```
**********************************************************
    1729 Wed May 20 11:55:51 2015
new/usr/src/cmd/make/bin/Makefile
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
#
# 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          \
        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

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
```

```
**********************************************************
   95642 Wed May 20 11:55:52 2015
new/usr/src/cmd/make/bin/doname.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
/*
 * 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(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
extern  void            reset_locals(register Name target, register Property old
extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
static  void            delete_query_chain(Chain ch);
```

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


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

/*
 * DONE.
 *
 *      doname_check(target, do_get, implicit, automatic)
 *
 *      Will call doname() and then inspect the return value
 *
 *      Return value:
 *                              Indication if the build failed or not
 *
 *      Parameters:
 *              target          The target to build
 *              do_get          Passed thru to doname()
 *              implicit        Passed thru to doname()
 *              automatic       Are we building a hidden dependency?
 *
 *      Global variables used:
 *              build_failed_seen       Set if -k is on and error occurs
 *              continue_after_error    Indicates that -k is on
 *              report_dependencies     No error msg if -P is on
 */
Doname
doname_check(register Name target, register Boolean do_get, register Boolean imp
{
        int first_time = 1;
        (void) fflush(stdout);
try_again:
        switch (doname(target, do_get, implicit, automatic)) {
        case build_ok:
                second_pass = 0;
                return build_ok;
        case build_running:
                second_pass = 0;
                return build_running;
        case build_failed:
                if (!continue_after_error) {
                        fatal(catgets(catd, 1, 13, "Target ‘%s’ not remade becau
                              target->string_mb);
                }
                build_failed_seen = true;
                second_pass = 0;
                return build_failed;
        case build_dont_know:
                /*
                 * If we can’t figure out how to build an automatic
                 * (hidden) dependency, we just ignore it.
                 * We later declare the target to be out of date just in
                 * case something changed.
                 * Also, don’t complain if just reporting the dependencies
                 * and not building anything.
                 */
                if (automatic || (report_dependencies_level > 0)) {
```

```
                        second_pass = 0;
                        return build_dont_know;
                }
                if(first_time) {
                        first_time = 0;
                        second_pass = 1;
                        goto try_again;
                }
                second_pass = 0;
                if (continue_after_error && !svr4) {
                        warning(catgets(catd, 1, 14, "Don’t know how to make tar
                                target->string_mb);
                        build_failed_seen = true;
                        return build_failed;
                }
                fatal(catgets(catd, 1, 15, "Don’t know how to make target ‘%s’")
                break;
        }
#ifdef lint
        return build_failed;
#endif
}
______unchanged_portion_omitted_

/*
 * DONE.
 *
 *      check_dependencies(result, line, do_get,
 *                      target, true_target, doing_subtree, out_of_date_tail,
 *                      old_locals, implicit, command, less, rechecking_target)
 *
 *      Return value:
 *                              True returned if some dependencies left running
 *
 *      Parameters:
 *              result          Pointer to cell we update if build failed
 *              line            We get the dependencies from here
 *              do_get          Allow use of sccs get in recursive doname()
 *              target          The target to chase dependencies for
 *              true_target     The real one for :: and lib(member)
 *              doing_subtree   True if building a conditional macro subtree
 *              out_of_date_tail Used to set the $? list
 *              old_locals      Used for resetting the local macros
 *              implicit        Called when scanning for implicit rules?
 *              command         Place to stuff command
 *              less            Set to $< value
 *
 *      Global variables used:
 *              command_changed Set if we suspect .make.state needs rewrite
 *              debug_level     Should we trace actions?
 *              force           The Name " FORCE", compared against
 *              recursion_level Used for tracing
 *              rewrite_statefile Set if .make.state needs rewriting
 *              wait_name       The Name ".WAIT", compared against
 */
static Boolean
#ifdef TEAMWARE_MAKE_CMN
check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
#else
check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
#endif
{
        Boolean                 dependencies_running;
        register Dependency     dependency;
        Doname                  dep_result;
        Boolean                 dependency_changed = false;
        line->body.line.dependency_time = file_doesnt_exist;
        if (line->body.line.query != NULL) {
                delete_query_chain(line->body.line.query);
        }
        line->body.line.query = NULL;
        line->body.line.is_out_of_date = false;
        dependencies_running = false;
        /*
         * Run thru all the dependencies and call doname() recursively
         * on each of them.
         */
        for (dependency = line->body.line.dependencies;
             dependency != NULL;
             dependency = dependency->next) {
                Boolean this_dependency_changed = false;

                if (!dependency->automatic &&
                    (rechecking_target || target->rechecking_target)) {
                        /*
                         * We only bother with the autos when rechecking
                         */
                        continue;
                }

                if (dependency->name == wait_name) {
                        /*
                         * The special target .WAIT means finish all of
                         * the prior dependencies before continuing.
                         */
                        if (dependencies_running) {
                                break;
                        }
#ifdef DISTRIBUTED
                } else if ((!parallel_ok(dependency->name, false)) &&
                           (dependencies_running)) {
                        /*
                         * If we can't execute the current dependency in
                         * parallel, hold off the dependency processing
                         * to preserve the order of the dependencies.
                         */
                        break;
#endif
                } else {
                        timestruc_t     depe_time = file_doesnt_exist;


                        if (true_target->is_member) {
                                depe_time = exists(dependency->name);
                        }
                        if (dependency->built ||
                            (dependency->name->state == build_failed)) {
                                dep_result = (Doname) dependency->name->state;
                        } else {
                                dep_result = doname_check(dependency->name,
                                                          do_get,
                                                          false,
                                                          (Boolean) dependency->
                        }
                        if (true_target->is_member || dependency->name->is_membe
                                /* should compare only secs, cause lib members d
                                if (depe_time.tv_sec != dependency->name->stat.t
                                        this_dependency_changed =
                                          dependency_changed =
                                            true;
                                }
```

```
                        } else {
                                if (depe_time != dependency->name->stat.time) {
                                        this_dependency_changed =
                                          dependency_changed =
                                            true;
                                }
                        }
                        dependency->built = true;
                        switch (dep_result) {
                        case build_running:
                                dependencies_running = true;
                                continue;
                        case build_failed:
                                *result = build_failed;
                                break;
                        case build_dont_know:
/*
 * If make can't figure out how to make a dependency, maybe the dependency
 * is out of date. In this case, we just declare the target out of date
 * and go on. If we really need the dependency, the make'ing of the target
 * will fail. This will only happen for automatic (hidden) dependencies.
 */
                                if(!recheck_conditionals) {
                                        line->body.line.is_out_of_date = true;
                                }
                                /*
                                 * Make sure the dependency is not saved
                                 * in the state file.
                                 */
                                dependency->stale = true;
                                rewrite_statefile =
                                  command_changed =
                                    true;
                                if (debug_level > 0) {
                                        (void) printf(catgets(catd, 1, 19, "Targ
                                                    true_target->string_mb,
                                                    dependency->name->string_mb
                                }
                                break;
                        }
                        if (dependency->name->depends_on_conditional) {
                                target->depends_on_conditional = true;
                        }
                        if (dependency->name == force) {
                                target->stat.time =
                                  dependency->name->stat.time;
                        }
                        /*
                         * Propagate new timestamp from "member" to
                         * "lib.a(member)".
                         */
                        (void) exists(dependency->name);

                        /* Collect the timestamp of the youngest dependency */
                        line->body.line.dependency_time =
                          MAX(dependency->name->stat.time,
                              line->body.line.dependency_time);

                        /* Correction: do not consider nanosecs for members */
                        if(true_target->is_member || dependency->name->is_member
                                line->body.line.dependency_time.tv_nsec = 0;
                        }

                        if (debug_level > 1) {
                                (void) printf(catgets(catd, 1, 20, "%*sDate(%s)=
                                              recursion_level,
                                              "",
                                              dependency->name->string_mb,
                                              time_to_string(dependency->name->
                                                             stat.time));
                                if (dependency->name->stat.time > line->body.lin
                                        (void) printf(catgets(catd, 1, 21, "%*sD
                                                      recursion_level,
                                                      "",
                                                      true_target->string_mb,
                                                      time_to_string(line->body.
                                                                     dependency_
                                }
                        }

                        /* Build the $? list */
                        if (true_target->is_member) {
                                if (this_dependency_changed == true) {
                                        true_target->stat.time = dependency->nam
                                        true_target->stat.time.tv_sec--;
                                } else {
                                        /* Dina:
                                         * The next statement is commented
                                         * out as a fix for bug #1051032.
                                         * if dependency hasn't changed
                                         * then there's no need to invalidate
                                         * true_target. This statemnt causes
                                         * make to take much longer to process
                                         * an already-built archive. Soren
                                         * said it was a quick fix for some
                                         * problem he doesn't remember.
                                        true_target->stat.time = file_no_time;
                                         */
                                        (void) exists(true_target);
                                }
                        } else {
                                (void) exists(true_target);
                        }
                        Boolean out_of_date;
                        if (true_target->is_member || dependency->name->is_membe
                                out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
                                                                        dependen
                        } else {
                                out_of_date = (Boolean) OUT_OF_DATE(true_target-
                                                                    dependency->
                        }
                        if ((build_unconditional || out_of_date) &&
                            (dependency->name != force) &&
                            (dependency->stale == false)) {
                                *out_of_date_tail = ALLOC(Chain);
                                if (dependency->name->is_member &&
                                    (get_prop(dependency->name->prop,
                                              member_prop) != NULL)) {
                                        (*out_of_date_tail)->name =
                                          get_prop(dependency->name->prop,
                                                   member_prop)->
                                                     body.member.member;
                                } else {
                                        (*out_of_date_tail)->name =
                                          dependency->name;
                                }
                                (*out_of_date_tail)->next = NULL;
                                out_of_date_tail = &(*out_of_date_tail)->next;
                                if (debug_level > 0) {
                                        if (dependency->name->stat.time == file_
                                                (void) printf(catgets(catd, 1, 2
                                                              recursion_level,
```

```
                                                              "",
                                                              true_target->strin
                                                              dependency->name->
                                        } else {
                                                (void) printf(catgets(catd, 1, 2
                                                              recursion_level,
                                                              "",
                                                              true_target->strin
                                                              dependency->name->
                                        }
                                }
                        }
                        if (dependency->name == force) {
                                force->stat.time =
                                  file_max_time;
                                force->state = build_dont_know;
                        }
                }
        }
#ifdef TEAMWARE_MAKE_CMN
        if (dependencies_running) {
                if (doing_subtree) {
                        if (target->conditional_cnt > 0) {
                                reset_locals(target,
                                             old_locals,
                                             get_prop(target->prop,
                                                      conditional_prop),
                                             0);
                        }
                        return true;
                } else {
                        target->state = build_running;
                        add_pending(target,
                                    --recursion_level,
                                    do_get,
                                    implicit,
                                    false);
                        if (target->conditional_cnt > 0) {
                                reset_locals(target,
                                             old_locals,
                                             get_prop(target->prop,
                                                      conditional_prop),
                                             0);
                        }
                        return true;
                }
        }
#endif
        /*
         * Collect the timestamp of the youngest double colon target
         * dependency.
         */
        if (target->is_double_colon_parent) {
                for (dependency = line->body.line.dependencies;
                     dependency != NULL;
                     dependency = dependency->next) {
                        Property        tmp_line;

                        if ((tmp_line = get_prop(dependency->name->prop, line_pr
                                if(tmp_line->body.line.dependency_time != file_m
                                        target->stat.time =
                                          MAX(tmp_line->body.line.dependency_tim
                                              target->stat.time);
                                }
                        }
                }
        }
        if ((true_target->is_member) && (dependency_changed == true)) {
                true_target->stat.time = file_no_time;
        }
        /*
         * After scanning all the dependencies, we check the rule
         * if we found one.
         */
        if (line->body.line.command_template != NULL) {
                if (line->body.line.command_template_redefined) {
                        warning(catgets(catd, 1, 24, "Too many rules defined for
                                target->string_mb);
                }
                *command = line;
                /* Check if the target is out of date */
                Boolean out_of_date;
                if (true_target->is_member) {
                        out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target->sta
                                                                line->body.line.
                } else {
                        out_of_date = (Boolean) OUT_OF_DATE(true_target->stat.ti
                                                            line->body.line.depe
                }
                if (build_unconditional || out_of_date){
                        if(!recheck_conditionals) {
                                line->body.line.is_out_of_date = true;
                        }
                }
                line->body.line.sccs_command = false;
                line->body.line.target = true_target;
                if(gnu_style) {

                        // set $< for explicit rule
                        if(line->body.line.dependencies != NULL) {
                                less = line->body.line.dependencies->name;
                        }

                        // set $* for explicit rule
                        Name                    target_body;
                        Name                    tt = true_target;
                        Property                member;
                        register wchar_t        *target_end;
                        register Dependency     suffix;
                        register int            suffix_length;
                        Wstring                 targ_string;
                        Wstring                 suf_string;

                        if (true_target->is_member &&
                            ((member = get_prop(target->prop, member_prop)) !=
                             NULL)) {
                                tt = member->body.member.member;
                        }
                        targ_string.init(tt);
                        target_end = targ_string.get_string() + tt->hash.length;
                        for (suffix = suffixes; suffix != NULL; suffix = suffix-
                                suffix_length = suffix->name->hash.length;
                                suf_string.init(suffix->name);
                                if (tt->hash.length < suffix_length) {
                                        continue;
                                } else if (!IS_WEQUALN(suf_string.get_string(),
                                                (target_end - suffix_length),
                                                suffix_length)) {
                                        continue;
                                }
                                target_body = GETNAME(
                                        targ_string.get_string(),
```

```
                                        (int)(tt->hash.length - suffix_length)
                                );
                                line->body.line.star = target_body;
                        }

                        // set result = build_ok so that implicit rules are not
                        if(*result == build_dont_know) {
                                *result = build_ok;
                        }
                }
                if (less != NULL) {
                        line->body.line.less = less;
                }
        }

        return false;
}
```

_____***unchanged_portion_omitted_***

```
/*
 *      execute_serial(line)
 *
 *      Runs thru the command line for the target and
 *      executes the rules one by one.
 *
 *      Return value:
 *                              The result of the command build
 *
 *      Parameters:
 *              line            The command to execute
 *
 *      Static variables used:
 *
 *      Global variables used:
 *              continue_after_error -k flag
 *              do_not_exec_rule -n flag
 *              report_dependencies -P flag
 *              silent          Don't echo commands before executing
 *              temp_file_name  Temp file for auto dependencies
 *              vpath_defined   If true, translate path for command
 */
Doname
execute_serial(Property line)
{
        int                     child_pid = 0;
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
        Avo_MToolJobResultMsg   *job_result_msg;
        RWCollectable           *xdr_msg;
#endif
        Boolean                 printed_serial;
        Doname                  result = build_ok;
        Cmd_line                rule, cmd_tail, command = NULL;
        char                    mbstring[MAXPATHLEN];
        int                     filed;
        Name                    target = line->body.line.target;

        SEND_MTOOL_MSG(
                if (!sent_rsrc_info_msg) {
                        if (userName[0] == '\0') {
                                avo_get_user(userName, NULL);
                        }
                        if (hostName[0] == '\0') {
                                strcpy(hostName, avo_hostname());
                        }
                        send_rsrc_info_msg(1, hostName, userName);
                        sent_rsrc_info_msg = 1;
                }
                send_job_start_msg(line);
                job_result_msg = new Avo_MToolJobResultMsg();
        );

        target->has_recursive_dependency = false;
        // We have to create a copy of the rules chain for processing because
        // the original one can be destroyed during .make.state file rereading.
        for (rule = line->body.line.command_used;
             rule != NULL;
             rule = rule->next) {
                if (command == NULL) {
                        command = cmd_tail = ALLOC(Cmd_line);
                } else {
                        cmd_tail->next = ALLOC(Cmd_line);
                        cmd_tail = cmd_tail->next;
                }
                *cmd_tail = *rule;
        }
        if (command) {
                cmd_tail->next = NULL;
        }
        for (rule = command; rule != NULL; rule = rule->next) {
                if (posix && (touch || quest) && !rule->always_exec) {
                        continue;
                }
                if (vpath_defined) {
                        rule->command_line =
                          vpath_translation(rule->command_line);
                }
                /* Echo command line, maybe. */
                if ((rule->command_line->hash.length > 0) &&
                    !silent &&
                    (!rule->silent || do_not_exec_rule) &&
                    (report_dependencies_level == 0)) {
                        (void) printf("%s\n", rule->command_line->string_mb);
                        SEND_MTOOL_MSG(
                                job_result_msg->appendOutput(AVO_STRDUP(rule->co
                        );
                }
                if (rule->command_line->hash.length > 0) {
                        SEND_MTOOL_MSG(
                                (void) sprintf(mbstring,
                                                NOCATGETS("%s/make.stdout.%d.%d.
                                                tmpdir,
                                                getpid(),
                                                file_number++);

                                int tmp_fd = mkstemp(mbstring);
                                if(tmp_fd) {
                                        (void) close(tmp_fd);
                                }

                                stdout_file = strdup(mbstring);
                                stderr_file = NULL;
                                child_pid = pollResults(stdout_file,
                                                        (char *)NULL,
                                                        (char *)NULL);
                        );
                        /* Do assignment if command line prefixed with "=" */
                        if (rule->assign) {
                                result = build_ok;
                                do_assign(rule->command_line, target);
                        } else if (report_dependencies_level == 0) {
                                /* Execute command line. */
#ifdef DISTRIBUTED
```

```
                                setvar_envvar((Avo_DoJobMsg *)NULL);
#else
                                setvar_envvar();
#endif
                                result = dosys(rule->command_line,
                                               (Boolean) rule->ignore_error,
                                               (Boolean) rule->make_refd,
                                               /* ds 98.04.23 bug #4085164. make
                                               false,
                                               /* BOOLEAN(rule->silent &&
                                                       rule->ignore_error), */
                                               (Boolean) rule->always_exec,
                                               target,
                                               send_mtool_msgs);
                                check_state(temp_file_name);
                        }
                        SEND_MTOOL_MSG(
                                append_job_result_msg(job_result_msg);
                                if (child_pid > 0) {
                                        kill(child_pid, SIGUSR1);
                                        while (!((waitpid(child_pid, 0, 0) == -1
                                                && (errno == ECHILD)));
                                }
                                child_pid = 0;
                                (void) unlink(stdout_file);
                                retmem_mb(stdout_file);
                                stdout_file = NULL;
                        );
                } else {
                        result = build_ok;
                }
                if (result == build_failed) {
                        if (silent || rule->silent) {
                                (void) printf(catgets(catd, 1, 242, "The followi
                                              rule->command_line->string_mb);
                                SEND_MTOOL_MSG(
                                        job_result_msg->appendOutput(AVO_STRDUP(
                                        job_result_msg->appendOutput(AVO_STRDUP(
                                );
                        }
                        if (!rule->ignore_error && !ignore_errors) {
                                if (!continue_after_error) {
                                        SEND_MTOOL_MSG(
                                                job_result_msg->setResult(job_ms
                                                xdr_msg = (RWCollectable*)
                                                        job_result_msg;
                                                xdr(&xdrs, xdr_msg);
                                                (void) fflush(mtool_msgs_fp);
                                                delete job_result_msg;
                                        );
                                        fatal(catgets(catd, 1, 244, "Command fai
                                              target->string_mb);
                                }
                                /*
                                 * Make sure a failing command is not
                                 * saved in .make.state.
                                 */
                                line->body.line.command_used = NULL;
                                break;
                        } else {
                                result = build_ok;
                        }
                }
        }
        for (rule = command; rule != NULL; rule = cmd_tail) {
                cmd_tail = rule->next;
                free(rule);
        }
        command = NULL;
        SEND_MTOOL_MSG(
                job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
                xdr_msg = (RWCollectable*) job_result_msg;
                xdr(&xdrs, xdr_msg);
                (void) fflush(mtool_msgs_fp);

                delete job_result_msg;
        );
        if (temp_file_name != NULL) {
                free_name(temp_file_name);
        }
        temp_file_name = NULL;

