```
**********************************************************
  104107 Wed May 20 11:42:01 2015
new/usr/src/cmd/make/bin/doname.cc
make: unifdef for PARALLEL (undefined, this relates to the old pmake)
**********************************************************
```

```
/*
 * 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 *      Figure out which targets are out of date and rebuild them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 <avo/avo_alloca.h>             /* alloca() */
#if defined(TEAMWARE_MAKE_CMN)
#include <avo/util.h>           /* avo_get_user(), avo_hostname() */
#endif

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

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

#ifdef PARALLEL
#       include <rx/api.h>
#endif
```



```
#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
 */
#ifndef PARALLEL
#       define LOCALHOST "localhost"
#endif

#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
```

```
123 extern  void            update_target(Property line, Doname result);
124 static  Doname          sccs_get(register Name target, register Property *comman
125 extern  void            read_directory_of_file(register Name file);
126 static  void            add_pattern_conditionals(register Name target);
127 extern  void            set_locals(register Name target, register Property old_l
128 extern  void            reset_locals(register Name target, register Property old
129 extern  Boolean         check_auto_dependencies(Name target, int auto_count, Nam
130 static  void            delete_query_chain(Chain ch);

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


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

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



```
189                  * If we can’t figure out how to build an automatic
190                  * (hidden) dependency, we just ignore it.
191                  * We later declare the target to be out of date just in
192                  * case something changed.
193                  * Also, don’t complain if just reporting the dependencies
194                  * and not building anything.
195                  */
196                 if (automatic || (report_dependencies_level > 0)) {
197                         second_pass = 0;
198                         return build_dont_know;
199                 }
200                 if(first_time) {
201                         first_time = 0;
202                         second_pass = 1;
203                         goto try_again;
204                 }
205                 second_pass = 0;
206                 if (continue_after_error && !svr4) {
207                         warning(catgets(catd, 1, 14, "Don’t know how to make tar
208                                 target->string_mb);
209                         build_failed_seen = true;
210                         return build_failed;
211                 }
212                 fatal(catgets(catd, 1, 15, "Don’t know how to make target ‘%s’")
213                 break;
214         }
215 #ifdef lint
216         return build_failed;
217 #endif
218 }
_____unchanged_portion_omitted_

293 /*
294  * DONE.
295  *
296  *      doname(target, do_get, implicit)
297  *
298  *      Chases all files the target depends on and builds any that
299  *      are out of date. If the target is out of date it is then rebuilt.
300  *
301  *      Return value:
302  *                              Indiates if build failed or nt
303  *
304  *      Parameters:
305  *              target          Target to build
306  *              do_get          Run sccs get is nessecary
307  *              implicit        doname is trying to find an implicit rule
308  *
309  *      Global variables used:
310  *              assign_done     True if command line assgnment has happened
311  *              commands_done   Preserved for the case that we need local value
312  *              debug_level     Should we trace make’s actions?
313  *              default_rule    The rule for ".DEFAULT", used as last resort
314  *              empty_name      The Name "", used when looking for single sfx
315  *              keep_state      Indicates that .KEEP_STATE is on
316  *              parallel        True if building in parallel
317  *              recursion_level Used for tracing
318  *              report_dependencies make -P is on
319  */
320 Doname
321 doname(register Name target, register Boolean do_get, register Boolean implicit,
322 {
323         Doname                  result = build_dont_know;
324         Chain                   out_of_date_list = NULL;
325 #ifdef TEAMWARE_MAKE_CMN
326         Chain                   target_group;
```

```
#endif
        Property                old_locals = NULL;
        register Property       line;
        Property                command = NULL;
        register Dependency     dependency;
        Name                    less = NULL;
        Name                    true_target = target;
        Name                    *automatics = NULL;
        register int            auto_count;
        Boolean                 rechecking_target = false;
        Boolean                 saved_commands_done;
        Boolean                 restart = false;
        Boolean                 save_parallel = parallel;
        Boolean                 doing_subtree = false;

        Boolean                 recheck_conditionals = false;

        if (target->state == build_running) {
                return build_running;
        }
        line = get_prop(target->prop, line_prop);
#ifdef TEAMWARE_MAKE_CMN
        if (line != NULL) {
                /*
                 * If this target is a member of target group and one of the
                 * other members of the group is running, mark this target
                 * as running.
                 */
                for (target_group = line->body.line.target_group;
                     target_group != NULL;
                     target_group = target_group->next) {
                        if (is_running(target_group->name)) {
                                target->state = build_running;
                                add_pending(target,
                                            recursion_level,
                                            do_get,
                                            implicit,
                                            false);
                                return build_running;
                        }
                }
        }
#endif
        /*
         * If the target is a constructed one for a "::" target,
         * we need to consider that.
         */
        if (target->has_target_prop) {
                true_target = get_prop(target->prop,
                                       target_prop)->body.target.target;
                if (true_target->colon_splits > 0) {
                        /* Make sure we have a valid time for :: targets */
                        Property        time;

