(stdout, "\n");
678                 (void) fflush(stdout);
679                 SEND_MTOOL_MSG(
680                         make_output_msg->appendOutput(strdup(tmp_buf));
681                         make_output_msg->appendOutput(AVO_STRDUP(tmp_buf));
681                 );
682         }
683         SEND_MTOOL_MSG(
684                 xdr_msg = (RWCollectable*) make_output_msg;
685                 xdr(xdrs_p, xdr_msg);
686                 delete make_output_msg;
687         );

689 #ifdef PRINT_EXIT_STATUS
690         warning_mksh(NOCATGETS("I'm in await(), returning failed."));
691 #endif

693         return failed;
694 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    7428 Wed May 20 11:53:29 2015
new/usr/src/cmd/make/lib/mksh/mksh.cc
make: ship the Joyent patch to enable parallel make (originally from rm)
**********************************************************
/*
 * 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 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


/*
 *      mksh.cc
 *
 *      Execute the command(s) of one Make or DMake rule
 */

/*
 * Included files
 */
#if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL) /* tolik */
#       include <avo/util.h>
#endif

#include <mksh/dosys.h>         /* redirect_io() */
#include <mksh/misc.h>          /* retmem() */
#include <mksh/mksh.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <errno.h>
#include <signal.h>


/*
 * Workaround for NFS bug. Sometimes, when running 'chdir' on a remote
 * dmake server, it fails with "Stale NFS file handle" error.
 * The second attempt seems to work.
 */
int
my_chdir(char * dir) {
        int res = chdir(dir);
        if (res != 0 && (errno == ESTALE || errno == EAGAIN)) {
                /* Stale NFS file handle. Try again */
                res = chdir(dir);
        }
        return res;
}
_____unchanged_portion_omitted_
```