        Property spro = get_prop(sunpro_dependencies->prop, macro_prop);
        if(spro != NULL) {
                Name val = spro->body.macro.value;
                if(val != NULL) {
                        free_name(val);
                        spro->body.macro.value = NULL;
                }
        }
        spro = get_prop(sunpro_dependencies->prop, env_mem_prop);
        if(spro) {
                char *val = spro->body.env_mem.value;
                if(val != NULL) {
                        /*
                         * Do not return memory allocated for SUNPRO_DEPENDENCIE
                         * It will be returned in setvar_daemon() in macro.cc
                         */
                        //      retmem_mb(val);
                        spro->body.env_mem.value = NULL;
                }
        }

        return result;
}


#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */

/*
 * Create and send an Avo_MToolRsrcInfoMsg.
 */
void
send_rsrc_info_msg(int max_jobs, char *hostname, char *username)
{
        static int              first = 1;
        Avo_MToolRsrcInfoMsg    *msg;
        RWSlistCollectables     server_list;
        Avo_ServerState         *server_state;
        RWCollectable           *xdr_msg;

        if (!first) {
                return;
        }
        first = 0;

        create_xdrs_ptr();

        server_state = new Avo_ServerState(max_jobs, hostname, username);
        server_list.append(server_state);
        msg = new Avo_MToolRsrcInfoMsg(&server_list);
```

```
        xdr_msg = (RWCollectable *)msg;
        xdr(get_xdrs_ptr(), xdr_msg);
        (void) fflush(get_mtool_msgs_fp());

        delete server_state;
        delete msg;
}

/*
 * Create and send an Avo_MToolJobStartMsg.
 */
void
send_job_start_msg(Property line)
{
        int                     cmd_options = 0;
        Avo_MToolJobStartMsg    *msg;
        Cmd_line                rule;
        Name                    target = line->body.line.target;
        RWCollectable           *xdr_msg;

        if (userName[0] == '\0') {
                avo_get_user(userName, NULL);
        }
        if (hostName[0] == '\0') {
                strcpy(hostName, avo_hostname());
        }

        msg = new Avo_MToolJobStartMsg();
        msg->setJobId(++job_msg_id);
        msg->setTarget(AVO_STRDUP(target->string_mb));
        msg->setHost(AVO_STRDUP(hostName));
        msg->setUser(AVO_STRDUP(userName));

        for (rule = line->body.line.command_used;
             rule != NULL;
             rule = rule->next) {
                if (posix && (touch || quest) && !rule->always_exec) {
                        continue;
                }
                if (vpath_defined) {
                        rule->command_line =
                          vpath_translation(rule->command_line);
                }
                cmd_options = 0;
                if (rule->ignore_error || ignore_errors) {
                        cmd_options |= ignore_mask;
                }
                if (rule->silent || silent) {
                        cmd_options |= silent_mask;
                }
                if (rule->command_line->meta) {
                        cmd_options |= meta_mask;
                }
                if (!touch && (rule->command_line->hash.length > 0)) {
                        msg->appendCmd(new Avo_DmakeCommand(rule->command_line->
                }
        }

        xdr_msg = (RWCollectable*) msg;
        xdr(&xdrs, xdr_msg);
        (void) fflush(mtool_msgs_fp);

/* tolik, 08/39/2002.
   I commented out this code because it causes using unallocated memory.
        delete msg;
*/
```

```
}

/*
 * Append the stdout/err to Avo_MToolJobResultMsg.
 */
static void
append_job_result_msg(Avo_MToolJobResultMsg *job_result_msg)
{
        FILE            *fp;
        char            line[MAXPATHLEN];
        char            stdout_file2[MAXPATHLEN];

        if (stdout_file != NULL) {
                fp = fopen(stdout_file, "r");
                if (fp == NULL) {
                        /* Hmmm... what should we do here? */
                        warning(catgets(catd, 1, 326, "fopen() of stdout_file fa
                        return;
                }
                while (fgets(line, MAXPATHLEN, fp) != NULL) {
                        if (line[strlen(line) - 1] == '\n') {
                                line[strlen(line) - 1] = '\0';
                        }
                        job_result_msg->appendOutput(AVO_STRDUP(line));
                }
                (void) fclose(fp);
                us_sleep(STAT_RETRY_SLEEP_TIME);
        } else {
                /* Hmmm... stdout_file shouldn't be NULL */
                warning(catgets(catd, 1, 327, "Internal stdout_file variable sho
        }
}
#endif /* TEAMWARE_MAKE_CMN */

/*
 *      vpath_translation(cmd)
 *
 *      Translates one command line by
 *      checking each word. If the word has an alias it is translated.
 *
 *      Return value:
 *                              The translated command
 *
 *      Parameters:
 *              cmd             Command to translate
 *
 *      Global variables used:
 */
Name
vpath_translation(register Name cmd)
{
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        String_rec              new_cmd;
        wchar_t                 *p;
        wchar_t                 *start;

        if (!vpath_defined || (cmd == NULL) || (cmd->hash.length == 0)) {
                return cmd;
        }
        INIT_STRING_FROM_STACK(new_cmd, buffer);

        Wstring wcb(cmd);
        p = wcb.get_string();

        while (*p != (int) nul_char) {
                while (iswspace(*p) && (*p != (int) nul_char)) {
```

```
                        append_char(*p++, &new_cmd);
                }
                start = p;
                while (!iswspace(*p) && (*p != (int) nul_char)) {
                        p++;
                }
                cmd = GETNAME(start, p - start);
                if (cmd->has_vpath_alias_prop) {
                        cmd = get_prop(cmd->prop, vpath_alias_prop)->
                                                body.vpath_alias.alias;
                        APPEND_NAME(cmd,
                                      &new_cmd,
                                      (int) cmd->hash.length);
                } else {
                        append_string(start, &new_cmd, p - start);
                }
        }
        cmd = GETNAME(new_cmd.buffer.start, FIND_LENGTH);
        if (new_cmd.free_after_use) {
                retmem(new_cmd.buffer.start);
        }
        return cmd;
}
______unchanged_portion_omitted_

/*
 *      touch_command(line, target, result)
 *
 *      If this is an "make -t" run we do this.
 *      We touch all targets in the target group ("foo + fie:") if any.
 *
 *      Return value:
 *                              Indicates if the command failed or not
 *
 *      Parameters:
 *              line            The command line to update
 *              target          The target we are touching
 *              result          Initial value for the result we return
 *
 *      Global variables used:
 *              do_not_exec_rule Indicates that -n is on
 *              silent          Do not echo commands
 */
static Doname
touch_command(register Property line, register Name target, Doname result)
{
        Name                    name;
        register Chain          target_group;
        String_rec              touch_string;
        wchar_t                 buffer[MAXPATHLEN];
        Name                    touch_cmd;
        Cmd_line                rule;

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
        Avo_MToolJobResultMsg   *job_result_msg;
        RWCollectable           *xdr_msg;
        int                     child_pid = 0;
        wchar_t                 string[MAXPATHLEN];
        char                    mbstring[MAXPATHLEN];
        int                     filed;
#endif

        SEND_MTOOL_MSG(
                if (!sent_rsrc_info_msg) {
                        if (userName[0] == '\0') {
                                avo_get_user(userName, NULL);
                        }
                        if (hostName[0] == '\0') {
                                strcpy(hostName, avo_hostname());
                        }
                        send_rsrc_info_msg(1, hostName, userName);
                        sent_rsrc_info_msg = 1;
                }
                send_job_start_msg(line);
                job_result_msg = new Avo_MToolJobResultMsg();
        );
        for (name = target, target_group = NULL; name != NULL;) {
                if (!name->is_member) {
                        /*
                         * Build a touch command that can be passed
                         * to dosys(). If KEEP_STATE is on, "make -t"
                         * will save the proper command, not the
                         * "touch" in .make.state.
                         */
                        INIT_STRING_FROM_STACK(touch_string, buffer);
                        MBSTOWCS(wcs_buffer, NOCATGETS("touch "));
                        append_string(wcs_buffer, &touch_string, FIND_LENGTH);
                        touch_cmd = name;
                        if (name->has_vpath_alias_prop) {
                                touch_cmd = get_prop(name->prop,
                                                 vpath_alias_prop)->
                                                   body.vpath_alias.alias;
                        }
                        APPEND_NAME(touch_cmd,
                                      &touch_string,
                                      FIND_LENGTH);
                        touch_cmd = GETNAME(touch_string.buffer.start,
                                            FIND_LENGTH);
                        if (touch_string.free_after_use) {
                                retmem(touch_string.buffer.start);
                        }
                        if (!silent ||
                            do_not_exec_rule &&
                            (target_group == NULL)) {
                                (void) printf("%s\n", touch_cmd->string_mb);
                                SEND_MTOOL_MSG(
                                        job_result_msg->appendOutput(AVO_STRDUP(
                                );
                        }
                        /* Run the touch command, or simulate it */
                        if (!do_not_exec_rule) {

                                SEND_MTOOL_MSG(
                                        (void) sprintf(mbstring,
                                                        NOCATGETS("%s/make.stdou
                                                        tmpdir,
                                                        getpid(),
                                                        file_number++);

                                        int tmp_fd = mkstemp(mbstring);
                                        if(tmp_fd) {
                                                (void) close(tmp_fd);
                                        }

                                        stdout_file = strdup(mbstring);
                                        stderr_file = NULL;
                                        child_pid = pollResults(stdout_file,
                                                                (char *)NULL,
                                                                (char *)NULL);
                                );

                                result = dosys(touch_cmd,
```

```
                                               false,
                                               false,
                                               false,
                                               false,
                                               name,
                                               send_mtool_msgs);

                                SEND_MTOOL_MSG(
                                        append_job_result_msg(job_result_msg);
                                        if (child_pid > 0) {
                                                kill(child_pid, SIGUSR1);
                                                while (!((waitpid(child_pid, 0,
                                                        && (errno == ECHILD)));
                                        }
                                        child_pid = 0;
                                        (void) unlink(stdout_file);
                                        retmem_mb(stdout_file);
                                        stdout_file = NULL;
                                );

                        } else {
                                result = build_ok;
                        }
                } else {
                        result = build_ok;
                }
                if (target_group == NULL) {
                        target_group = line->body.line.target_group;
                } else {
                        target_group = target_group->next;
                }
                if (target_group != NULL) {
                        name = target_group->name;
                } else {
                        name = NULL;
                }
        }
        SEND_MTOOL_MSG(
                job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
                xdr_msg = (RWCollectable*) job_result_msg;
                xdr(&xdrs, xdr_msg);
                (void) fflush(mtool_msgs_fp);
                delete job_result_msg;
        );
        return result;
}
```

______unchanged_portion_omitted_

```
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
void
create_xdrs_ptr(void)
{
        static int      xdrs_init = 0;

        if (!xdrs_init) {
                xdrs_init = 1;
                mtool_msgs_fp = fdopen(mtool_msgs_fd, "a");
                xdrstdio_create(&xdrs,
                                mtool_msgs_fp,
                                XDR_ENCODE);
        }
}

XDR *
get_xdrs_ptr(void)
{
        return &xdrs;
}

FILE *
get_mtool_msgs_fp(void)
{
        return mtool_msgs_fp;
}

int
get_job_msg_id(void)
{
        return job_msg_id;
}

// Continuously poll and show the results of remotely executing a job,
// i.e., output the stdout and stderr files.

static int
pollResults(char *outFn, char *errFn, char *hostNm)
{
        int             child;

        child = fork();
        switch (child) {
        case -1:
                break;
        case 0:
                enable_interrupt((void (*) (int))SIG_DFL);
                (void) sigset(SIGUSR1, Avo_PollResultsAction_Sigusr1Handler);
                pollResultsAction(outFn, errFn);

                exit(0);
                break;
        default:
                break;
        }
        return child;
}

// This is the PollResultsAction SIGUSR1 handler.

static bool_t pollResultsActionTimeToFinish = FALSE;

extern "C" void
Avo_PollResultsAction_Sigusr1Handler(int foo)
{
        pollResultsActionTimeToFinish = TRUE;
}

static void
pollResultsAction(char *outFn, char *errFn)
{
        int                     fd;
        time_t                  file_time = 0;
        long                    file_time_nsec = 0;
        struct stat             statbuf;
        int                     stat_rc;

        // Keep stat'ing until file exists.
        while (((stat_rc = stat(outFn, &statbuf)) != 0) &&
               (errno == ENOENT) &&
               !pollResultsActionTimeToFinish) {
                us_sleep(STAT_RETRY_SLEEP_TIME);
        }
        // The previous stat() could be failed due to EINTR
```

```
        // So one more try is needed
        if (stat_rc != 0 && stat(outFn, &statbuf) != 0) {
                // stat() failed
                warning(NOCATGETS("Internal error: stat(\"%s\", ...) failed: %s\
                        outFn, strerror(errno));
                exit(1);
        }

        if ((fd = open(outFn, O_RDONLY)) < 0
                && (errno != EINTR || (fd = open(outFn, O_RDONLY)) < 0)) {
                // open() failed
                warning(NOCATGETS("Internal error: open(\"%s\", O_RDONLY) failed
                        outFn, strerror(errno));
                exit(1);
        }

        while (!pollResultsActionTimeToFinish && stat(outFn, &statbuf) == 0) {
                if ((statbuf.st_mtim.tv_sec > file_time) ||
                    ((statbuf.st_mtim.tv_sec == file_time) &&
                    (statbuf.st_mtim.tv_nsec > file_time_nsec))
                   ) {
                        file_time = statbuf.st_mtim.tv_sec;
                        file_time_nsec = statbuf.st_mtim.tv_nsec;
                        rxmGetNextResultsBlock(fd);
                }
                us_sleep(STAT_RETRY_SLEEP_TIME);
        }
        // Check for the rest of output
        rxmGetNextResultsBlock(fd);

        (void) close(fd);
}

static void
rxmGetNextResultsBlock(int fd)
{
        size_t                  to_read = 8 * 1024;
        ssize_t                 bytes_read;
        ssize_t                 bytes_written;
        char                    results_buf[8 * 1024];
        sigset_t                newset;
        sigset_t                oldset;

        // Read some more from the output/results file.
        // Hopefully the kernel managed to prefetch the stuff.
        bytes_read = read(fd, results_buf, to_read);
        while (bytes_read > 0) {
                AVO_BLOCK_INTERUPTS;
                bytes_written = write(1, results_buf, bytes_read);
                AVO_UNBLOCK_INTERUPTS;
                if (bytes_written != bytes_read) {
                        // write() failed
                        warning(NOCATGETS("Internal error: write(1, ...) failed:
                                strerror(errno));
                        exit(1);
                }
                bytes_read = read(fd, results_buf, to_read);
        }
}

// Generic, interruptable microsecond resolution sleep member function.

static int
us_sleep(unsigned int nusecs)
{
        struct pollfd dummy;
        int timeout;

        if ((timeout = nusecs/1000) <= 0) {
                timeout = 1;
        }
        return poll(&dummy, 0, timeout);
}
#endif /* TEAMWARE_MAKE_CMN */

// Recursively delete each of the Chain struct on the chain.

static void
delete_query_chain(Chain ch)
{
        if (ch == NULL) {
                return;
        } else {
                delete_query_chain(ch->next);
                retmem_mb((char *) ch);
        }
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    4182 Wed May 20 11:55:52 2015
new/usr/src/cmd/make/bin/dosys.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
/*
 * 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.
 */

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

/*
 * Included files
 */
#include <fcntl.h>              /* open() */
#include <mk/defs.h>
#include <mksh/dosys.h>         /* doshell(), doexec() */
#include <mksh/misc.h>          /* getmem() */
#include <sys/stat.h>           /* open() */
#include <unistd.h>             /* getpid() */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  int             filter_file;
static  char            *filter_file_name;

/*
 * File table of contents
 */
static  void            redirect_stderr(void);

/*
```

```
 *      dosys(command, ignore_error, call_make, silent_error, target)
 *
 *      Check if command string contains meta chars and dispatch to
 *      the proper routine for executing one command line.
 *
 *      Return value:
 *                              Indicates if the command execution failed
 *
 *      Parameters:
 *              command         The command to run
 *              ignore_error    Should make abort when an error is seen?
 *              call_make       Did command reference $(MAKE) ?
 *              silent_error    Should error messages be suppressed for pmake?
 *              target          Target we are building
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              working_on_targets We started processing real targets
 */
Doname
dosys(register Name command, register Boolean ignore_error, register Boolean cal
{
        timestruc_t             before;
        register int            length = command->hash.length;
        Wstring                 wcb(command);
        register wchar_t        *p = wcb.get_string();
        register wchar_t        *q;
        Doname                  result;

        /* Strip spaces from head of command string */
        while (iswspace(*p)) {
                p++, length--;
        }
        if (*p == (int) nul_char) {
                return build_failed;
        }
        /* If we are faking it we just return */
        if (do_not_exec_rule &&
            working_on_targets &&
            !call_make &&
            !always_exec) {
                return build_ok;
        }
        /* no_action_was_taken is used to print special message */
        no_action_was_taken = false;

        /* Copy string to make it OK to write it. */
        q = ALLOC_WC(length + 1);
        (void) wscpy(q, p);
        /* Write the state file iff this command uses make. */
        if (call_make && command_changed) {
                write_state_file(0, false);
        }
        make_state->stat.time = file_no_time;
        (void)exists(make_state);
        before = make_state->stat.time;
        /*
         * Run command directly if it contains no shell meta chars,
         * else run it using the shell.
         */
        if (await(ignore_error,
                  silent_error,
                  target,
                  wcb.get_string(),
                  command->meta ?
                    doshell(q, ignore_error, redirect_out_err,
                            stdout_file, stderr_file, 0) :
                    doexec(q, ignore_error, redirect_out_err,
                           stdout_file, stderr_file,
                           vroot_path, 0),
                  send_mtool_msgs,
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
                  get_xdrs_ptr(),
                  get_job_msg_id()
#else
                  NULL,
                  -1
#endif
                  )) {
                result = build_ok;
        } else {
                result = build_failed;
        }
        retmem(q);

        if ((report_dependencies_level == 0) &&
            call_make) {
                make_state->stat.time = file_no_time;
                (void)exists(make_state);
                if (before == make_state->stat.time) {
                        return result;
                }
                makefile_type = reading_statefile;
                if (read_trace_level > 1) {
                        trace_reader = true;
                }
                temp_file_number++;
                (void) read_simple_file(make_state,
                                        false,
                                        false,
                                        false,
                                        false,
                                        false,
                                        true);
                trace_reader = false;
        }
        return result;
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    4711 Wed May 20 11:55:53 2015
new/usr/src/cmd/make/bin/globals.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
/*
 * 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.
 */

/*
 *      globals.cc
 *
 *      This declares all global variables
 */

/*
 * Included files
 */
#include <nl_types.h>
#include <mk/defs.h>
#include <sys/stat.h>

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Global variables used by make only
 */
        FILE            *dependency_report_file;

/*
 * Global variables used by make
 */
        Boolean         allrules_read=false;
        Name            posix_name;
        Name            svr4_name;
        Boolean         sdot_target;    /* used to identify s.m(/M)akefile */
        Boolean         all_parallel;                   /* TEAMWARE_MAKE_CMN */
        Boolean         assign_done;
        int foo;
```

```
        Boolean         build_failed_seen;
#ifdef DISTRIBUTED
        Boolean         building_serial;
#endif
        Name            built_last_make_run;
        Name            c_at;
#ifdef DISTRIBUTED
        Boolean         called_make = false;
#endif
        Boolean         cleanup;
        Boolean         close_report;
        Boolean         command_changed;
        Boolean         commands_done;
        Chain           conditional_targets;
        Name            conditionals;
        Boolean         continue_after_error;           /* '-k' */
        Property        current_line;
        Name            current_make_version;
        Name            current_target;
        short           debug_level;
        Cmd_line        default_rule;
        Name            default_rule_name;
        Name            default_target_to_build;
        Name            dmake_group;
        Name            dmake_max_jobs;
        Name            dmake_mode;
        DMake_mode      dmake_mode_type;
        Name            dmake_output_mode;
        DMake_output_mode       output_mode = txt1_mode;
        Name            dmake_odir;
        Name            dmake_rcfile;
        Name            done;
        Name            dot;
        Name            dot_keep_state;
        Name            dot_keep_state_file;
        Name            empty_name;
        Boolean         fatal_in_progress;
        int             file_number;
#if 0
        Boolean         filter_stderr;                  /* '-X' */
#endif
        Name            force;
        Name            ignore_name;
        Boolean         ignore_errors;                  /* '-i' */
        Boolean         ignore_errors_all;              /* '-i' */
        Name            init;
        int             job_msg_id;
        Boolean         keep_state;
        Name            make_state;
#ifdef TEAMWARE_MAKE_CMN
        timestruc_t     make_state_before;
#endif
        Dependency      makefiles_used;
        Name            makeflags;
//      Boolean         make_state_locked; // Moved to lib/mksh
        Name            make_version;
        char            mbs_buffer2[(MAXPATHLEN * MB_LEN_MAX)];
        char            *mbs_ptr;
        char            *mbs_ptr2;
        int             mtool_msgs_fd;
        Boolean         depinfo_already_read = false;
        Boolean         no_action_was_taken = true;     /* true if we've not **
                                                        ** run any command   */