                        time = get_prop(true_target->prop, time_prop);
                        if (time != NULL) {
                                true_target->stat.time = time->body.time.time;
                        }
                }
        }
        (void) exists(true_target);
        /*
         * If the target has been processed, we don't need to do it again,
         * unless it depends on conditional macros or a delayed assignment,
         * or it has been done when KEEP_STATE is on.
         */
```



```
        if (target->state == build_ok) {
                if((!keep_state || (!target->depends_on_conditional && !assign_d
                        return build_ok;
                } else {
                        recheck_conditionals = true;
                }
        }
        if (target->state == build_subtree) {
                /* A dynamic macro subtree is being built */
                target->state = build_dont_know;
                doing_subtree = true;
                if (!target->checking_subtree) {
                        /*
                         * This target has been started before and therefore
                         * not all dependencies have to be built.
                         */
                        restart = true;
                }
        } else if (target->state == build_pending) {
                target->state = build_dont_know;
                restart = true;
/*
#ifdef TEAMWARE_MAKE_CMN
        } else if (parallel &&
                   keep_state &&
                   (target->conditional_cnt > 0)) {
            if (!parallel_ok(target, false)) {
                add_subtree(target, recursion_level, do_get, implicit);
                target->state = build_running;
                return build_running;
            }
#endif
 */
        }
        /*
         * If KEEP_STATE is on, we have to rebuild the target if the
         * building of it caused new automatic dependencies to be reported.
         * This is where we restart the build.
         */
        if (line != NULL) {
                line->body.line.percent = NULL;
        }
recheck_target:
        /* Init all local variables */
        result = build_dont_know;
        out_of_date_list = NULL;
        command = NULL;
        less = NULL;
        auto_count = 0;
        if (!restart && line != NULL) {
                /*
                 * If this target has never been built before, mark all
                 * of the dependencies as never built.
                 */
                for (dependency = line->body.line.dependencies;
                     dependency != NULL;
                     dependency = dependency->next) {
                        dependency->built = false;
                }
        }
        /* Save the set of automatic depes defined for this target */
        if (keep_state &&
            (line != NULL) &&
            (line->body.line.dependencies != NULL)) {
                Name *p;
```

```
459                 /*
460                  * First run thru the dependency list to see how many
461                  * autos there are.
462                  */
463                 for (dependency = line->body.line.dependencies;
464                      dependency != NULL;
465                      dependency = dependency->next) {
466                         if (dependency->automatic && !dependency->stale) {
467                                 auto_count++;
468                         }
469                 }
470                 /* Create vector to hold the current autos */
471                 automatics =
472                   (Name *) alloca((int) (auto_count * sizeof (Name)));
473                 /* Copy them */
474                 for (p = automatics, dependency = line->body.line.dependencies;
475                      dependency != NULL;
476                      dependency = dependency->next) {
477                         if (dependency->automatic && !dependency->stale) {
478                                 *p++ = dependency->name;
479                         }
480                 }
481         }
482         if (debug_level > 1) {
483                 (void) printf(NOCATGETS("%*sdoname(%s)\n"),
484                               recursion_level,
485                               "",
486                               target->string_mb);
487         }
488         recursion_level++;
489         /* Avoid infinite loops */
490         if (target->state == build_in_progress) {
491                 warning(catgets(catd, 1, 16, "Infinite loop: Target '%s' depends
492                         target->string_mb);
493                 return build_ok;
494         }
495         target->state = build_in_progress;

497         /* Activate conditional macros for the target */
498         if (!target->added_pattern_conditionals) {
499                 add_pattern_conditionals(target);
500                 target->added_pattern_conditionals = true;
501         }
502         if (target->conditional_cnt > 0) {
503                 old_locals = (Property) alloca(target->conditional_cnt *
504                                                sizeof (Property_rec));
505                 set_locals(target, old_locals);
506         }

508 /*
509  * after making the call to dynamic_dependecies unconditional we can handle
510  * target names that are same as file name. In this case $$@ in the
511  * dependencies did not mean anything. WIth this change it expands it
512  * as expected.
513  */
514         if (!target->has_depe_list_expanded)
515         {
516                 dynamic_dependencies(target);
517         }

519 /*
520  *      FIRST SECTION -- GO THROUGH DEPENDENCIES AND COLLECT EXPLICIT
521  *      COMMANDS TO RUN
522  */
523         if ((line = get_prop(target->prop, line_prop)) != NULL) {
524                 if (check_dependencies(&result,
```