        Boolean         no_parallel = false;            /* TEAMWARE_MAKE_CMN */
        Name            no_parallel_name;
        Name            not_auto;
        Boolean         only_parallel;                  /* TEAMWARE_MAKE_CMN */
        Boolean         parallel;                       /* TEAMWARE_MAKE_CMN */
        Name            parallel_name;
        Name            localhost_name;
        int             parallel_process_cnt;
        Percent         percent_list;
        Dyntarget       dyntarget_list;
        Name            plus;
        Name            pmake_machinesfile;
        Name            precious;
        Name            primary_makefile;
        Boolean         quest;                          /* ‘-q’ */
        short           read_trace_level;
        Boolean         reading_dependencies = false;
        Name            recursive_name;
        int             recursion_level;
        short           report_dependencies_level = 0;  /* -P */
        Boolean         report_pwd;
        Boolean         rewrite_statefile;
        Running         running_list;
        char            *sccs_dir_path;
        Name            sccs_get_name;
        Name            sccs_get_posix_name;
        Cmd_line        sccs_get_rule;
        Cmd_line        sccs_get_org_rule;
        Cmd_line        sccs_get_posix_rule;
        Name            get_name;
        Cmd_line        get_rule;
        Name            get_posix_name;
        Cmd_line        get_posix_rule;
        Boolean         send_mtool_msgs;                /* ‘-K’ */
        Boolean         all_precious;
        Boolean         silent_all;                     /* ‘-s’ */
        Boolean         report_cwd;                     /* ‘-w’ */
        Boolean         silent;                         /* ‘-s’ */
        Name            silent_name;
        char            *stderr_file = NULL;
        char            *stdout_file = NULL;
        Boolean         stdout_stderr_same;
        Dependency      suffixes;
        Name            suffixes_name;
        Name            sunpro_dependencies;
        Boolean         target_variants;
        const char      *tmpdir = NOCATGETS("/tmp");
        const char      *temp_file_directory = NOCATGETS(".");
        Name            temp_file_name;
        short           temp_file_number;
        time_t          timing_start;
        wchar_t         *top_level_target;
        Boolean         touch;                          /* ‘-t’ */
        Boolean         trace_reader;                   /* ‘-D’ */
        Boolean         build_unconditional;            /* ‘-u’ */
        pathpt          vroot_path = VROOT_DEFAULT;
        Name            wait_name;
        wchar_t         wcs_buffer2[MAXPATHLEN];
        wchar_t         *wcs_ptr;
        wchar_t         *wcs_ptr2;
        nl_catd         catd;
        long int        hostid;

/*
 * File table of contents
 */
```

```
**********************************************************
    4000 Wed May 20 11:55:55 2015
new/usr/src/cmd/make/bin/macro.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
/*
 * 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.
 */

/*
 *      macro.cc
 *
 *      Handle expansion of make macros
 */

/*
 * Included files
 */
#ifdef DISTRIBUTED
#include <avo/strings.h>        /* AVO_STRDUP() */
#include <dm/Avo_DoJobMsg.h>
#endif
#include <mk/defs.h>
#include <mksh/macro.h>         /* getvar(), expand_value() */
#include <mksh/misc.h>          /* getmem() */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
 */

void
setvar_append(register Name name, register Name value)
{
```

```
        register Property       macro_apx = get_prop(name->prop, macro_append_pr
        register Property       macro = get_prop(name->prop, macro_prop);
        int                     length;
        String_rec              destination;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register Chain          chain;
        Name                    val = NULL;

        if(macro_apx == NULL) {
                macro_apx = append_prop(name, macro_append_prop);
                if(macro != NULL) {
                        macro_apx->body.macro_appendix.value = macro->body.macro
                }
        }

        val = macro_apx->body.macro_appendix.value_to_append;

        INIT_STRING_FROM_STACK(destination, buffer);
        buffer[0] = 0;
        if (val != NULL) {
                APPEND_NAME(val,
                              &destination,
                              (int) val->hash.length);
                if (value != NULL) {
                        MBTOWC(wcs_buffer, " ");
                        append_char(wcs_buffer[0], &destination);
                }
        }
        if (value != NULL) {
                APPEND_NAME(value,
                              &destination,
                              (int) value->hash.length);
        }
        value = GETNAME(destination.buffer.start, FIND_LENGTH);
        if (destination.free_after_use) {
                retmem(destination.buffer.start);
        }
        macro_apx->body.macro_appendix.value_to_append = value;

        SETVAR(name, empty_name, true);
}

/*
 *      setvar_envvar()
 *
 *      This function scans the list of environment variables that have
 *      dynamic values and sets them.
 *
 *      Parameters:
 *
 *      Global variables used:
 *              envvar          A list of environment vars with $ in value
 */
void
#ifdef DISTRIBUTED
setvar_envvar(Avo_DoJobMsg *dmake_job_msg)
#else
setvar_envvar(void)
#endif
{
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        int                     length;
#ifdef DISTRIBUTED
        Property                macro;
#endif
        register        char    *mbs, *tmp_mbs_buffer = NULL;
        register        char    *env, *tmp_mbs_buffer2 = NULL;
        Envvar                  p;
        String_rec              value;

        for (p = envvar; p != NULL; p = p->next) {
                if (p->already_put
#ifdef DISTRIBUTED
                    && !dmake_job_msg
#endif
                    ) {
                        continue;
                }
                INIT_STRING_FROM_STACK(value, buffer);
                expand_value(p->value, &value, false);
                if ((length = wslen(value.buffer.start)) >= MAXPATHLEN) {
                        mbs = tmp_mbs_buffer = getmem((length + 1) * MB_LEN_MAX)
                        (void) wcstombs(mbs,
                                        value.buffer.start,
                                        (length + 1) * MB_LEN_MAX);
                } else {
                        mbs = mbs_buffer;
                        WCSTOMBS(mbs, value.buffer.start);
                }
                length = 2 + strlen(p->name->string_mb) + strlen(mbs);
                if (!p->already_put || length > (MAXPATHLEN * MB_LEN_MAX)) {
                        env = tmp_mbs_buffer2 = getmem(length);
                } else {
                        env = mbs_buffer2;
                }
                (void) sprintf(env,
                               "%s=%s",
                               p->name->string_mb,
                               mbs);
                if (!p->already_put) {
                        (void) putenv(env);
                        p->already_put = true;
                        if (p->env_string) {
                                retmem_mb(p->env_string);
                        }
                        p->env_string = env;
                        tmp_mbs_buffer2 = NULL; // We should not return this mem
                }
#ifdef DISTRIBUTED
                if (dmake_job_msg) {
                        dmake_job_msg->appendVar(env);
                }
#endif
                if (tmp_mbs_buffer2) {
                        retmem_mb(tmp_mbs_buffer2);
                        tmp_mbs_buffer2 = NULL;
                }
                if (tmp_mbs_buffer) {
                        retmem_mb(tmp_mbs_buffer);
                        tmp_mbs_buffer = NULL;
                }
        }
#ifdef DISTRIBUTED
        /* Append SUNPRO_DEPENDENCIES to the dmake_job_msg. */
        if (keep_state && dmake_job_msg) {
                macro = get_prop(sunpro_dependencies->prop, macro_prop);
                length = 2 +
                         strlen(sunpro_dependencies->string_mb) +
                         strlen(macro->body.macro.value->string_mb);
                if (length > (MAXPATHLEN * MB_LEN_MAX)) {
                        env = tmp_mbs_buffer2 = getmem(length);
                } else {
```

```
                        env = mbs_buffer2;
                }
                (void) sprintf(env,
                               "%s=%s",
                               sunpro_dependencies->string_mb,
                               macro->body.macro.value->string_mb);
                dmake_job_msg->appendVar(env);
                if (tmp_mbs_buffer2) {
                        retmem_mb(tmp_mbs_buffer2);
                        tmp_mbs_buffer2 = NULL;
                }
        }
#endif
}
```
______unchanged_portion_omitted_

```
************************************************************
   94173 Wed May 20 11:55:55 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
************************************************************
/*
 * 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.
 */

/*
 *      main.cc
 *
 *      make program main routine plus some helper routines
 */

/*
 * Included files
 */
#if defined(TEAMWARE_MAKE_CMN)
#       include <avo/intl.h>
#endif

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

#ifdef DISTRIBUTED
#       include <dm/Avo_AcknowledgeMsg.h>
#       include <rw/xdrstrea.h>
#       include <dmrc/dmrc.h> /* dmakerc file processing */
#endif

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

#if defined(TEAMWARE_MAKE_CMN)
#endif

#include <pwd.h>                /* getpwnam() */
#include <setjmp.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/errno.h>          /* ENOENT */
#include <sys/stat.h>           /* fstat() */
```

```
#include <fcntl.h>              /* open() */

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

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

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

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

#ifdef MAXJOBS_ADJUST_RFE4694000
extern void job_adjust_fini();
#endif /* MAXJOBS_ADJUST_RFE4694000 */
#endif /* TEAMWARE_MAKE_CMN */


/*
 * Defined macros
 */
#define MAKE_PREFIX             NOCATGETS("/usr")
#define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT_32")
#define LD_SUPPORT_ENV_VAR_32   NOCATGETS("SGS_SUPPORT_32")
#define LD_SUPPORT_ENV_VAR_64   NOCATGETS("SGS_SUPPORT_64")
#define LD_SUPPORT_MAKE_LIB     NOCATGETS("libmakestate.so.1")
#define LD_SUPPORT_MAKE_LIB_DIR NOCATGETS("/lib")
#define LD_SUPPORT_MAKE_LIB_DIR_64      NOCATGETS("/64")

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  char            *argv_zero_string;
static  Boolean         build_failed_ever_seen;
static  Boolean         continue_after_error_ever_seen; /* '-k' */
static  Boolean         dmake_group_specified;          /* '-g' */
static  Boolean         dmake_max_jobs_specified;       /* '-j' */
static  Boolean         dmake_mode_specified;           /* '-m' */
static  Boolean         dmake_add_mode_specified;       /* '-x' */
static  Boolean         dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE
static  Boolean         dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M
static  Boolean         dmake_odir_specified;           /* '-o' */
static  Boolean         dmake_rcfile_specified;         /* '-c' */
static  Boolean         env_wins;                       /* '-e' */
static  Boolean         ignore_default_mk;              /* '-r' */
static  Boolean         list_all_targets;               /* '-T' */
static  int             mf_argc;
static  char            **mf_argv;
static  Dependency_rec  not_auto_depen_struct;
static  Dependency      not_auto_depen = &not_auto_depen_struct;
static  Boolean         pmake_cap_r_specified;          /* '-R' */
static  Boolean         pmake_machinesfile_specified;   /* '-M' */
static  Boolean         stop_after_error_ever_seen;     /* '-S' */
static  Boolean         trace_status;                   /* '-p' */

#ifdef DMAKE_STATISTICS
static  Boolean         getname_stat = false;
#endif
```

```
#if defined(TEAMWARE_MAKE_CMN)
        static  time_t          start_time;
        static  int             g_argc;
        static  char            **g_argv;
#endif

/*
 * File table of contents
 */
        extern "C" void         cleanup_after_exit(void);

#ifdef TEAMWARE_MAKE_CMN
extern "C" {
        extern  void            dmake_exit_callback(void);
        extern  void            dmake_message_callback(char *);
}
#endif

extern  Name            normalize_name(register wchar_t *name_string, register i

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

static  void            append_makeflags_string(Name, String);
static  void            doalarm(int);
static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
static  void            make_targets(int, char **, Boolean);
static  int             parse_command_option(char);
static  void            read_command_options(int, char **);
static  void            read_environment(Boolean);
static  void            read_files_and_state(int, char **);
static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
static  void            report_recursion(Name);
static  void            set_sgs_support(void);
static  void            setup_for_projectdir(void);
static  void            setup_makeflags_argv(void);
static  void            report_dir_enter_leave(Boolean entering);

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

#ifdef DISTRIBUTED
        extern  int             dmake_ofd;
        extern  FILE*           dmake_ofp;
        extern  int             rxmPid;
        extern  XDR             xdrs_out;
#endif
#ifdef TEAMWARE_MAKE_CMN
        static const char       verstring[] = "illumos make";
#endif

jmp_buf jmpbuffer;
extern nl_catd catd;

/*
 *      main(argc, argv)
 *
 *      Parameters:
 *              argc                    You know what this is
 *              argv                    You know what this is
 *
 *      Static variables used:
 *              list_all_targets        make -T seen
 *              trace_status            make -p seen
 *
 *      Global variables used:
 *              debug_level             Should we trace make actions?
```

```
 *              keep_state              Set if .KEEP_STATE seen
 *              makeflags               The Name "MAKEFLAGS", used to get macro
 *              remote_command_name     Name of remote invocation cmd ("on")
 *              running_list            List of parallel running processes
 *              stdout_stderr_same      true if stdout and stderr are the same
 *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
 *              temp_file_directory     Set to the dir where we create tmp file
 *              trace_reader            Set to reflect tracing status
 *              working_on_targets      Set when building user targets
 */
int
main(int argc, char *argv[])
{
        /*
         * cp is a -> to the value of the MAKEFLAGS env var,
         * which has to be regular chars.
         */
        register char           *cp;
        char                    make_state_dir[MAXPATHLEN];
        Boolean                 parallel_flag = false;
        char                    *prognameptr;
        char                    *slash_ptr;
        mode_t                  um;
        int                     i;
#ifdef TEAMWARE_MAKE_CMN
        struct itimerval        value;
        char                    def_dmakerc_path[MAXPATHLEN];
        Name                    dmake_name, dmake_name2;
        Name                    dmake_value, dmake_value2;
        Property                prop, prop2;
        struct stat             statbuf;
        int                     statval;
#endif

        struct stat             out_stat, err_stat;
        hostid = gethostid();
        bsd_signals();

        (void) setlocale(LC_ALL, "");


#ifdef DMAKE_STATISTICS
        if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
                getname_stat = true;
        }
#endif

#if defined(TEAMWARE_MAKE_CMN)
        catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
#endif

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


#if defined(TEAMWARE_MAKE_CMN)
#if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
/*
 * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
 * from avo_util library.
 */
        libmksdmsi18n_init();
#endif


#ifndef TEAMWARE_MAKE_CMN
        textdomain(NOCATGETS("SUNW_SPRO_MAKE"));
```

```
#endif /* TEAMWARE_MAKE_CMN */

#ifdef TEAMWARE_MAKE_CMN
        g_argc = argc;
        g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
        for (i = 0; i < argc; i++) {
                g_argv[i] = argv[i];
        }
        g_argv[i] = NULL;
#endif /* TEAMWARE_MAKE_CMN */

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

        if (*argv[0] == (int) slash_char) {
                /* argv[0] starts with a slash */
                argv_zero_string = strdup(argv[0]);
        } else if (strchr(argv[0], (int) slash_char) == NULL) {
                /* argv[0] contains no slashes */
                argv_zero_string = strdup(argv[0]);
        } else {
                /*
                 * argv[0] contains at least one slash,
                 * but doesn't start with a slash
                 */
                char    *tmp_current_path;
                char    *tmp_string;

                tmp_current_path = get_current_path();
                tmp_string = getmem(strlen(tmp_current_path) + 1 +
                                    strlen(argv[0]) + 1);
                (void) sprintf(tmp_string,
                               "%s/%s",
                               tmp_current_path,
                               argv[0]);
                argv_zero_string = strdup(tmp_string);
                retmem_mb(tmp_string);
        }

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


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

        /* Sun OS make standart */
        svr4 = false;
        posix = false;
        if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
                svr4 = false;
                posix = true;
        } else {
```

```
                prognameptr = strrchr(argv[0], '/');
                if(prognameptr) {
                        prognameptr++;
                } else {
                        prognameptr = argv[0];
                }
                if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
                        svr4 = true;
                        posix = false;
                }
        }
        if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
           svr4 = true;
           posix = false;
        }

        /*
         * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
         */
        char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
        if (dmake_compat_mode_var != NULL) {
                if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
                        gnu_style = true;
                }
                //svr4 = false;
                //posix = false;
        }

        /*
         * Temporary directory set up.
         */
        char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
        if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
                strcpy(mbs_buffer, tmpdir_var);
                for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
                        *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
                        *tmpdir_var = '\0');
                if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
                        sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
                                mbs_buffer, getpid());
                        int fd = mkstemp(mbs_buffer2);
                        if (fd >= 0) {
                                close(fd);
                                unlink(mbs_buffer2);
                                tmpdir = strdup(mbs_buffer);
                        }
                }
        }

        /* find out if stdout and stderr point to the same place */
        if (fstat(1, &out_stat) < 0) {
                fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
        }
        if (fstat(2, &err_stat) < 0) {
                fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
        }
        if ((out_stat.st_dev == err_stat.st_dev) &&
            (out_stat.st_ino == err_stat.st_ino)) {
                stdout_stderr_same = true;
        } else {
                stdout_stderr_same = false;
        }
        /* Make the vroot package scan the path using shell semantics */
        set_path_style(0);

        setup_char_semantics();
        setup_for_projectdir();

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

        load_cached_names();

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

        setup_interrupt(handle_interrupt);

        read_files_and_state(argc, argv);

#ifdef TEAMWARE_MAKE_CMN
        /*
         * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
         */
        MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
        dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
        prop2 = get_prop(dmake_name2->prop, macro_prop);
        if (prop2 == NULL) {
                /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
                output_mode = txt1_mode;
        } else {
                dmake_value2 = prop2->body.macro.value;
                if ((dmake_value2 == NULL) ||
                    (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
                        output_mode = txt1_mode;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
                        output_mode = txt2_mode;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
                        output_mode = html1_mode;
                } else {
                        warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
                              dmake_value2->string_mb);
                }
        }
        /*
         * Find the dmake_mode: distributed, parallel, or serial.
         */
    if ((!pmake_cap_r_specified) &&
        (!pmake_machinesfile_specified)) {
        MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
        dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
        prop2 = get_prop(dmake_name2->prop, macro_prop);
        if (prop2 == NULL) {
                /* DMAKE_MODE not defined, default to distributed mode */
                dmake_mode_type = distributed_mode;
                no_parallel = false;
        } else {
                dmake_value2 = prop2->body.macro.value;
                if ((dmake_value2 == NULL) ||
                    (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
```

```
                        dmake_mode_type = distributed_mode;
                        no_parallel = false;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                } else {
                        fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
                }
        }

        if ((!list_all_targets) &&
            (report_dependencies_level == 0)) {
                /*
                 * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
                 * They could be defined in the env, in the makefile, or on the
                 * command line.
                 * If neither is defined, and $(HOME)/.dmakerc does not exists,
                 * then print a message, and default to parallel mode.
                 */
#ifdef DISTRIBUTED
                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
                dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
                dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
                if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
                     ((dmake_value = prop->body.macro.value) == NULL)) &&
                    (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
                     ((dmake_value2 = prop2->body.macro.value) == NULL))) {
                        Boolean empty_dmakerc = true;
                        char *homedir = getenv(NOCATGETS("HOME"));
                        if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
                                sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
                                if ((((statval = stat(def_dmakerc_path, &statbuf
                                        ((statval == 0) && (statbuf.st_size == 0
                                } else {
                                        Avo_dmakerc     *rcfile = new Avo_dmaker
                                        Avo_err         *err = rcfile->read(def_
                                        if (err) {
                                                fatal(err->str);
                                        }
                                        empty_dmakerc = rcfile->was_empty();
                                        delete rcfile;
                                }
                        }
                        if (empty_dmakerc) {
                                if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
                                        putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
                                        (void) fprintf(stdout, catgets(catd, 1,
                                        (void) fprintf(stdout, catgets(catd, 1,
                                }
                                dmake_mode_type = parallel_mode;
                                no_parallel = false;
                        }
                }
#else
                if(dmake_mode_type == distributed_mode) {
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                }
#endif  /* DISTRIBUTED */
        }
    }
#endif
```

```
#ifdef TEAMWARE_MAKE_CMN
        parallel_flag = true;
        putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));

//
// If dmake is running with -t option, set dmake_mode_type to serial.
// This is done because doname() calls touch_command() that runs serially.
// If we do not do that, maketool will have problems.
//
        if(touch) {
                dmake_mode_type = serial_mode;
                no_parallel = true;
        }
#else
        parallel_flag = false;
#endif

#if defined (TEAMWARE_MAKE_CMN)
        /*
         * Check whether stdout and stderr are physically same.
         * This is in order to decide whether we need to redirect
         * stderr separately from stdout.
         * This check is performed only if __DMAKE_SEPARATE_STDERR
         * is not set. This variable may be used in order to preserve
         * the 'old' behaviour.
         */
        out_err_same = true;
        char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
        if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
                struct stat stdout_stat;
                struct stat stderr_stat;
                if( (fstat(1, &stdout_stat) == 0)
                 && (fstat(2, &stderr_stat) == 0) )
                {
                        if( (stdout_stat.st_dev != stderr_stat.st_dev)
                         || (stdout_stat.st_ino != stderr_stat.st_ino) )
                        {
                                out_err_same = false;
                        }
                }
        }
#endif

#ifdef DISTRIBUTED
        /*
         * At this point, DMake should startup an rxm with any and all
         * DMake command line options. Rxm will, among other things,
         * read the rc file.
         */
        if ((!list_all_targets) &&
            (report_dependencies_level == 0) &&
            (dmake_mode_type == distributed_mode)) {
                startup_rxm();
        }
#endif

/*
 *      Enable interrupt handler for alarms
 */
        (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

/*
 *      Check if make should report
 */
        if (getenv(sunpro_dependencies->string_mb) != NULL) {
```

```
                FILE    *report_file;

                report_dependency("");
                report_file = get_report_file();
                if ((report_file != NULL) && (report_file != (FILE*)-1)) {
                        (void) fprintf(report_file, "\n");
                }
        }

/*
 *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
 */
        if (keep_state) {
                maybe_append_prop(sunpro_dependencies, macro_prop)->
                  body.macro.exported = true;
        } else {
                maybe_append_prop(sunpro_dependencies, macro_prop)->
                  body.macro.exported = false;
        }

        working_on_targets = true;
        if (trace_status) {
                dump_make_state();
                fclose(stdout);
                fclose(stderr);
                exit_status = 0;
                exit(0);
        }
        if (list_all_targets) {
                dump_target_list();
                fclose(stdout);
                fclose(stderr);
                exit_status = 0;
                exit(0);
        }
        trace_reader = false;

        /*
         * Set temp_file_directory to the directory the .make.state
         * file is written to.
         */
        if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
                temp_file_directory = strdup(get_current_path());
        } else {
                *slash_ptr = (int) nul_char;
                (void) strcpy(make_state_dir, make_state->string_mb);
                *slash_ptr = (int) slash_char;
                   /* when there is only one slash and it's the first
                   ** character, make_state_dir should point to '/'.
                   */
                if(make_state_dir[0] == '\0') {
                   make_state_dir[0] = '/';
                   make_state_dir[1] = '\0';
                }
                if (make_state_dir[0] == (int) slash_char) {
                        temp_file_directory = strdup(make_state_dir);
                } else {
                        char    tmp_current_path2[MAXPATHLEN];

                        (void) sprintf(tmp_current_path2,
                                       "%s/%s",
                                       get_current_path(),
                                       make_state_dir);
                        temp_file_directory = strdup(tmp_current_path2);
                }
        }
```

```
#ifdef DISTRIBUTED
        building_serial = false;
#endif

        report_dir_enter_leave(true);

        make_targets(argc, argv, parallel_flag);

        report_dir_enter_leave(false);

        if (build_failed_ever_seen) {
                if (posix) {
                        exit_status = 1;
                }
                exit(1);
        }
        exit_status = 0;
        exit(0);
        /* NOTREACHED */
}
______unchanged_portion_omitted_
#endif

/*
#ifdef DISTRIBUTED
    if (get_parent() == TRUE) {
#endif
 */

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

                MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
                failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
                if ((exit_status != 0) && (failed_name->prop != NULL)) {
#ifdef TEAMWARE_MAKE_CMN
                        /*
                         * [tolik] switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
#endif
                        (void) doname(failed_name, false, true);
                } else {
                    if (!trace_status) {
#ifdef TEAMWARE_MAKE_CMN
                        /*
                         * Switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
#endif
                        (void) doname(done, false, true);
                    }
                }
            }
        }
        /*
         * Remove the temp file utilities report dependencies thru if it
         * is still around
         */
```

```
        if (temp_file_name != NULL) {
                (void) unlink(temp_file_name->string_mb);
        }
        /*
         * Do not save the current command in .make.state if make
         * was interrupted.
         */
        if (current_line != NULL) {
                command_changed = true;
                current_line->body.line.command_used = NULL;
        }
        /*
         * For each parallel build process running, remove the temp files
         * and zap the command line so it won't be put in .make.state
         */
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->temp_file != NULL) {
                        (void) unlink(rp->temp_file->string_mb);
                }
                if (rp->stdout_file != NULL) {
                        (void) unlink(rp->stdout_file);
                        retmem_mb(rp->stdout_file);
                        rp->stdout_file = NULL;
                }
                if (rp->stderr_file != NULL) {
                        (void) unlink(rp->stderr_file);
                        retmem_mb(rp->stderr_file);
                        rp->stderr_file = NULL;
                }
                command_changed = true;
/*
                line = get_prop(rp->target->prop, line_prop);
                if (line != NULL) {
                        line->body.line.command_used = NULL;
                }
 */
        }
        /* Remove the statefile lock file if the file has been locked */
        if ((make_state_lockfile != NULL) && (make_state_locked)) {
                (void) unlink(make_state_lockfile);
                make_state_lockfile = NULL;
                make_state_locked = false;
        }
        /* Write .make.state */
        write_state_file(1, (Boolean) 1);
/*
#ifdef DISTRIBUTED
    }
#endif
 */

#if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
        job_adjust_fini();
#endif

#ifdef TEAMWARE_MAKE_CMN
        catclose(catd);
#endif
#ifdef DISTRIBUTED
        if (rxmPid > 0) {
                // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
                Avo_AcknowledgeMsg acknowledgeMsg;
                RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

                int xdrResult = xdr(&xdrs_out, msg);
                if (xdrResult) {
                        fflush(dmake_ofp);
                } else {
/*
                        fatal(catgets(catd, 1, 266, "couldn’t tell rxm to exit")
 */
                        kill(rxmPid, SIGTERM);
                }

                waitpid(rxmPid, NULL, 0);
                rxmPid = 0;
        }
#endif
}

/*
 *      handle_interrupt()
 *
 *      This is where C-C traps are caught.
 *
 *      Parameters:
 *
 *      Global variables used (except DMake 1.0):
 *              current_target          Sometimes the current target is removed
 *              do_not_exec_rule        But not if -n is on
 *              quest                   or -q
 *              running_list            List of parallel running processes
 *              touch                   Current target is not removed if -t on
 */
void
handle_interrupt(int)
{
        Property                member;
        Running                 rp;

        (void) fflush(stdout);
#ifdef DISTRIBUTED
        if (rxmPid > 0) {
                // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
                Avo_AcknowledgeMsg acknowledgeMsg;
                RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

                int xdrResult = xdr(&xdrs_out, msg);

                if (xdrResult) {
                        fflush(dmake_ofp);
                } else {
                        kill(rxmPid, SIGTERM);
                        rxmPid = 0;
                }
        }
#endif
        if (childPid > 0) {
                kill(childPid, SIGTERM);
                childPid = -1;
        }
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->state != build_running) {
                        continue;
                }
                if (rp->pid > 0) {
                        kill(rp->pid, SIGTERM);
                        rp->pid = -1;
                }
        }
        if (getpid() == getpgrp()) {
```

```
                bsd_signal(SIGTERM, SIG_IGN);
                kill (-getpid(), SIGTERM);
        }
#ifdef TEAMWARE_MAKE_CMN
        /* Clean up all parallel/distributed children already finished */
        finish_children(false);
#endif

        /* Make sure the processes running under us terminate first */

        while (wait((int *) NULL) != -1);
        /* Delete the current targets unless they are precious */
        if ((current_target != NULL) &&
            current_target->is_member &&
            ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
                current_target = member->body.member.library;
        }
        if (!do_not_exec_rule &&
            !touch &&
            !quest &&
            (current_target != NULL) &&
            !(current_target->stat.is_precious || all_precious)) {

/* BID_1030811 */
/* azv 16 Oct 95 */
                current_target->stat.time = file_no_time;

                if (exists(current_target) != file_doesnt_exist) {
                        (void) fprintf(stderr,
                                       "\n*** %s ",
                                       current_target->string_mb);
                        if (current_target->stat.is_dir) {
                                (void) fprintf(stderr,
                                               catgets(catd, 1, 168, "not remove
                                               current_target->string_mb);
                        } else if (unlink(current_target->string_mb) == 0) {
                                (void) fprintf(stderr,
                                               catgets(catd, 1, 169, "removed.\n
                                               current_target->string_mb);
                        } else {
                                (void) fprintf(stderr,
                                               catgets(catd, 1, 170, "could not
                                               current_target->string_mb,
                                               errmsg(errno));
                        }
                }
        }
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->state != build_running) {
                        continue;
                }
                if (rp->target->is_member &&
                    ((member = get_prop(rp->target->prop, member_prop)) !=
                     NULL)) {
                        rp->target = member->body.member.library;
                }
                if (!do_not_exec_rule &&
                    !touch &&
                    !quest &&
                    !(rp->target->stat.is_precious || all_precious)) {

                        rp->target->stat.time = file_no_time;
                        if (exists(rp->target) != file_doesnt_exist) {
                                (void) fprintf(stderr,
                                               "\n*** %s ",
                                               rp->target->string_mb);
                                if (rp->target->stat.is_dir) {
                                        (void) fprintf(stderr,
                                                       catgets(catd, 1, 171, "no
                                                       rp->target->string_mb);
                                } else if (unlink(rp->target->string_mb) == 0) {
                                        (void) fprintf(stderr,
                                                       catgets(catd, 1, 172, "re
                                                       rp->target->string_mb);
                                } else {
                                        (void) fprintf(stderr,
                                                       catgets(catd, 1, 173, "co
                                                       rp->target->string_mb,
                                                       errmsg(errno));
                                }
                        }
                }
        }


        /* Have we locked .make.state or .nse_depinfo? */
        if ((make_state_lockfile != NULL) && (make_state_locked)) {
                unlink(make_state_lockfile);
                make_state_lockfile = NULL;
                make_state_locked = false;
        }
        /*
         * Re-read .make.state file (it might be changed by recursive make)
         */
        check_state(NULL);

        report_dir_enter_leave(false);

        exit_status = 2;
        exit(2);
}
______unchanged_portion_omitted_

/*
 *      parse_command_option(ch)
 *
 *      Parse make command line options.
 *
 *      Return value:
 *                              Indicates if any -f -c or -M were seen
 *
 *      Parameters:
 *              ch              The character to parse
 *
 *      Static variables used:
 *              dmake_group_specified   Set for make -g
 *              dmake_max_jobs_specified        Set for make -j
 *              dmake_mode_specified    Set for make -m
 *              dmake_add_mode_specified        Set for make -x
 *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
 *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
 *              dmake_odir_specified    Set for make -o
 *              dmake_rcfile_specified  Set for make -c
 *              env_wins                Set for make -e
 *              ignore_default_mk       Set for make -r
 *              trace_status            Set for make -p
 *
 *      Global variables used:
 *              .make.state path & name set for make -K
 *              continue_after_error    Set for make -k
 *              debug_level             Set for make -d
 *              do_not_exec_rule        Set for make -n
```

```
 *              filter_stderr           Set for make -X
 *              ignore_errors_all       Set for make -i
 *              no_parallel             Set for make -R
 *              quest                   Set for make -q
 *              read_trace_level        Set for make -D
 *              report_dependencies     Set for make -P
 *              send_mtool_msgs         Set for make -K
 *              silent_all              Set for make -s
 *              touch                   Set for make -t
 */
static int
parse_command_option(register char ch)
{
        static int              invert_next = 0;
        int                     invert_this = invert_next;

        invert_next = 0;
        switch (ch) {
        case '-':                        /* Ignore "--" */
                return 0;
        case '~':                        /* Invert next option */
                invert_next = 1;
                return 0;
        case 'B':                        /* Obsolete */
                return 0;
        case 'b':                        /* Obsolete */
                return 0;
        case 'c':                        /* Read alternative dmakerc file */
                if (invert_this) {
                        dmake_rcfile_specified = false;
                } else {
                        dmake_rcfile_specified = true;
                }
                return 2;
        case 'D':                        /* Show lines read */
                if (invert_this) {
                        read_trace_level--;
                } else {
                        read_trace_level++;
                }
                return 0;
        case 'd':                        /* Debug flag */
                if (invert_this) {
                        debug_level--;
                } else {
                        debug_level++;
                }
                return 0;
        case 'e':                        /* Environment override flag */
                if (invert_this) {
                        env_wins = false;
                } else {
                        env_wins = true;
                }
                return 0;
        case 'f':                        /* Read alternative makefile(s) */
                return 1;
        case 'g':                        /* Use alternative DMake group */
                if (invert_this) {
                        dmake_group_specified = false;
                } else {
                        dmake_group_specified = true;
                }
                return 4;
        case 'i':                        /* Ignore errors */
                if (invert_this) {
                        ignore_errors_all = false;
                } else {
                        ignore_errors_all = true;
                }
                return 0;
        case 'j':                       /* Use alternative DMake max jobs */
                if (invert_this) {
                        dmake_max_jobs_specified = false;
                } else {
                        dmake_max_jobs_specified = true;
                }
                return 8;
        case 'K':                       /* Read alternative .make.state */
                return 256;
        case 'k':                       /* Keep making even after errors */
                if (invert_this) {
                        continue_after_error = false;
                } else {
                        continue_after_error = true;
                        continue_after_error_ever_seen = true;
                }
                return 0;
        case 'M':                       /* Read alternative make.machines file
                if (invert_this) {
                        pmake_machinesfile_specified = false;
                } else {
                        pmake_machinesfile_specified = true;
                        dmake_mode_type = parallel_mode;
                        no_parallel = false;
                }
                return 16;
        case 'm':                       /* Use alternative DMake build mode */
                if (invert_this) {
                        dmake_mode_specified = false;
                } else {
                        dmake_mode_specified = true;
                }
                return 32;
        case 'x':                       /* Use alternative DMake mode */
                if (invert_this) {
                        dmake_add_mode_specified = false;
                } else {
                        dmake_add_mode_specified = true;
                }
                return 1024;
        case 'N':                       /* Reverse -n */
                if (invert_this) {
                        do_not_exec_rule = true;
                } else {
                        do_not_exec_rule = false;
                }
                return 0;
        case 'n':                       /* Print, not exec commands */
                if (invert_this) {
                        do_not_exec_rule = false;
                } else {
                        do_not_exec_rule = true;
                }
                return 0;
        case 'O':                       /* Send job start & result msgs */
                if (invert_this) {
                        send_mtool_msgs = false;
                } else {
#ifdef DISTRIBUTED
                        send_mtool_msgs = true;
#endif
```

```
                }
                return 128;
        case 'o':                       /* Use alternative dmake output dir */
                if (invert_this) {
                        dmake_odir_specified = false;
                } else {
                        dmake_odir_specified = true;
                }
                return 512;
        case 'P':                       /* Print for selected targets */
                if (invert_this) {
                        report_dependencies_level--;
                } else {
                        report_dependencies_level++;
                }
                return 0;
        case 'p':                       /* Print description */
                if (invert_this) {
                        trace_status = false;
                        do_not_exec_rule = false;
                } else {
                        trace_status = true;
                        do_not_exec_rule = true;
                }
                return 0;
        case 'q':                       /* Question flag */
                if (invert_this) {
                        quest = false;
                } else {
                        quest = true;
                }
                return 0;
        case 'R':                       /* Don't run in parallel */
#ifdef TEAMWARE_MAKE_CMN
                if (invert_this) {
                        pmake_cap_r_specified = false;
                        no_parallel = false;
                } else {
                        pmake_cap_r_specified = true;
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                }
#else
                warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option")
#endif
                return 0;
        case 'r':                       /* Turn off internal rules */
                if (invert_this) {
                        ignore_default_mk = false;
                } else {
                        ignore_default_mk = true;
                }
                return 0;
        case 'S':                       /* Reverse -k */
                if (invert_this) {
                        continue_after_error = true;
                } else {
                        continue_after_error = false;
                        stop_after_error_ever_seen = true;
                }
                return 0;
        case 's':                       /* Silent flag */
                if (invert_this) {
                        silent_all = false;
                } else {
                        silent_all = true;
                }
                return 0;
        case 'T':                       /* Print target list */
                if (invert_this) {
                        list_all_targets = false;
                        do_not_exec_rule = false;
                } else {
                        list_all_targets = true;
                        do_not_exec_rule = true;
                }
                return 0;
        case 't':                       /* Touch flag */
                if (invert_this) {
                        touch = false;
                } else {
                        touch = true;
                }
                return 0;
        case 'u':                       /* Unconditional flag */
                if (invert_this) {
                        build_unconditional = false;
                } else {
                        build_unconditional = true;
                }
                return 0;
        case 'V':                      /* SVR4 mode */
                svr4 = true;
                return 0;
        case 'v':                      /* Version flag */
                if (invert_this) {
                } else {
#ifdef TEAMWARE_MAKE_CMN
                        fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
                        exit_status = 0;
                        exit(0);
#else
                        warning(catgets(catd, 1, 324, "Ignoring DistributedMake
#endif
                }
                return 0;
        case 'w':                       /* Unconditional flag */
                if (invert_this) {
                        report_cwd = false;
                } else {
                        report_cwd = true;
                }
                return 0;
#if 0
        case 'X':                      /* Filter stdout */
                if (invert_this) {
                        filter_stderr = false;
                } else {
                        filter_stderr = true;
                }
                return 0;
#endif
        default:
                break;
        }
        return 0;
}
```
______***unchanged_portion_omitted_***

```
#endif

#ifdef DISTRIBUTED
/*
```

```
 * Returns whether -c is set or not.
 */
Boolean
get_dmake_rcfile_specified(void)
{
        return(dmake_rcfile_specified);
}

/*
 * Returns whether -g is set or not.
 */
Boolean
get_dmake_group_specified(void)
{
        return(dmake_group_specified);
}

/*
 * 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);
        }
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   25623 Wed May 20 11:55:56 2015
new/usr/src/cmd/make/bin/misc.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
______unchanged_portion_omitted_

/****************************************
 *
 *      String manipulation
 */

/****************************************
 *
 *      Nameblock property handling
 */

/****************************************
 *
 *      Error message handling
 */

/*
 *      fatal(format, args...)
 *
 *      Print a message and die
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 *
 *      Global variables used:
 *              fatal_in_progress Indicates if this is a recursive call
 *              parallel_process_cnt Do we need to wait for anything?
 *              report_pwd      Should we report the current path?
 */
/*VARARGS*/
void
fatal(const char *message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
#ifdef DISTRIBUTED
        (void) fprintf(stderr, catgets(catd, 1, 262, "dmake: Fatal error: "));
#else
        (void) fprintf(stderr, catgets(catd, 1, 263, "make: Fatal error: "));
#endif
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                               catgets(catd, 1, 156, "Current working directory
                               get_current_path());
        }
        (void) fflush(stderr);
        if (fatal_in_progress) {
                exit_status = 1;
                exit(1);
        }
        fatal_in_progress = true;
#ifdef TEAMWARE_MAKE_CMN
        /* Let all parallel children finish */
        if ((dmake_mode_type == parallel_mode) &&
```

```
            (parallel_process_cnt > 0)) {
                (void) fprintf(stderr,
                               catgets(catd, 1, 157, "Waiting for %d %s to finis
                               parallel_process_cnt,
                               parallel_process_cnt == 1 ?
                               catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
                (void) fflush(stderr);
        }

        while (parallel_process_cnt > 0) {
#ifdef DISTRIBUTED
                if (dmake_mode_type == distributed_mode) {
                        (void) await_dist(false);
                } else {
                        await_parallel(true);
                }
#else
                await_parallel(true);
#endif
                finish_children(false);
        }
#endif

#if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
        job_adjust_fini();
#endif

        exit_status = 1;
        exit(1);
}

/*
 *      warning(format, args...)
 *
 *      Print a message and continue.
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 *
 *      Global variables used:
 *              report_pwd      Should we report the current path?
 */
/*VARARGS*/
void
warning(char * message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
#ifdef DISTRIBUTED
        (void) fprintf(stderr, catgets(catd, 1, 264, "dmake: Warning: "));
#else
        (void) fprintf(stderr, catgets(catd, 1, 265, "make: Warning: "));
#endif
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                               catgets(catd, 1, 161, "Current working directory
                               get_current_path());
        }
        (void) fflush(stderr);
}
______unchanged_portion_omitted_
/*
 *      get_current_path()
 *
 *      Stuff current_path with the current path if it isnt there already.
 *
 *      Parameters:
 *
 *      Global variables used:
 */
char *
get_current_path(void)
{
        char                    pwd[(MAXPATHLEN * MB_LEN_MAX)];
        static char             *current_path;

        if (current_path == NULL) {
                getcwd(pwd, sizeof(pwd));
                if (pwd[0] == (int) nul_char) {
                        pwd[0] = (int) slash_char;
                        pwd[1] = (int) nul_char;
#ifdef DISTRIBUTED
                        current_path = strdup(pwd);
                } else if (IS_EQUALN(pwd, NOCATGETS("/tmp_mnt"), 8)) {
                        current_path = strdup(pwd + 8);
                } else {
                        current_path = strdup(pwd);
                }
#else
                }
                current_path = strdup(pwd);
#endif
        }
        return current_path;
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   46861 Wed May 20 11:55:56 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
/*
 * 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 <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>



/*
 * 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)
{
        int                     argcnt;
        int                     cmd_options = 0;
        char                    *commands[MAXRULES + 5];
        char                    *cp;
#ifdef DISTRIBUTED
        Avo_DoJobMsg            *dmake_job_msg = NULL;
#endif
        Name                    dmake_name;
        Name                    dmake_value;
        int                     ignore;
        Name                    make_machines_name;
        char                    **p;
        Property                prop;
        Doname                  result = build_ok;
        Cmd_line                rule;
        Boolean                 silent_flag;
        Name                    target = line->body.line.target;
        Boolean                 wrote_state_file = false;

        if ((pmake_max_jobs == 0) &&
            (dmake_mode_type == parallel_mode)) {
                if (local_host[0] == '\0') {
                        (void) gethostname(local_host, MAXNAMELEN);
                }
                MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
                dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
                if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
                    ((dmake_value = prop->body.macro.value) != NULL)) {
                        pmake_max_jobs = atoi(dmake_value->string_mb);
                        if (pmake_max_jobs <= 0) {
                                warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
                                warning(catgets(catd, 1, 309, "setting DMAKE_MAX
                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                        }
                } else {
                        /*
                         * For backwards compatibility w/ PMake 1.x, when
                         * DMake 2.x is being run in parallel mode, DMake
                         * should parse the PMake startup file
                         * $(HOME)/.make.machines to get the pmake_max_jobs.
                         */
                        MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
                        dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
                        if (((prop = get_prop(dmake_name->prop, macro_prop)) !=
                            ((dmake_value = prop->body.macro.value) != NULL)) {
                                make_machines_name = dmake_value;
                        } else {
                                make_machines_name = NULL;
                        }
                        if ((pmake_max_jobs = read_make_machines(make_machines_n
                                pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
                        }
                }
#ifdef DISTRIBUTED
                if (send_mtool_msgs) {
                        send_rsrc_info_msg(pmake_max_jobs, local_host, user_name
                }
#endif
        }

        if ((dmake_mode_type == serial_mode) ||
            ((dmake_mode_type == parallel_mode) && (waitflg))) {
                return (execute_serial(line));
        }

#ifdef DISTRIBUTED
        if (dmake_mode_type == distributed_mode) {
```

```
                if(local) {
//                      return (execute_serial(line));
                        waitflg = true;
                }
                dmake_job_msg = new Avo_DoJobMsg();
                dmake_job_msg->setJobId(++job_msg_id);
                dmake_job_msg->setTarget(target->string_mb);
                dmake_job_msg->setImmediateOutput(0);
                called_make = false;
        } else
#endif
        {
                p = commands;
        }

        argcnt = 0;
        for (rule = line->body.line.command_used;
             rule != NULL;
             rule = rule->next) {
                if (posix && (touch || quest) && !rule->always_exec) {
                        continue;
                }
                if (vpath_defined) {
                        rule->command_line =
                          vpath_translation(rule->command_line);
                }
                if (dmake_mode_type == distributed_mode) {
                        cmd_options = 0;
                        if(local) {
                                cmd_options |= local_host_mask;
                        }
                } else {
                        silent_flag = false;
                        ignore = 0;
                }
                if (rule->command_line->hash.length > 0) {
                        if (++argcnt == MAXRULES) {
                                if (dmake_mode_type == distributed_mode) {
                                        /* XXX - tell rxm to execute on local ho
                                        /* I WAS HERE!!! */
                                } else {
                                        /* Too many rules, run serially instead.
                                        return build_serial;
                                }
                        }
#ifdef DISTRIBUTED
                        if (dmake_mode_type == distributed_mode) {
                                /*
                                 * XXX - set assign_mask to tell rxm
                                 * to do the following.
                                 */
/* From execute_serial():
                                if (rule->assign) {
                                        result = build_ok;
                                        do_assign(rule->command_line, target);
 */
                                if (0) {
                                } else if (report_dependencies_level == 0) {
                                        if (rule->ignore_error) {
                                                cmd_options |= ignore_mask;
                                        }
                                        if (rule->silent) {
                                                cmd_options |= silent_mask;
                                        }
                                        if (rule->command_line->meta) {
                                                cmd_options |= meta_mask;
                                        }
                                        if (rule->make_refd) {
                                                cmd_options |= make_refd_mask;
                                        }
                                        if (do_not_exec_rule) {
                                                cmd_options |= do_not_exec_mask;
                                        }
                                        append_dmake_cmd(dmake_job_msg,
                                                         rule->command_line->str
                                                         cmd_options);
                                        /* Copying dosys()... */
                                        if (rule->make_refd) {
                                                if (waitflg) {
                                                        dmake_job_msg->setImmedi
                                                }
                                                called_make = true;
                                                if (command_changed &&
                                                    !wrote_state_file) {
                                                        write_state_file(0, fals
                                                        wrote_state_file = true;
                                                }
                                        }
                                }
                        } else
#endif
                        {
                                if (rule->silent && !silent) {
                                        silent_flag = true;
                                }
                                if (rule->ignore_error) {
                                        ignore++;
                                }
                                /* XXX - need to add support for + prefix */
                                if (silent_flag || ignore) {
                                        *p = getmem((silent_flag ? 1 : 0) +
                                                    ignore +
                                                    (strlen(rule->
                                                           command_line->
                                                           string_mb)) +
                                                    1);
                                        cp = *p++;
                                        if (silent_flag) {
                                                *cp++ = (int) at_char;
                                        }
                                        if (ignore) {
                                                *cp++ = (int) hyphen_char;
                                        }
                                        (void) strcpy(cp, rule->command_line->st
                                } else {
                                        *p++ = rule->command_line->string_mb;
                                }
                        }
                }
        }
        if ((argcnt == 0) ||
            (report_dependencies_level > 0)) {
#ifdef DISTRIBUTED
                if (dmake_job_msg) {
                        delete dmake_job_msg;
                }
#endif
                return build_ok;
        }
#ifdef DISTRIBUTED
        if (dmake_mode_type == distributed_mode) {
                // Send  a DoJob message to the rxm process.
```

```
                distribute_rxm(dmake_job_msg);

                // Wait for an acknowledgement.
                Avo_AcknowledgeMsg *ackMsg = getAcknowledgeMsg();
                if (ackMsg) {
                        delete ackMsg;
                }

                if (waitflg) {
                        // Wait for, and process a job result.
                        result = await_dist(waitflg);
                        if (called_make) {
                                maybe_reread_make_state();
                        }
                        check_state(temp_file_name);
                        if (result == build_failed) {
                                if (!continue_after_error) {

#ifdef PRINT_EXIT_STATUS
                                        warning(NOCATGETS("I'm in execute_parall
#endif

                                        fatal(catgets(catd, 1, 252, "Command fai
                                              target->string_mb);
                                }
                                /*
                                 * Make sure a failing command is not
                                 * saved in .make.state.
                                 */
                                line->body.line.command_used = NULL;
                        }
                        if (temp_file_name != NULL) {
                                free_name(temp_file_name);
                        }
                        temp_file_name = NULL;
                        Property spro = get_prop(sunpro_dependencies->prop, macr
                        if(spro != NULL) {
                                Name val = spro->body.macro.value;
                                if(val != NULL) {
                                        free_name(val);
                                        spro->body.macro.value = NULL;
                                }
                        }
                        spro = get_prop(sunpro_dependencies->prop, env_mem_prop)
                        if(spro) {
                                char *val = spro->body.env_mem.value;
                                if(val != NULL) {
                                        retmem_mb(val);
                                        spro->body.env_mem.value = NULL;
                                }
                        }
                        return result;
                } else {
                        parallel_process_cnt++;
                        return build_running;
                }
        } else
#endif
        {
                *p = NULL;

                Doname res = distribute_process(commands, line);
                if (res == build_running) {
                        parallel_process_cnt++;
                }
                /*
                 * Return only those memory that were specially allocated
                 * for part of commands.
                 */
                for (int i = 0; commands[i] != NULL; i++) {
                        if ((commands[i][0] == (int) at_char) ||
                            (commands[i][0] == (int) hyphen_char)) {
                                retmem_mb(commands[i]);
                        }
                }
                return res;
        }
}

#ifdef DISTRIBUTED
/*
 *      append_dmake_cmd()
 *
 *      Replaces all escaped newline's (\<cr>)
 *        in the original command line with space's,
 *        then append the new command line to the DoJobMsg object.
 */
static void
append_dmake_cmd(Avo_DoJobMsg *dmake_job_msg,
                 char *orig_cmd_line,
                 int cmd_options)
{
/*
        Avo_DmakeCommand        *tmp_dmake_command;

        tmp_dmake_command = new Avo_DmakeCommand(orig_cmd_line, cmd_options);
        dmake_job_msg->appendCmd(tmp_dmake_command);
        delete tmp_dmake_command;
 */
        dmake_job_msg->appendCmd(new Avo_DmakeCommand(orig_cmd_line, cmd_options
}
#endif

#ifdef TEAMWARE_MAKE_CMN
#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000

#include <unistd.h>     /* sysconf(_SC_NPROCESSORS_ONLN) */
#include <sys/ipc.h>            /* ftok() */
#include <sys/shm.h>            /* shmget(), shmat(), shmdt(), shmctl() */
#include <semaphore.h>          /* sem_init(), sem_trywait(), sem_post(), sem_de
#include <sys/loadavg.h>        /* getloadavg() */

/*
 *      adjust_pmake_max_jobs (int pmake_max_jobs)
 *
 *      Parameters:
 *              pmake_max_jobs  - max jobs limit set by user
 *
 *      External functions used:
 *              sysconf()
 *              getloadavg()
 */
static int
adjust_pmake_max_jobs (int pmake_max_jobs)
{
        static int      ncpu = 0;
        double          loadavg[3];
        int             adjustment;
        int             adjusted_max_jobs;
```

```
        if (ncpu <= 0) {
                if ((ncpu = sysconf(_SC_NPROCESSORS_ONLN)) <= 0) {
                        ncpu = 1;
                }
        }
        if (getloadavg(loadavg, 3) != 3) return(pmake_max_jobs);
        adjustment = ((int)loadavg[LOADAVG_1MIN]);
        if (adjustment < 2) return(pmake_max_jobs);
        if (ncpu > 1) {
                adjustment = adjustment / ncpu;
        }
        adjusted_max_jobs = pmake_max_jobs - adjustment;
        if (adjusted_max_jobs < 1) adjusted_max_jobs = 1;
        return(adjusted_max_jobs);
}
```

______***unchanged_portion_omitted_***

```
#endif /* MAXJOBS_ADJUST_RFE4694000 */
#endif /* TEAMWARE_MAKE_CMN */

/*
 *      distribute_process(char **commands, Property line)
 *
 *      Parameters:
 *              commands        argv vector of commands to execute
 *
 *      Return value:
 *                              The result of the execution
 *
 *      Static variables used:
 *              process_running Set to the pid of the process set running
 * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
 *              job_adjust_mode Current job adjust mode
 * #endif
 */
static Doname
distribute_process(char **commands, Property line)
{
        static unsigned file_number = 0;
        wchar_t         string[MAXPATHLEN];
        char            mbstring[MAXPATHLEN];
        int             filed;
        int             res;
        int             tmp_index;
        char            *tmp_index_str_ptr;

#if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
        while (parallel_process_cnt >= pmake_max_jobs) {
                await_parallel(false);
                finish_children(true);
        }
#else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
        /* initialize adjust mode, if not initialized */
        if (job_adjust_mode == ADJUST_UNKNOWN) {
                job_adjust_init();
        }

        /* actions depend on adjust mode */
        switch (job_adjust_mode) {
        case ADJUST_M1:
                while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
                        await_parallel(false);
                        finish_children(true);
                }
                break;
        case ADJUST_M2:
                if ((res = m2_acquire_job()) == 0) {
                        if (parallel_process_cnt > 0) {
                                await_parallel(false);
                                finish_children(true);

                                if ((res = m2_acquire_job()) == 0) {
                                        return build_serial;
                                }
                        } else {
                                return build_serial;
                        }
                }
                if (res < 0) {
                        /* job adjustment error */
                        job_adjust_error();

                        /* no adjustment */
                        while (parallel_process_cnt >= pmake_max_jobs) {
                                await_parallel(false);
                                finish_children(true);
                        }
                }
                break;
        default:
                while (parallel_process_cnt >= pmake_max_jobs) {
                        await_parallel(false);
                        finish_children(true);
                }
        }
#endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
#ifdef DISTRIBUTED
        if (send_mtool_msgs) {
                send_job_start_msg(line);
        }
#endif
#ifdef DISTRIBUTED
        setvar_envvar((Avo_DoJobMsg *)NULL);
#else
        setvar_envvar();
#endif
        /*
         * Tell the user what DMake is doing.
         */
        if (!silent && output_mode != txt2_mode) {
                /*
                 * Print local_host --> x job(s).
                 */
                (void) fprintf(stdout,
                               catgets(catd, 1, 325, "%s --> %d %s\n"),
                               local_host,
                               parallel_process_cnt + 1,
                               (parallel_process_cnt == 0) ? catgets(catd, 1, 12

                /* Print command line(s). */
                tmp_index = 0;
                while (commands[tmp_index] != NULL) {
                    /* No @ char. */
                    /* XXX - need to add [2] when + prefix is added */
                    if ((commands[tmp_index][0] != (int) at_char) &&
                        (commands[tmp_index][1] != (int) at_char)) {
                        tmp_index_str_ptr = commands[tmp_index];
                        if (*tmp_index_str_ptr == (int) hyphen_char) {
                                tmp_index_str_ptr++;
                        }
                        (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
```

```
                    }
                    tmp_index++;
                }
                (void) fflush(stdout);
        }

        (void) sprintf(mbstring,
                        NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
                        tmpdir,
                        getpid(),
                        file_number++);

        mktemp(mbstring);

        stdout_file = strdup(mbstring);
        stderr_file = NULL;
#if defined (TEAMWARE_MAKE_CMN)
        if (!out_err_same) {
                (void) sprintf(mbstring,
                                NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
                                tmpdir,
                                getpid(),
                                file_number++);

                mktemp(mbstring);

                stderr_file = strdup(mbstring);
        }
#endif

        process_running = run_rule_commands(local_host, commands);

        return build_running;
}
______unchanged_portion_omitted_

/*
 *      finish_running()
 *
 *      Keeps processing until the running_list is emptied out.
 *
 *      Parameters:
 *
 *      Global variables used:
 *              running_list    The list of running processes
 */
void
finish_running(void)
{
        while (running_list != NULL) {
#ifdef DISTRIBUTED
                if (dmake_mode_type == distributed_mode) {
                        if ((just_did_subtree) ||
                            (parallel_process_cnt == 0)) {
                                just_did_subtree = false;
                        } else {
                                (void) await_dist(false);
                                finish_children(true);
                        }
                } else
#endif
                {
                        await_parallel(false);
                        finish_children(true);
                }
                if (running_list != NULL) {
                        process_next();
                }
        }
}

/*
 *      process_next()
 *
 *      Searches the running list for any targets which can start processing.
 *      This can be a pending target, a serial target, or a subtree target.
 *
 *      Parameters:
 *
 *      Static variables used:
 *              running_tail            The end of the list of running procs
 *              subtree_conflict        A target which conflicts with a subtree
 *              subtree_conflict2       The other target which conflicts
 *
 *      Global variables used:
 *              commands_done           True if commands executed
 *              debug_level             Controls debug output
 *              parallel_process_cnt    Number of parallel process running
 *              recursion_level         Indentation for debug output
 *              running_list            List of running processes
 */
static void
process_next(void)
{
        Running         rp;
        Running         *rp_prev;
        Property        line;
        Chain           target_group;
        Dependency      dep;
        Boolean         quiescent = true;
        Running         *subtree_target;
        Boolean         saved_commands_done;
        Property        *conditionals;

        subtree_target = NULL;
        subtree_conflict = NULL;
        subtree_conflict2 = NULL;
        /*
         * If nothing currently running, build a serial target, if any.
         */
start_loop_1:
        for (rp_prev = &running_list, rp = running_list;
             rp != NULL && parallel_process_cnt == 0;
             rp = rp->next) {
                if (rp->state == build_serial) {
                        *rp_prev = rp->next;
                        if (rp->next == NULL) {
                                running_tail = rp_prev;
                        }
                        recursion_level = rp->recursion_level;
                        rp->target->state = build_pending;
                        (void) doname_check(rp->target,
                                            rp->do_get,
                                            rp->implicit,
                                            false);
                        quiescent = false;
                        delete_running_struct(rp);
                        goto start_loop_1;
                } else {
                        rp_prev = &rp->next;
                }
        }
```

```
        /*
         * Find a target to build.  The target must be pending, have all
         * its dependencies built, and not be in a target group with a target
         * currently building.
         */
start_loop_2:
        for (rp_prev = &running_list, rp = running_list;
             rp != NULL;
             rp = rp->next) {
                if (!(rp->state == build_pending ||
                      rp->state == build_subtree)) {
                        quiescent = false;
                        rp_prev = &rp->next;
                } else if (rp->state == build_pending) {
                        line = get_prop(rp->target->prop, line_prop);
                        for (dep = line->body.line.dependencies;
                             dep != NULL;
                             dep = dep->next) {
                                if (dep->name->state == build_running ||
                                    dep->name->state == build_pending ||
                                    dep->name->state == build_serial) {
                                        break;
                                }
                        }
                        if (dep == NULL) {
                                for (target_group = line->body.line.target_group
                                     target_group != NULL;
                                     target_group = target_group->next) {
                                        if (is_running(target_group->name)) {
                                                break;
                                        }
                                }
                                if (target_group == NULL) {
                                        *rp_prev = rp->next;
                                        if (rp->next == NULL) {
                                                running_tail = rp_prev;
                                        }
                                        recursion_level = rp->recursion_level;
                                        rp->target->state = rp->redo ?
                                          build_dont_know : build_pending;
                                        saved_commands_done = commands_done;
                                        conditionals =
                                                set_conditionals
                                                    (rp->conditional_cnt,
                                                     rp->conditional_targets);
                                        rp->target->dont_activate_cond_values =
                                        if ((doname_check(rp->target,
                                                          rp->do_get,
                                                          rp->implicit,
                                                          rp->target->has_target
                                             build_running) &&
                                            !commands_done) {
                                                commands_done =
                                                  saved_commands_done;
                                        }
                                        rp->target->dont_activate_cond_values =
                                        reset_conditionals
                                                (rp->conditional_cnt,
                                                 rp->conditional_targets,
                                                 conditionals);
                                        quiescent = false;
                                        delete_running_struct(rp);
                                        goto start_loop_2;
                                } else {
                                        rp_prev = &rp->next;
                                }
                        } else {
                                rp_prev = &rp->next;
                        }
                } else {
                        rp_prev = &rp->next;
                }
        }
        /*
         * If nothing has been found to build and there exists a subtree
         * target with no dependency conflicts, build it.
         */
        if (quiescent) {
start_loop_3:
                for (rp_prev = &running_list, rp = running_list;
                     rp != NULL;
                     rp = rp->next) {
                        if (rp->state == build_subtree) {
                                if (!dependency_conflict(rp->target)) {
                                        *rp_prev = rp->next;
                                        if (rp->next == NULL) {
                                                running_tail = rp_prev;
                                        }
                                        recursion_level = rp->recursion_level;
                                        doname_subtree(rp->target,
                                                       rp->do_get,
                                                       rp->implicit);
#ifdef DISTRIBUTED
                                        just_did_subtree = true;
#endif
                                        quiescent = false;
                                        delete_running_struct(rp);
                                        goto start_loop_3;
                                } else {
                                        subtree_target = rp_prev;
                                        rp_prev = &rp->next;
                                }
                        } else {
                                rp_prev = &rp->next;
                        }
                }
        }
        /*
         * If still nothing found to build, we either have a deadlock
         * or a subtree with a dependency conflict with something waiting
         * to build.
         */
        if (quiescent) {
                if (subtree_target == NULL) {
                        fatal(catgets(catd, 1, 126, "Internal error: deadlock de
                } else {
                        rp = *subtree_target;
                        if (debug_level > 0) {
                                warning(catgets(catd, 1, 127, "Conditional macro
                                        subtree_conflict2->string_mb,
                                        rp->target->string_mb,
                                        subtree_conflict->string_mb);
                        }
                        *subtree_target = (*subtree_target)->next;
                        if (rp->next == NULL) {
                                running_tail = subtree_target;
                        }
                        recursion_level = rp->recursion_level;
                        doname_subtree(rp->target, rp->do_get, rp->implicit);
#ifdef DISTRIBUTED
                        just_did_subtree = true;
#endif
```

```
                        delete_running_struct(rp);
                }
        }
}
______unchanged_portion_omitted_

/*
 *      finish_children(docheck)
 *
 *      Finishes the processing for all targets which were running
 *      and have now completed.
 *
 *      Parameters:
 *              docheck         Completely check the finished target
 *
 *      Static variables used:
 *              running_tail    The tail of the running list
 *
 *      Global variables used:
 *              continue_after_error  -k flag
 *              fatal_in_progress  True if we are finishing up after fatal err
 *              running_list    List of running processes
 */
void
finish_children(Boolean docheck)
{
        int             cmds_length;
        Property        line;
        Property        line2;
        struct stat     out_buf;
        Running         rp;
        Running         *rp_prev;
        Cmd_line        rule;
        Boolean         silent_flag;

        for (rp_prev = &running_list, rp = running_list;
             rp != NULL;
             rp = rp->next) {
bypass_for_loop_inc_4:
                /*
                 * If the state is ok or failed, then this target has
                 * finished building.
                 * In parallel_mode, output the accumulated stdout/stderr.
                 * Read the auto dependency stuff, handle a failed build,
                 * update the target, then finish the doname process for
                 * that target.
                 */
                if (rp->state == build_ok || rp->state == build_failed) {
                        *rp_prev = rp->next;
                        if (rp->next == NULL) {
                                running_tail = rp_prev;
                        }
                        if ((line2 = rp->command) == NULL) {
                                line2 = get_prop(rp->target->prop, line_prop);
                        }
                        if (dmake_mode_type == distributed_mode) {
                                if (rp->make_refd) {
                                        maybe_reread_make_state();
                                }
                        } else {
                                /*
                                 * Send an Avo_MToolJobResultMsg to maketool.
                                 */
#ifdef DISTRIBUTED
                                if (send_mtool_msgs) {
                                        send_job_result_msg(rp);
                                }
#endif
                                /*
                                 * Check if there were any job output
                                 * from the parallel build.
                                 */
                                if (rp->stdout_file != NULL) {
                                        if (stat(rp->stdout_file, &out_buf) < 0)
                                                fatal(catgets(catd, 1, 130, "sta
                                                      rp->stdout_file,
                                                      errmsg(errno));
                                        }
                                        if ((line2 != NULL) &&
                                            (out_buf.st_size > 0)) {
                                                cmds_length = 0;
                                                for (rule = line2->body.line.com
                                                     silent_flag = silent;
                                                     rule != NULL;
                                                     rule = rule->next) {
                                                        cmds_length += rule->com
                                                        silent_flag = BOOLEAN(si
                                                }
                                                if (out_buf.st_size != cmds_leng
                                                    output_mode == txt2_mode) {
                                                        dump_out_file(rp->stdout
                                                }
                                        }
                                        (void) unlink(rp->stdout_file);
                                        retmem_mb(rp->stdout_file);
                                        rp->stdout_file = NULL;
                                }

                                if (!out_err_same && (rp->stderr_file != NULL))
                                        if (stat(rp->stderr_file, &out_buf) < 0)
                                                fatal(catgets(catd, 1, 130, "sta
                                                      rp->stderr_file,
                                                      errmsg(errno));
                                        }
                                        if ((line2 != NULL) &&
                                            (out_buf.st_size > 0)) {
                                                dump_out_file(rp->stderr_file, t
                                        }
                                        (void) unlink(rp->stderr_file);
                                        retmem_mb(rp->stderr_file);
                                        rp->stderr_file = NULL;
                                }
                        }
                        check_state(rp->temp_file);
                        if (rp->temp_file != NULL) {
                                free_name(rp->temp_file);
                        }
                        rp->temp_file = NULL;
                        if (rp->state == build_failed) {
                                line = get_prop(rp->target->prop, line_prop);
                                if (line != NULL) {
                                        line->body.line.command_used = NULL;
                                }
                                if (continue_after_error ||
                                    fatal_in_progress ||
                                    !docheck) {
                                        warning(catgets(catd, 1, 256, "Command f
                                                rp->command ? line2->body.line.t
                                        build_failed_seen = true;
                                } else {
                                        /*
                                         * XXX??? - DMake needs to exit(),
```

```
                                         * but shouldn’t call fatal().
                                         */
#ifdef PRINT_EXIT_STATUS
                                        warning(NOCATGETS("I’m in finish_childre
#endif

                                        fatal(catgets(catd, 1, 258, "Command fai
                                                rp->command ? line2->body.line.t
                                }
                        }
                        if (!docheck) {
                                delete_running_struct(rp);
                                rp = *rp_prev;
                                if (rp == NULL) {
                                        break;
                                } else {
                                        goto bypass_for_loop_inc_4;
                                }
                        }
                        update_target(get_prop(rp->target->prop, line_prop),
                                      rp->state);
                        finish_doname(rp);
                        delete_running_struct(rp);
                        rp = *rp_prev;
                        if (rp == NULL) {
                                break;
                        } else {
                                goto bypass_for_loop_inc_4;
                        }
                } else {
                        rp_prev = &rp->next;
                }
        }
}
______unchanged_portion_omitted_

/*
 *      add_running(target, true_target, command, recursion_level, auto_count,
 *                                      automatics, do_get, implicit)
 *
 *      Adds a record on the running list for this target, which
 *      was just spawned and is running.
 *
 *      Parameters:
 *              target          Target being built
 *              true_target     True target for target
 *              command         Running command.
 *              recursion_level Debug indentation level
 *              auto_count      Count of automatic dependencies
 *              automatics      List of automatic dependencies
 *              do_get          Sccs get flag
 *              implicit        Implicit flag
 *
 *      Static variables used:
 *              running_tail    Tail of running list
 *              process_running PID of process
 *
 *      Global variables used:
 *              current_line    Current line for target
 *              current_target  Current target being built
 *              stderr_file     Temporary file for stdout
 *              stdout_file     Temporary file for stdout
 *              temp_file_name  Temporary file for auto dependencies
 */
void
add_running(Name target, Name true_target, Property command, int recursion_level
{
        Running         rp;
        Name            *p;

        rp = new_running_struct();
        rp->state = build_running;
        rp->target = target;
        rp->true_target = true_target;
        rp->command = command;
        rp->recursion_level = recursion_level;
        rp->do_get = do_get;
        rp->implicit = implicit;
        rp->auto_count = auto_count;
        if (auto_count > 0) {
                rp->automatics = (Name *) getmem(auto_count * sizeof (Name));
                for (p = rp->automatics; auto_count > 0; auto_count--) {
                        *p++ = *automatics++;
                }
        } else {
                rp->automatics = NULL;
        }
#ifdef DISTRIBUTED
        if (dmake_mode_type == distributed_mode) {
                rp->make_refd = called_make;
                called_make = false;
        } else
#endif
        {
                rp->pid = process_running;
                process_running = -1;
                childPid = -1;
        }
        rp->job_msg_id = job_msg_id;
        rp->stdout_file = stdout_file;
        rp->stderr_file = stderr_file;
        rp->temp_file = temp_file_name;
        rp->redo = false;
        rp->next = NULL;
        store_conditionals(rp);
        stdout_file = NULL;
        stderr_file = NULL;
        temp_file_name = NULL;
        current_target = NULL;
        current_line = NULL;
        *running_tail = rp;
        running_tail = &rp->next;
}
```

______unchanged_portion_omitted_

```
#ifdef DISTRIBUTED
/*
 * Create and send an Avo_MToolJobResultMsg.
 */
static void
send_job_result_msg(Running rp)
{
        Avo_MToolJobResultMsg   *msg;
        RWCollectable           *xdr_msg;

        msg = new Avo_MToolJobResultMsg();
        msg->setResult(rp->job_msg_id,
                       (rp->state == build_ok) ? 0 : 1,
                       DONE);
        append_job_result_msg(msg,
                                rp->stdout_file,
                                rp->stderr_file);
```

```

        xdr_msg = (RWCollectable *)msg;
        xdr(get_xdrs_ptr(), xdr_msg);
        (void) fflush(get_mtool_msgs_fp());

        delete msg;
}

/*
 * Append the stdout/err to Avo_MToolJobResultMsg.
 */
static void
append_job_result_msg(Avo_MToolJobResultMsg *msg, char *outFn, char *errFn)
{
        FILE            *fp;
        char            line[MAXPATHLEN];

        fp = fopen(outFn, "r");
        if (fp == NULL) {
                /* Hmmm... what should we do here? */
                return;
        }
        while (fgets(line, MAXPATHLEN, fp) != NULL) {
                if (line[strlen(line) - 1] == '\n') {
                        line[strlen(line) - 1] = '\0';
                }
                msg->appendOutput(AVO_STRDUP(line));
        }
        (void) fclose(fp);
}
#endif

static void
delete_running_struct(Running rp)
{
        if ((rp->conditional_cnt > 0) &&
            (rp->conditional_targets != NULL)) {
                retmem_mb((char *) rp->conditional_targets);
        }
/**/
        if ((rp->auto_count > 0) &&
            (rp->automatics != NULL)) {
                retmem_mb((char *) rp->automatics);
        }
/**/
        if(rp->sprodep_value) {
                free_name(rp->sprodep_value);
        }
        if(rp->sprodep_env) {
                retmem_mb(rp->sprodep_env);
        }
        retmem_mb((char *) rp);

}
```

______unchanged_portion_omitted_

```
************************************************************
   56788 Wed May 20 11:55:57 2015
new/usr/src/cmd/make/bin/read.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
************************************************************
/*
 * 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.
 */

/*
 *      read.c
 *
 *      This file contains the makefile reader.
 */

/*
 * Included files
 */
#include <alloca.h>             /* alloca() */
#include <errno.h>              /* errno */
#include <fcntl.h>              /* fcntl() */
#include <mk/defs.h>
#include <mksh/macro.h>         /* expand_value(), expand_macro() */
#include <mksh/misc.h>          /* getmem() */
#include <mksh/read.h>          /* get_next_block_fn() */
#include <sys/uio.h>            /* read() */
#include <unistd.h>             /* read(), unlink() */


/*
 * typedefs & structs
 */

/*
 * Static variables
 */

static int line_started_with_space=0; // Used to diagnose spaces instead of tabs

/*
 * File table of contents
 */
static  void            parse_makefile(register Name true_makefile_name, registe
static  Source          push_macro_value(register Source bp, register wchar_t *b
extern  void            enter_target_groups_and_dependencies(Name_vector target,
```

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

/*
 *      read_simple_file(makefile_name, chase_path, doname_it,
 *               complain, must_exist, report_file, lock_makefile)
 *
 *      Make the makefile and setup to read it. Actually read it if it is stdio
 *
 *      Return value:
 *                              false if the read failed
 *
 *      Parameters:
 *              makefile_name   Name of the file to read
 *              chase_path      Use the makefile path when opening file
 *              doname_it       Call doname() to build the file first
 *              complain        Print message if doname/open fails
 *              must_exist      Generate fatal if file is missing
 *              report_file     Report file when running -P
 *              lock_makefile   Lock the makefile when reading
 *
 *      Static variables used:
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              file_being_read Set to the name of the new file
 *              line_number     The number of the current makefile line
 *              makefiles_used  A list of all makefiles used, appended to
 */


Boolean
read_simple_file(register Name makefile_name, register Boolean chase_path, regis
{
        static short            max_include_depth;
        register Property       makefile = maybe_append_prop(makefile_name,
                                                             makefile_prop);
        Boolean                 forget_after_parse = false;
        static pathpt           makefile_path;
        register int            n;
        char                    *path;
        register Source         source = ALLOC(Source);
        Property                orig_makefile = makefile;
        Dependency              *dpp;
        Dependency              dp;
        register int            length;
        wchar_t                 *previous_file_being_read = file_being_read;
        int                     previous_line_number = line_number;
        wchar_t                 previous_current_makefile[MAXPATHLEN];
        Makefile_type           save_makefile_type;
        Name                    normalized_makefile_name;
        register wchar_t        *string_start;
        register wchar_t        *string_end;



        wchar_t * wcb = get_wstring(makefile_name->string_mb);

        if (max_include_depth++ >= 40) {
                fatal(catgets(catd, 1, 66, "Too many nested include statements")
        }
        if (makefile->body.makefile.contents != NULL) {
                retmem(makefile->body.makefile.contents);
        }
        source->inp_buf =
          source->inp_buf_ptr =
            source->inp_buf_end = NULL;
        source->error_converting = false;
        makefile->body.makefile.contents = NULL;
        makefile->body.makefile.size = 0;
        if ((makefile_name->hash.length != 1) ||
            (wcb[0] != (int) hyphen_char)) {
                if ((makefile->body.makefile.contents == NULL) &&
                    (doname_it)) {
                        if (makefile_path == NULL) {
                                add_dir_to_path(".",
                                                &makefile_path,
                                                -1);
                                add_dir_to_path(NOCATGETS("/usr/share/lib/make")
                                                &makefile_path,
                                                -1);
                                add_dir_to_path(NOCATGETS("/etc/default"),
                                                &makefile_path,
                                                -1);
                        }
                        save_makefile_type = makefile_type;
                        makefile_type = reading_nothing;
                        if (doname(makefile_name, true, false) == build_dont_kno
                                /* Try normalized filename */
                                string_start=get_wstring(makefile_name->string_m
                                for (string_end=string_start+1; *string_end != L
                                normalized_makefile_name=normalize_name(string_s
                                if ((strcmp(makefile_name->string_mb, normalized
                                        (doname(normalized_makefile_name, true,
                                        n = access_vroot(makefile_name->string_m
                                                 4,
                                                 chase_path ?
                                                 makefile_path : NULL,
                                                 VROOT_DEFAULT);
                                        if (n == 0) {
                                                get_vroot_path((char **) NULL,
                                                       &path,
                                                       (char **) NULL);
                                                if ((path[0] == (int) period_cha
                                                    (path[1] == (int) slash_char
                                                        path += 2;
                                                }
                                                MBSTOWCS(wcs_buffer, path);
                                                makefile_name = GETNAME(wcs_buff
                                                                FIND_LENGTH);
                                        }
                                }
                                retmem(string_start);
                                /*
                                 * Commented out: retmem_mb(normalized_makefile_
                                 * We have to return this memory, but it seems t
                                 * in dmake or in Sun C++ 5.7 compiler (it works
                                 * is compiled using Sun C++ 5.6).
                                 */
                                // retmem_mb(normalized_makefile_name->string_mb
                        }
                        makefile_type = save_makefile_type;
                }
                source->string.free_after_use = false;
                source->previous = NULL;
                source->already_expanded = false;
                /* Lock the file for read, but not when -n. */
                if (lock_makefile &&
                    !do_not_exec_rule) {

                         make_state_lockfile = getmem(strlen(make_state->string_
                         (void) sprintf(make_state_lockfile,
                                                NOCATGETS("%s.lock"),
```

```
                                                make_state->string_mb);
                        (void) file_lock(make_state->string_mb,
                                         make_state_lockfile,
                                         (int *) &make_state_locked,
                                         0);
                        if(!make_state_locked) {
                                printf(NOCATGETS("-- NO LOCKING for read\n"));
                                retmem_mb(make_state_lockfile);
                                make_state_lockfile = 0;
                                return failed;
                        }
                }
                if (makefile->body.makefile.contents == NULL) {
                        save_makefile_type = makefile_type;
                        makefile_type = reading_nothing;
                        if ((doname_it) &&
                            (doname(makefile_name, true, false) == build_failed)
                                if (complain) {
                                        (void) fprintf(stderr,
#ifdef DISTRIBUTED
                                                       catgets(catd, 1, 67, "dma
#else
                                                       catgets(catd, 1, 237, "ma
#endif
                                                       makefile_name->string_mb)
                                }
                                max_include_depth--;
                                makefile_type = save_makefile_type;
                                return failed;
                        }
                        makefile_type = save_makefile_type;
                        //
                        // Before calling exists() make sure that we have the ri
                        //
                        makefile_name->stat.time = file_no_time;

                        if (exists(makefile_name) == file_doesnt_exist) {
                                if (complain ||
                                    (makefile_name->stat.stat_errno != ENOENT))
                                        if (must_exist) {
                                                fatal(catgets(catd, 1, 68, "Can'
                                                      makefile_name->string_mb,
                                                      errmsg(makefile_name->
                                                             stat.stat_errno));
                                        } else {
                                                warning(catgets(catd, 1, 69, "Ca
                                                        makefile_name->string_mb
                                                        errmsg(makefile_name->
                                                               stat.stat_errno))
                                        }
                                }
                                max_include_depth--;
                                if(make_state_locked && (make_state_lockfile !=
                                        (void) unlink(make_state_lockfile);
                                        retmem_mb(make_state_lockfile);
                                        make_state_lockfile = NULL;
                                        make_state_locked = false;
                                }
                                retmem(wcb);
                                retmem_mb((char *)source);
                                return failed;
                        }
                        /*
                         * These values are the size and bytes of
                         * the MULTI-BYTE makefile.
                         */
                        orig_makefile->body.makefile.size =
                          makefile->body.makefile.size =
                            source->bytes_left_in_file =
                              makefile_name->stat.size;
                        if (report_file) {
                                for (dpp = &makefiles_used;
                                     *dpp != NULL;
                                     dpp = &(*dpp)->next);
                                dp = ALLOC(Dependency);
                                dp->next = NULL;
                                dp->name = makefile_name;
                                dp->automatic = false;
                                dp->stale = false;
                                dp->built = false;
                                *dpp = dp;
                        }
                        source->fd = open_vroot(makefile_name->string_mb,
                                                O_RDONLY,
                                                0,
                                                NULL,
                                                VROOT_DEFAULT);
                        if (source->fd < 0) {
                                if (complain || (errno != ENOENT)) {
                                        if (must_exist) {
                                                fatal(catgets(catd, 1, 70, "Can’
                                                      makefile_name->string_mb,
                                                      errmsg(errno));
                                        } else {
                                                warning(catgets(catd, 1, 71, "Ca
                                                        makefile_name->string_mb
                                                        errmsg(errno));
                                        }
                                }
                                max_include_depth--;
                                return failed;
                        }
                        (void) fcntl(source->fd, F_SETFD, 1);
                        orig_makefile->body.makefile.contents =
                          makefile->body.makefile.contents =
                            source->string.text.p =
                              source->string.buffer.start =
                                ALLOC_WC((int) (makefile_name->stat.size + 2));
                        if (makefile_type == reading_cpp_file) {
                                forget_after_parse = true;
                        }
                        source->string.text.end = source->string.text.p;
                        source->string.buffer.end =
                          source->string.text.p + makefile_name->stat.size;
                } else {
                        /* Do we ever reach here? */
                        source->fd = -1;
                        source->string.text.p =
                          source->string.buffer.start =
                            makefile->body.makefile.contents;
                        source->string.text.end =
                          source->string.buffer.end =
                            source->string.text.p + makefile->body.makefile.size
                        source->bytes_left_in_file =
                          makefile->body.makefile.size;
                }
                file_being_read = wcb;
        } else {
                char            *stdin_text_p;
                char            *stdin_text_end;
                char            *stdin_buffer_start;
                char            *stdin_buffer_end;
```

```
                char            *p_mb;
                int             num_mb_chars;
                size_t          num_wc_chars;

                MBSTOWCS(wcs_buffer, NOCATGETS("Standard in"));
                makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
                /*
                 * Memory to read standard in, then convert it
                 * to wide char strings.
                 */
                stdin_buffer_start =
                  stdin_text_p = getmem(length = 1024);
                stdin_buffer_end = stdin_text_p + length;
                MBSTOWCS(wcs_buffer, NOCATGETS("standard input"));
                file_being_read = (wchar_t *) wsdup(wcs_buffer);
                line_number = 0;
                while ((n = read(fileno(stdin),
                                 stdin_text_p,
                                 length)) > 0) {
                        length -= n;
                        stdin_text_p += n;
                        if (length == 0) {
                                p_mb = getmem(length = 1024 +
                                              (stdin_buffer_end -
                                               stdin_buffer_start));
                                (void) strncpy(p_mb,
                                               stdin_buffer_start,
                                               (stdin_buffer_end -
                                                stdin_buffer_start));
                                retmem_mb(stdin_buffer_start);
                                stdin_text_p = p_mb +
                                  (stdin_buffer_end - stdin_buffer_start);
                                stdin_buffer_start = p_mb;
                                stdin_buffer_end =
                                  stdin_buffer_start + length;
                                length = 1024;
                        }
                }
                if (n < 0) {
                        fatal(catgets(catd, 1, 72, "Error reading standard input
                              errmsg(errno));
                }
                stdin_text_p = stdin_buffer_start;
                stdin_text_end = stdin_buffer_end - length;
                num_mb_chars = stdin_text_end - stdin_text_p;

                /*
                 * Now, convert the sequence of multibyte chars into
                 * a sequence of corresponding wide character codes.
                 */
                source->string.free_after_use = false;
                source->previous = NULL;
                source->bytes_left_in_file = 0;
                source->fd = -1;
                source->already_expanded = false;
                source->string.buffer.start =
                  source->string.text.p = ALLOC_WC(num_mb_chars + 1);
                source->string.buffer.end =
                    source->string.text.p + num_mb_chars;
                num_wc_chars = mbstowcs(source->string.text.p,
                                        stdin_text_p,
                                        num_mb_chars);
                if ((int) num_wc_chars >= 0) {
                        source->string.text.end =
                          source->string.text.p + num_wc_chars;
                }
                (void) retmem_mb(stdin_text_p);
        }
        line_number = 1;
        if (trace_reader) {
                (void) printf(catgets(catd, 1, 73, ">>>>>>>>>>>>>>>> Reading mak
                              makefile_name->string_mb);
        }
        parse_makefile(makefile_name, source);
        if (trace_reader) {
                (void) printf(catgets(catd, 1, 74, ">>>>>>>>>>>>>>>> End of make
                              makefile_name->string_mb);
        }
        if(file_being_read) {
                retmem(file_being_read);
        }
        file_being_read = previous_file_being_read;
        line_number = previous_line_number;
        makefile_type = reading_nothing;
        max_include_depth--;
        if (make_state_locked) {
                /* Unlock .make.state. */
                unlink(make_state_lockfile);
                make_state_locked = false;
                retmem_mb(make_state_lockfile);
        }
        if (forget_after_parse) {
                retmem(makefile->body.makefile.contents);
                makefile->body.makefile.contents = NULL;
        }
        retmem_mb((char *)source);
        return succeeded;
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   51505 Wed May 20 11:55:58 2015
new/usr/src/cmd/make/bin/read2.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
______unchanged_portion_omitted_

/*
 *      fatal_reader(format, args...)
 *
 *      Parameters:
 *              format          printf style format string
 *              args            arguments to match the format
 *
 *      Global variables used:
 *              file_being_read Name of the makefile being read
 *              line_number     Line that is being read
 *              report_pwd      Indicates whether current path should be shown
 *              temp_file_name  When reading tempfile we report that name
 */
/*VARARGS*/
void
fatal_reader(char * pattern, ...)
{
        va_list args;
        char    message[1000];

        va_start(args, pattern);
        if (file_being_read != NULL) {
                WCSTOMBS(mbs_buffer, file_being_read);
                if (line_number != 0) {
                        (void) sprintf(message,
                                       catgets(catd, 1, 112, "%s, line %d: %s"),
                                       mbs_buffer,
                                       line_number,
                                       pattern);
                } else {
                        (void) sprintf(message,
                                       "%s: %s",
                                       mbs_buffer,
                                       pattern);
                }
                pattern = message;
        }

        (void) fflush(stdout);
#ifdef DISTRIBUTED
        (void) fprintf(stderr, catgets(catd, 1, 113, "dmake: Fatal error in read
#else
        (void) fprintf(stderr, catgets(catd, 1, 238, "make: Fatal error in reade
#endif
        (void) vfprintf(stderr, pattern, args);
        (void) fprintf(stderr, "\n");
        va_end(args);

        if (temp_file_name != NULL) {
                (void) fprintf(stderr,
#ifdef DISTRIBUTED
                               catgets(catd, 1, 114, "dmake: Temp-file %s not re
#else
                               catgets(catd, 1, 239, "make: Temp-file %s not rem
#endif
                               temp_file_name->string_mb);
                temp_file_name = NULL;
        }
```

```
        if (report_pwd) {
                (void) fprintf(stderr,
                               catgets(catd, 1, 115, "Current working directory
                               get_current_path());
        }
        (void) fflush(stderr);
        exit_status = 1;
        exit(1);
}
______unchanged_portion_omitted_
```

****
1412 Wed May 20 11:55:59 2015
new/usr/src/cmd/make/include/avo/intl.h
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
****

```
/*
 * 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 2001 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _AVO_INTL_H
#define _AVO_INTL_H


/*
 * For catgets
 */
#include <nl_types.h>


/*
 * NOCATGETS is a dummy macro that returns it argument.
 * It is used to identify strings that we consciously do not
 * want to apply catgets() to.  We have tools that check the
 * sources for strings that are not catgets'd and the tools
 * ignore strings that are NOCATGETS'd.
 */
#define NOCATGETS(str)  (str)

/*
 * Define the various text domains
 */
#define AVO_DOMAIN_CODEMGR      "codemgr"
#define AVO_DOMAIN_VERTOOL      "vertool"
#define AVO_DOMAIN_FILEMERGE    "filemerge"
#define AVO_DOMAIN_DMAKE        "dmake"
#define AVO_DOMAIN_PMAKE        "pmake"
#define AVO_DOMAIN_FREEZEPOINT  "freezept"
#define AVO_DOMAIN_MAKETOOL     "maketool"

#endif
```

```
**********************************************************
   14246 Wed May 20 11:55:59 2015
new/usr/src/cmd/make/include/mk/defs.h
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
#ifndef _MK_DEFS_H
#define _MK_DEFS_H
/*
 * 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.
 */

/*
 * Included files
 */
#ifdef DISTRIBUTED
#       include <dm/Avo_AcknowledgeMsg.h>
#       include <dm/Avo_DoJobMsg.h>
#       include <dm/Avo_JobResultMsg.h>
#endif

#include <mksh/defs.h>

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <rw/xdrstrea.h>
#endif


/*
 * Defined macros
 */

#define SKIPSPACE(x)    while (*x &&                                    \
                               ((*x == (int) space_char) ||             \
                                (*x == (int) tab_char) ||               \
                                (*x == (int) comma_char))) {            \
                                        x++;                            \
                        }

#define SKIPWORD(x)     while (*x &&                                    \
                               (*x != (int) space_char) &&              \
                               (*x != (int) tab_char) &&                \
                               (*x != (int) newline_char) &&            \
                               (*x != (int) comma_char) &&              \
                               (*x != (int) equal_char)) {              \
                                        x++;                            \
```

```
                        }

#define SKIPTOEND(x)    while (*x &&                                    \
                               (*x != (int) newline_char)) {            \
                                        x++;                            \
                        }

#define PMAKE_DEF_MAX_JOBS      2       /* Default number of parallel jobs. */

#define OUT_OF_DATE(a,b) \
        (((a) < (b)) || (((a) == file_doesnt_exist) && ((b) == file_doesnt_exist

#define OUT_OF_DATE_SEC(a,b) \
        (((a).tv_sec < (b).tv_sec) || (((a).tv_sec == file_doesnt_exist.tv_sec)

#define SETVAR(name, value, append) \
                                setvar_daemon(name, value, append, no_daemon, \
                                              true, debug_level)
#define MAX(a,b)                (((a)>(b))?(a):(b))
/*
 * New feature added to SUN5_0 make,  invoke the vanilla svr4 make when
 * the USE_SVR4_MAKE environment variable is set.
 */
#define SVR4_MAKE               "/usr/ccs/lib/svr4.make"
#define USE_SVR4_MAKE           "USE_SVR4_MAKE"
/*
 * The standard MAXHOSTNAMELEN is 64. We want 32.
 */
#define MAX_HOSTNAMELEN         32


/*
 * typedefs & structs
 */
typedef enum {
        no_state,
        scan_name_state,
        scan_command_state,
        enter_dependencies_state,
        enter_conditional_state,
        enter_equal_state,
        illegal_bytes_state,
        illegal_eoln_state,
        poorly_formed_macro_state,
        exit_state
}                       Reader_state;
______unchanged_portion_omitted_


/*
 * Typedefs for all structs
 */
typedef struct _Cmd_line        *Cmd_line, Cmd_line_rec;
typedef struct _Dependency      *Dependency, Dependency_rec;
typedef struct _Macro           *Macro, Macro_rec;
typedef struct _Name_vector     *Name_vector, Name_vector_rec;
typedef struct _Percent         *Percent, Percent_rec;
typedef struct _Dyntarget       *Dyntarget;
typedef struct _Recursive_make  *Recursive_make, Recursive_make_rec;
typedef struct _Running         *Running, Running_rec;


/*
 *      extern declarations for all global variables.
 *      The actual declarations are in globals.cc
 */
```

```
extern  Boolean         allrules_read;
extern  Name            posix_name;
extern  Name            svr4_name;
extern  Boolean         sdot_target;
extern  Boolean         all_parallel;
extern  Boolean         assign_done;
extern  Boolean         build_failed_seen;
#ifdef DISTRIBUTED
extern  Boolean         building_serial;
#endif
extern  Name            built_last_make_run;
extern  Name            c_at;
#ifdef DISTRIBUTED
extern  Boolean         called_make;
#endif
extern  Boolean         command_changed;
extern  Boolean         commands_done;
extern  Chain           conditional_targets;
extern  Name            conditionals;
extern  Boolean         continue_after_error;
extern  Property        current_line;
extern  Name            current_make_version;
extern  Name            current_target;
extern  short           debug_level;
extern  Cmd_line        default_rule;
extern  Name            default_rule_name;
extern  Name            default_target_to_build;
extern  Boolean         depinfo_already_read;
extern  Name            dmake_group;
extern  Name            dmake_max_jobs;
extern  Name            dmake_mode;
extern  DMake_mode      dmake_mode_type;
extern  Name            dmake_output_mode;
extern  DMake_output_mode       output_mode;
extern  Name            dmake_odir;
extern  Name            dmake_rcfile;
extern  Name            done;
extern  Name            dot;
extern  Name            dot_keep_state;
extern  Name            dot_keep_state_file;
extern  Name            empty_name;
extern  Boolean         fatal_in_progress;
extern  int             file_number;
extern  Name            force;
extern  Name            ignore_name;
extern  Boolean         ignore_errors;
extern  Boolean         ignore_errors_all;
extern  Name            init;
extern  int             job_msg_id;
extern  Boolean         keep_state;
extern  Name            make_state;
#ifdef TEAMWARE_MAKE_CMN
extern  timestruc_t     make_state_before;
#endif
extern  Boolean         make_state_locked;
extern  Dependency      makefiles_used;
extern  Name            makeflags;
extern  Name            make_version;
extern  char            mbs_buffer2[];
extern  char            *mbs_ptr;
extern  char            *mbs_ptr2;
extern  Boolean         no_action_was_taken;
extern  int             mtool_msgs_fd;
extern  Boolean         no_parallel;
extern  Name            no_parallel_name;
extern  Name            not_auto;
```

```
extern  Boolean         only_parallel;
extern  Boolean         parallel;
extern  Name            parallel_name;
extern  Name            localhost_name;
extern  int             parallel_process_cnt;
extern  Percent         percent_list;
extern  Dyntarget       dyntarget_list;
extern  Name            plus;
extern  Name            pmake_machinesfile;
extern  Name            precious;
extern  Name            primary_makefile;
extern  Boolean         quest;
extern  short           read_trace_level;
extern  Boolean         reading_dependencies;
extern  int             recursion_level;
extern  Name            recursive_name;
extern  short           report_dependencies_level;
extern  Boolean         report_pwd;
extern  Boolean         rewrite_statefile;
extern  Running         running_list;
extern  char            *sccs_dir_path;
extern  Name            sccs_get_name;
extern  Name            sccs_get_posix_name;
extern  Cmd_line        sccs_get_rule;
extern  Cmd_line        sccs_get_org_rule;
extern  Cmd_line        sccs_get_posix_rule;
extern  Name            get_name;
extern  Name            get_posix_name;
extern  Cmd_line        get_rule;
extern  Cmd_line        get_posix_rule;
extern  Boolean         send_mtool_msgs;
extern  Boolean         all_precious;
extern  Boolean         report_cwd;
extern  Boolean         silent_all;
extern  Boolean         silent;
extern  Name            silent_name;
extern  char            *stderr_file;
extern  char            *stdout_file;
extern  Boolean         stdout_stderr_same;
extern  Dependency      suffixes;
extern  Name            suffixes_name;
extern  Name            sunpro_dependencies;
extern  Boolean         target_variants;
extern  const char      *tmpdir;
extern  const char      *temp_file_directory;
extern  Name            temp_file_name;
extern  short           temp_file_number;
extern  wchar_t         *top_level_target;
extern  Boolean         touch;
extern  Boolean         trace_reader;
extern  Boolean         build_unconditional;
extern  pathpt          vroot_path;
extern  Name            wait_name;
extern  wchar_t         wcs_buffer2[];
extern  wchar_t         *wcs_ptr;
extern  wchar_t         *wcs_ptr2;
extern  nl_catd         catd;
extern  long int        hostid;

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

/*
```

```
 * Declarations of system supplied functions
 */
extern  int             file_lock(char *, char *, int *, int);

/*
 * Declarations of functions declared and used by make
 */
extern  void            add_pending(Name target, int recursion_level, Boolean do
extern  void            add_running(Name target, Name true_target, Property comm
extern  void            add_serial(Name target, int recursion_level, Boolean do_
extern  void            add_subtree(Name target, int recursion_level, Boolean do
extern  void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
#ifdef DISTRIBUTED
extern  Doname          await_dist(Boolean waitflg);
#endif
#ifdef TEAMWARE_MAKE_CMN
extern  void            await_parallel(Boolean waitflg);
#endif
extern  void            build_suffix_list(Name target_suffix);
extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
extern  void            check_state(Name temp_file_name);
extern  void            cond_macros_into_string(Name np, String_rec *buffer);
extern  void            construct_target_string();
extern  void            create_xdrs_ptr(void);
extern  void            depvar_add_to_list (Name name, Boolean cmdline);
#ifdef DISTRIBUTED
extern  void            distribute_rxm(Avo_DoJobMsg *dmake_job_msg);
extern  int             getRxmMessage(void);
extern  Avo_JobResultMsg* getJobResultMsg(void);
extern  Avo_AcknowledgeMsg* getAcknowledgeMsg(void);
#endif
extern  Doname          doname(register Name target, register Boolean do_get, re
extern  Doname          doname_check(register Name target, register Boolean do_g
extern  Doname          doname_parallel(Name target, Boolean do_get, Boolean imp
extern  Doname          dosys(register Name command, register Boolean ignore_err
extern  void            dump_make_state(void);
extern  void            dump_target_list(void);
extern  void            enter_conditional(register Name target, Name name, Name 
extern  void            enter_dependencies(register Name target, Chain target_gr
extern  void            enter_dependency(Property line, register Name depe, Bool
extern  void            enter_equal(Name name, Name value, register Boolean appe
extern  Percent         enter_percent(register Name target, Chain target_group, 
extern  Dyntarget       enter_dyntarget(register Name target);
extern  Name_vector     enter_name(String string, Boolean tail_present, register
extern  Boolean         exec_vp(register char *name, register char **argv, char 
extern  Doname          execute_parallel(Property line, Boolean waitflg, Boolean
extern  Doname          execute_serial(Property line);
extern  timestruc_t&    exists(register Name target);
extern  void            fatal(const char *, ...);
extern  void            fatal_reader(char *, ...);
extern  Doname          find_ar_suffix_rule(register Name target, Name true_targ
extern  Doname          find_double_suffix_rule(register Name target, Property *
extern  Doname          find_percent_rule(register Name target, Property *comman
extern  int             find_run_directory (char *cmd, char *cwd, char *dir, cha
extern  Doname          find_suffix_rule(Name target, Name target_body, Name tar
extern  Chain           find_target_groups(register Name_vector target_list, reg
extern  void            finish_children(Boolean docheck);
extern  void            finish_running(void);
extern  void            free_chain(Name_vector ptr);
extern  void            gather_recursive_deps(void);
extern  char            *get_current_path(void);
extern  int             get_job_msg_id(void);
extern  FILE            *get_mtool_msgs_fp(void);
#ifdef DISTRIBUTED
extern  Boolean         get_dmake_group_specified(void);
extern  Boolean         get_dmake_max_jobs_specified(void);
```

```
extern  Boolean         get_dmake_mode_specified(void);
extern  Boolean         get_dmake_odir_specified(void);
extern  Boolean         get_dmake_rcfile_specified(void);
extern  Boolean         get_pmake_machinesfile_specified(void);
#endif
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
extern  XDR             *get_xdrs_ptr(void);
#endif
extern  wchar_t         *getmem_wc(register int size);
/* On linux getwd(char *) is defined in 'unistd.h' */
#ifdef __cplusplus
extern "C" {
#endif
extern  char            *getwd(char *);
#ifdef __cplusplus
}
#endif
extern  void            handle_interrupt(int);
extern  Boolean         is_running(Name target);
extern  void            load_cached_names(void);
extern  Boolean         parallel_ok(Name target, Boolean line_prop_must_exists);
extern  void            print_dependencies(register Name target, register Proper
extern  void            send_job_start_msg(Property line);
extern  void            send_rsrc_info_msg(int max_jobs, char *hostname, char *u
extern  void            print_value(register Name value, Daemon daemon);
extern  timestruc_t&    read_archive(register Name target);
extern  int             read_dir(Name dir, wchar_t *pattern, Property line, wcha
extern  void            read_directory_of_file(register Name file);
extern  int             read_make_machines(Name make_machines_name);
extern  Boolean         read_simple_file(register Name makefile_name, register B
extern  void            remove_recursive_dep(Name target);
extern  void            report_recursive_dep(Name target, char *line);
extern  void            report_recursive_done(void);
extern  void            report_recursive_init(void);
extern  Recursive_make  find_recursive_target(Name target);
extern  void            reset_locals(register Name target, register Property old
extern  void            set_locals(register Name target, register Property old_l
extern  void            setvar_append(register Name name, register Name value);
#ifdef DISTRIBUTED
extern  void            setvar_envvar(Avo_DoJobMsg *dmake_job_msg);
#else
extern  void            setvar_envvar(void);
#endif
extern  void            special_reader(Name target, register Name_vector depes, 
extern  void            startup_rxm();
extern  Doname          target_can_be_built(register Name target);
extern  char            *time_to_string(const timestruc_t &time);
extern  void            update_target(Property line, Doname result);
extern  void            warning(char *, ...);
extern  void            write_state_file(int report_recursive, Boolean exiting);
extern  Name            vpath_translation(register Name cmd);

#define DEPINFO_FMT_VERSION "VERS2$"
#define VER_LEN strlen(DEPINFO_FMT_VERSION)


#endif
```

****
1947 Wed May 20 11:55:59 2015
new/usr/src/cmd/make/include/mksh/dosys.h
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
****

```
#ifndef _MKSH_DOSYS_H
#define _MKSH_DOSYS_H
/*
 * 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.
 */

#include <mksh/defs.h>
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <rw/xdrstrea.h>
#endif
#include <vroot/vroot.h>

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
extern Boolean  await(register Boolean ignore_error, register Boolean silent_err
#else
extern Boolean  await(register Boolean ignore_error, register Boolean silent_err
#endif
extern int      doexec(register wchar_t *command, register Boolean ignore_error,
extern int      doshell(wchar_t *command, register Boolean ignore_error, Boolean
extern Doname   dosys_mksh(register Name command, register Boolean ignore_error,
extern void     redirect_io(char *stdout_file, char *stderr_file);
extern void     sh_command2string(register String command, register String desti

#endif
```

****
    1047 Wed May 20 11:56:00 2015
new/usr/src/cmd/make/include/mksh/mksh.h
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
****

```
#ifndef _MKSH_MKSH_H
#define _MKSH_MKSH_H
/*
 * 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 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


/*
 * Included files
 */
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
#       include <dm/Avo_DmakeCommand.h>
#endif

#include <mksh/defs.h>
#include <unistd.h>

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */

extern int      do_job(Avo_DmakeCommand *cmd_list[], char *env_list[], char *std

#endif /* TEAMWARE_MAKE_CMN */

#endif
```

```
**********************************************************
   20098 Wed May 20 11:56:00 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
/*
 * 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 */
#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
await(register Boolean ignore_error, register Boolean silent_error, Name target,
#else
Boolean
await(register Boolean ignore_error, register Boolean silent_error, Name target,
#endif
{
        int                     status;
        char                    *buffer;
        int                     core_dumped;
        int                     exit_status;
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
        Avo_CmdOutput           *make_output_msg;
#endif
        FILE                    *outfp;
        register pid_t          pid;
        struct stat             stat_buff;
        int                     termination_signal;
        char                    tmp_buf[MAXPATHLEN];
#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
        RWCollectable           *xdr_msg;
#endif

        while ((pid = wait(&status)) != running_pid) {
                if (pid == -1) {
                        fatal_mksh(catgets(libmksdmsi18n_catd, 1, 98, "wait() fa
                }
        }
        (void) fflush(stdout);
        (void) fflush(stderr);

        if (status == 0) {

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

                return succeeded;
        }

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


        exit_status = WEXITSTATUS(status);

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

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

        /*
         * If the child returned an error, we now try to print a
         * nice message about it.
         */
        SEND_MTOOL_MSG(
                make_output_msg = new Avo_CmdOutput();
                (void) sprintf(tmp_buf, "%d", job_msg_id);
                make_output_msg->appendOutput(strdup(tmp_buf));
        );

        tmp_buf[0] = (int) nul_char;
        if (!silent_error) {
                if (exit_status != 0) {
                        (void) fprintf(stdout,
```

```
                                       catgets(libmksdmsi18n_catd, 1, 103, "***
                                       exit_status);
                        SEND_MTOOL_MSG(
                                (void) sprintf(&tmp_buf[strlen(tmp_buf)],
                                               catgets(libmksdmsi18n_catd, 1, 10
                                               exit_status);
                        );
                } else {
                                (void) fprintf(stdout,
                                               catgets(libmksdmsi18n_catd, 1, 10
                                               termination_signal);
                                SEND_MTOOL_MSG(
                                        (void) sprintf(&tmp_buf[strlen(tmp_buf)]
                                                       catgets(libmksdmsi18n_cat
                                                       termination_signal);
                                );
                        if (core_dumped) {
                                (void) fprintf(stdout,
                                               catgets(libmksdmsi18n_catd, 1, 10
                                SEND_MTOOL_MSG(
                                        (void) sprintf(&tmp_buf[strlen(tmp_buf)]
                                                       catgets(libmksdmsi18n_cat
                                );
                        }
                }
                if (ignore_error) {
                        (void) fprintf(stdout,
                                       catgets(libmksdmsi18n_catd, 1, 109, " (ig
                        SEND_MTOOL_MSG(
                                (void) sprintf(&tmp_buf[strlen(tmp_buf)],
                                               catgets(libmksdmsi18n_catd, 1, 11
                        );
                }
                (void) fprintf(stdout, "\n");
                (void) fflush(stdout);
                SEND_MTOOL_MSG(
                        make_output_msg->appendOutput(strdup(tmp_buf));
                );
        }
        SEND_MTOOL_MSG(
                xdr_msg = (RWCollectable*) make_output_msg;
                xdr(xdrs_p, xdr_msg);
                delete make_output_msg;
        );

#ifdef PRINT_EXIT_STATUS
        warning_mksh(NOCATGETS("I’m in await(), returning failed."));
#endif

        return failed;
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
   37120 Wed May 20 11:56:01 2015
new/usr/src/cmd/make/lib/mksh/macro.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
______unchanged_portion_omitted_

/*
 * We use a permanent buffer to reset SUNPRO_DEPENDENCIES value.
 */
char    *sunpro_dependencies_buf = NULL;
char    *sunpro_dependencies_oldbuf = NULL;
int     sunpro_dependencies_buf_size = 0;

/*
 *      setvar_daemon(name, value, append, daemon, strip_trailing_spaces)
 *
 *      Set a macro value, possibly supplying a daemon to be used
 *      when referencing the value.
 *
 *      Return value:
 *                              The property block with the new value
 *
 *      Parameters:
 *              name            Name of the macro to set
 *              value           The value to set
 *              append          Should we reset or append to the current value?
 *              daemon          Special treatment when reading the value
 *              strip_trailing_spaces from the end of value->string
 *              debug_level     Indicates how much tracing we should do
 *
 *      Global variables used:
 *              makefile_type   Used to check if we should enforce read only
 *              path_name       The Name "PATH", compared against
 *              virtual_root    The Name "VIRTUAL_ROOT", compared against
 *              vpath_defined   Set if the macro VPATH is set
 *              vpath_name      The Name "VPATH", compared against
 *              envvar          A list of environment vars with $ in value
 */
Property
setvar_daemon(register Name name, register Name value, Boolean append, Daemon da
{
        register Property       macro = maybe_append_prop(name, macro_prop);
        register Property       macro_apx = get_prop(name->prop, macro_append_pr
        int                     length = 0;
        String_rec              destination;
        wchar_t                 buffer[STRING_BUFFER_LENGTH];
        register Chain          chain;
        Name                    val;
        wchar_t                 *val_string = (wchar_t*)NULL;
        Wstring                 wcb;


        if ((makefile_type != reading_nothing) &&
            macro->body.macro.read_only) {
                return macro;
        }
        /* Strip spaces from the end of the value */
        if (daemon == no_daemon) {
                if(value != NULL) {
                        wcb.init(value);
                        length = wcb.length();
                        val_string = wcb.get_string();
                }
                if ((length > 0) && iswspace(val_string[length-1])) {
                        INIT_STRING_FROM_STACK(destination, buffer);
```

```
                        buffer[0] = 0;
                        append_string(val_string, &destination, length);
                        if (strip_trailing_spaces) {
                                while ((length > 0) &&
                                       iswspace(destination.buffer.start[length-
                                        destination.buffer.start[--length] = 0;
                                }
                        }
                        value = GETNAME(destination.buffer.start, FIND_LENGTH);
                }
        }

        if(macro_apx != NULL) {
                val = macro_apx->body.macro_appendix.value;
        } else {
                val = macro->body.macro.value;
        }

        if (append) {
                /*
                 * If we are appending, we just tack the new value after
                 * the old one with a space in between.
                 */
                INIT_STRING_FROM_STACK(destination, buffer);
                buffer[0] = 0;
                if ((macro != NULL) && (val != NULL)) {
                        APPEND_NAME(val,
                                      &destination,
                                      (int) val->hash.length);
                        if (value != NULL) {
                                wcb.init(value);
                                if(wcb.length() > 0) {
                                        MBTOWC(wcs_buffer, " ");
                                        append_char(wcs_buffer[0], &destination)
                                }
                        }
                }
                if (value != NULL) {
                        APPEND_NAME(value,
                                      &destination,
                                      (int) value->hash.length);
                }
                value = GETNAME(destination.buffer.start, FIND_LENGTH);
                wcb.init(value);
                if (destination.free_after_use) {
                        retmem(destination.buffer.start);
                }
        }

        /* Debugging trace */
        if (debug_level > 1) {
                if (value != NULL) {
                        switch (daemon) {
                        case chain_daemon:
                                (void) printf("%s =", name->string_mb);
                                for (chain = (Chain) value;
                                     chain != NULL;
                                     chain = chain->next) {
                                        (void) printf(" %s", chain->name->string
                                }
                                (void) printf("\n");
                                break;
                        case no_daemon:
                                (void) printf("%s= %s\n",
                                              name->string_mb,
                                              value->string_mb);
                                break;
                        }
                } else {
                        (void) printf("%s =\n", name->string_mb);
                }
        }
        /* Set the new values in the macro property block */
/**/
        if(macro_apx != NULL) {
                macro_apx->body.macro_appendix.value = value;
                INIT_STRING_FROM_STACK(destination, buffer);
                buffer[0] = 0;
                if (value != NULL) {
                        APPEND_NAME(value,
                                      &destination,
                                      (int) value->hash.length);
                        if (macro_apx->body.macro_appendix.value_to_append != NU
                                MBTOWC(wcs_buffer, " ");
                                append_char(wcs_buffer[0], &destination);
                        }
                }
                if (macro_apx->body.macro_appendix.value_to_append != NULL) {
                        APPEND_NAME(macro_apx->body.macro_appendix.value_to_appe
                                      &destination,
                                      (int) macro_apx->body.macro_appendix.value
                }
                value = GETNAME(destination.buffer.start, FIND_LENGTH);
                if (destination.free_after_use) {
                        retmem(destination.buffer.start);
                }
        }
/**/
        macro->body.macro.value = value;
        macro->body.macro.daemon = daemon;
        /*
         * If the user changes the VIRTUAL_ROOT, we need to flush
         * the vroot package cache.
         */
        if (name == path_name) {
                flush_path_cache();
        }
        if (name == virtual_root) {
                flush_vroot_cache();
        }
        /* If this sets the VPATH we remember that */
        if ((name == vpath_name) &&
            (value != NULL) &&
            (value->hash.length > 0)) {
                vpath_defined = true;
        }
        /*
         * For environment variables we also set the
         * environment value each time.
         */
        if (macro->body.macro.exported) {
                static char     *env;

#ifdef DISTRIBUTED
                if (!reading_environment && (value != NULL)) {
#else
                if (!reading_environment && (value != NULL) && value->dollar) {
#endif
                        Envvar  p;

                        for (p = envvar; p != NULL; p = p->next) {
                                if (p->name == name) {
```

```
                                        p->value = value;
                                        p->already_put = false;
                                        goto found_it;
                                }
                        }
                        p = ALLOC(Envvar);
                        p->name = name;
                        p->value = value;
                        p->next = envvar;
                        p->env_string = NULL;
                        p->already_put = false;
                        envvar = p;
found_it:;
#ifdef DISTRIBUTED
                }
                if (reading_environment || (value == NULL) || !value->dollar) {
#else
                } else {
#endif
                        length = 2 + strlen(name->string_mb);
                        if (value != NULL) {
                                length += strlen(value->string_mb);
                        }
                        Property env_prop = maybe_append_prop(name, env_mem_prop
                        /*
                         * We use a permanent buffer to reset SUNPRO_DEPENDENCIE
                         */
                        if (!strncmp(name->string_mb, NOCATGETS("SUNPRO_DEPENDEN
                                if (length >= sunpro_dependencies_buf_size) {
                                        sunpro_dependencies_buf_size=length*2;
                                        if (sunpro_dependencies_buf_size < 4096)
                                                sunpro_dependencies_buf_size = 4
                                        if (sunpro_dependencies_buf)
                                                sunpro_dependencies_oldbuf = sun
                                        sunpro_dependencies_buf=getmem(sunpro_de
                                }
                                env = sunpro_dependencies_buf;
                        } else {
                                env = getmem(length);
                        }
                        env_alloc_num++;
                        env_alloc_bytes += length;
                        (void) sprintf(env,
                                       "%s=%s",
                                       name->string_mb,
                                       value == NULL ?
                                         "" : value->string_mb);
                        (void) putenv(env);
                        env_prop->body.env_mem.value = env;
                        if (sunpro_dependencies_oldbuf) {
                                /* Return old buffer */
                                retmem_mb(sunpro_dependencies_oldbuf);
                                sunpro_dependencies_oldbuf = NULL;
                        }
                }
        }
        if (name == target_arch) {
                Name            ha = getvar(host_arch);
                Name            ta = getvar(target_arch);
                Name            vr = getvar(virtual_root);
                int             length;
                wchar_t         *new_value;
                wchar_t         *old_vr;
                Boolean         new_value_allocated = false;

                Wstring         ha_str(ha);
                Wstring         ta_str(ta);
                Wstring         vr_str(vr);

                wchar_t * wcb_ha = ha_str.get_string();
                wchar_t * wcb_ta = ta_str.get_string();
                wchar_t * wcb_vr = vr_str.get_string();

                length = 32 +
                  wslen(wcb_ha) +
                    wslen(wcb_ta) +
                      wslen(wcb_vr);
                old_vr = wcb_vr;
                MBSTOWCS(wcs_buffer, NOCATGETS("/usr/arch/"));
                if (IS_WEQUALN(old_vr,
                               wcs_buffer,
                               wslen(wcs_buffer))) {
                        old_vr = (wchar_t *) wschr(old_vr, (int) colon_char) + 1
                }
                if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
                        new_value = old_vr;
                } else {
                        new_value = ALLOC_WC(length);
                        new_value_allocated = true;
                        WCSTOMBS(mbs_buffer, old_vr);
                        (void) wsprintf(new_value,
                                        NOCATGETS("/usr/arch/%s/%s:%s"),
                                        ha->string_mb + 1,
                                        ta->string_mb + 1,
                                        mbs_buffer);
                }
                if (new_value[0] != 0) {
                        (void) setvar_daemon(virtual_root,
                                             GETNAME(new_value, FIND_LENGTH),
                                             false,
                                             no_daemon,
                                             true,
                                             debug_level);
                }
                if (new_value_allocated) {
                        retmem(new_value);
                }
        }
        return macro;
}
```
______***unchanged_portion_omitted_***

```
**********************************************************
    3326 Wed May 20 11:56:01 2015
new/usr/src/cmd/make/lib/mksh/mksh.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
**********************************************************
______unchanged_portion_omitted_

/*
 * File table of contents
 */
static void     change_sunpro_dependencies_value(char *oldpath, char *newpath);
static void     init_mksh_globals(char *shell);
static void     set_env_vars(char *env_list[]);

#if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
/*
 *      Execute the command(s) of one Make or DMake rule
 */
int
do_job(Avo_DmakeCommand *cmd_list[], char *env_list[], char *stdout_file, char *
{
        Boolean                 always_exec_flag;
        char                    *cmd;
        Avo_DmakeCommand        **cmd_list_p;
        Name                    command;
        Boolean                 do_not_exec_flag;
        Boolean                 ignore_flag;
        int                     length;
        Boolean                 make_refd_flag;
        Boolean                 meta_flag;
        char                    pathname[MAXPATHLEN];
        Doname                  result;
        Boolean                 silent_flag;
        wchar_t                 *tmp_wcs_buffer;

        if ((childPid = fork()) < 0) {  /* error */
                ;
        } else if (childPid > 0) {              /* parent */
                ;
        } else {                        /* child, mksh */
                (void) sigset(SIGCHLD, SIG_DFL);
                enable_interrupt(handle_interrupt_mksh);
                /* set environment variables */
                set_env_vars(env_list);
                /* redirect stdout and stderr to temp files */
                dup2(1, 2);     // Because fatal_mksh() prints error messages in
                                // stderr but dmake uses stderr for XDR communic
                                // and stdout for errors messages.
                redirect_io(stdout_file, stderr_file);
                /* try cd'ing to cwd */
                if (my_chdir(cwd) != 0) {
                        /* try the netpath machine:pathname */
                        if (!avo_netpath_to_path(cnwd, pathname)) {
                                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 137, "
                        } else if (my_chdir(pathname) != 0) {
                                fatal_mksh(catgets(libmksdmsi18n_catd, 1, 138, "
                        }
                        /*
                         * change the value of SUNPRO_DEPENDENCIES
                         * to the new path.
                         */
                        change_sunpro_dependencies_value(cwd, pathname);
                }
                init_mksh_globals(shell);
                for (cmd_list_p = cmd_list;
```

```
                     *cmd_list_p != (Avo_DmakeCommand *) NULL;
                     cmd_list_p++) {
                        if ((*cmd_list_p)->ignore()) {
                                ignore_flag = true;
                        } else {
                                ignore_flag = false;
                        }
                        if ((*cmd_list_p)->silent()) {
                                silent_flag = true;
                        } else {
                                silent_flag = false;
                        }
/*
                        if ((*cmd_list_p)->always_exec()) {
                                always_exec_flag = true;
                        } else {
                                always_exec_flag = false;
                        }
 */
                        always_exec_flag = false;
                        if ((*cmd_list_p)->meta()) {
                                meta_flag = true;
                        } else {
                                meta_flag = false;
                        }
                        if ((*cmd_list_p)->make_refd()) {
                                make_refd_flag = true;
                        } else {
                                make_refd_flag = false;
                        }
                        if ((*cmd_list_p)->do_not_exec()) {
                                do_not_exec_flag = true;
                        } else {
                                do_not_exec_flag = false;
                        }
                        do_not_exec_rule = do_not_exec_flag;
                        cmd = (*cmd_list_p)->getCmd();
                        if ((length = strlen(cmd)) >= MAXPATHLEN) {
                                tmp_wcs_buffer = ALLOC_WC(length + 1);
                                (void) mbstowcs(tmp_wcs_buffer, cmd, length + 1)
                                command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
                                retmem(tmp_wcs_buffer);
                        } else {
                                MBSTOWCS(wcs_buffer, cmd);
                                command = GETNAME(wcs_buffer, FIND_LENGTH);
                        }
                        if ((command->hash.length > 0) &&
                            (!silent_flag || do_not_exec_flag)) {
                                (void) printf("%s\n", command->string_mb);
                        }
                        result = dosys_mksh(command,
                                            ignore_flag,
                                            make_refd_flag,
                                            false, /* bugs #4085164 & #4990057 *
                                            /* BOOLEAN(silent_flag && ignore_fla
                                            always_exec_flag,
                                            (Name) NULL,
                                            false,
                                            NULL,
                                            NULL,
                                            vroot_path,
                                            nice_prio);
                        if (result == build_failed) {

#ifdef PRINT_EXIT_STATUS
                                warning_mksh(NOCATGETS("I'm in do_job(), and dos
#endif

                                if (silent_flag) {
                                        (void) printf(catgets(libmksdmsi18n_catd
                                                      command->string_mb);
                                }
                                if (!ignore_flag && !ignore) {

#ifdef PRINT_EXIT_STATUS
                                        warning_mksh(NOCATGETS("I’m in do_job(),
#endif

                                        exit(1);
                                }
                        }
                }

#ifdef PRINT_EXIT_STATUS
                warning_mksh(NOCATGETS("I’m in do_job(), exiting 0."));
#endif

                exit(0);
        }
        return childPid;
}
#endif /* TEAMWARE_MAKE_CMN */

static void
set_env_vars(char *env_list[])
{
        char                    **env_list_p;

        for (env_list_p = env_list;
             *env_list_p != (char *) NULL;
             env_list_p++) {
                putenv(*env_list_p);
        }
}
```
______***unchanged_portion_omitted***_