```
525                                        line,
526                                        do_get,
527                                        target,
528                                        true_target,
529                                        doing_subtree,
530                                        &out_of_date_list,
531                                        old_locals,
532                                        implicit,
533                                        &command,
534                                        less,
535                                        rechecking_target,
536                                        recheck_conditionals)) {
537                         return build_running;
538                 }
539                 if (line->body.line.query != NULL) {
540                         delete_query_chain(line->body.line.query);
541                 }
542                 line->body.line.query = out_of_date_list;
543         }

550 #ifdef PARALLEL
551         if (doing_subtree) {
552                 parallel = false;
553         }
554 #endif

546 /*
547  * If the target is a :: type, do not try to find the rule for the target,
548  * all actions will be taken by separate branches.
549  * Else, we try to find an implicit rule using various methods,
550  * we quit as soon as one is found.
551  *
552  * [tolik, 12 Sep 2002] Do not try to find implicit rule for the target
553  * being rechecked - the target is being rechecked means that it already
554  * has explicit dependencies derived from an implicit rule found
555  * in previous step.
556  */
557         if (target->colon_splits == 0 && !rechecking_target) {
558                 /* Look for percent matched rule */
559                 if ((result == build_dont_know) &&
560                     (command == NULL)) {
561                         switch (find_percent_rule(
562                                         target,
563                                         &command,
564                                         recheck_conditionals)) {
565                         case build_failed:
566                                 result = build_failed;
567                                 break;
568 #ifdef TEAMWARE_MAKE_CMN
569                         case build_running:
570                                 target->state = build_running;
571                                 add_pending(target,
572                                             --recursion_level,
573                                             do_get,
574                                             implicit,
575                                             false);
576                                 if (target->conditional_cnt > 0) {
577                                         reset_locals(target,
578                                                      old_locals,
579                                                      get_prop(target->prop,
580                                                               conditional_prop),
581                                                      0);
582                                 }
583                                 return build_running;
584 #endif
585                         case build_ok:
```

```
586                                 result = build_ok;
587                                 break;
588                         }
589                 }
590                 /* Look for double suffix rule */
591                 if (result == build_dont_know) {
592                         Property member;
593
594                         if (target->is_member &&
595                             ((member = get_prop(target->prop, member_prop)) !=
596                              NULL)) {
597                                 switch (find_ar_suffix_rule(target,
598                                                 member->body.
599                                                 member.member,
600                                                 &command,
601                                                 recheck_conditionals)) {
602                                 case build_failed:
603                                         result = build_failed;
604                                         break;
605 #ifdef TEAMWARE_MAKE_CMN
606                                 case build_running:
607                                         target->state = build_running;
608                                         add_pending(target,
609                                                     --recursion_level,
610                                                     do_get,
611                                                     implicit,
612                                                     false);
613                                     if (target->conditional_cnt > 0) {
614                                          reset_locals(target,
615                                                       old_locals,
616                                                       get_prop(target->prop,
617                                                           conditional_prop),
618                                                       0);
619                                     }
620                                         return build_running;
621 #endif
622                                 default:
623                                         /* ALWAYS bind $% for old style */
624                                         /* ar rules */
625                                         if (line == NULL) {
626                                                 line =
627                                                   maybe_append_prop(target,
628                                                                     line_prop);
629                                         }
630                                         line->body.line.percent =
631                                           member->body.member.member;
632                                         break;
633                                 }
634                         } else {
635                                 switch (find_double_suffix_rule(target,
636                                                 &command,
637                                                 recheck_conditionals)) {
638                                 case build_failed:
639                                         result = build_failed;
640                                         break;
641 #ifdef TEAMWARE_MAKE_CMN
642                                 case build_running:
643                                         target->state = build_running;
644                                         add_pending(target,
645                                                     --recursion_level,
646                                                     do_get,
647                                                     implicit,
648                                                     false);
649                                         if (target->conditional_cnt > 0) {
650                                                 reset_locals(target,
651                                                              old_locals,
```



```
652                                                              get_prop(target->
653                                                                       prop,
654                                                                       conditiona
655                                                              0);
656                                         }
657                                         return build_running;
658 #endif
659                                 }
660                         }
661                 }
662                 /* Look for single suffix rule */
663
664 /* /tolik/
665  * I commented !implicit to fix bug 1247448: Suffix Rules failed when combine wi
666  * This caused problem with SVR4 tilde rules (infinite recursion). So I made som
667  */
668 /* /tolik, 06.21.96/
669  * Regression! See BugId 1255360
670  * If more than one percent rules are defined for the same target then
671  * the behaviour of 'make' with my previous fix may be different from one
672  * of the 'old make'.
673  * The global variable second_pass (maybe it should be an argument to doname())
674  * is intended to avoid this regression. It is set in doname_check().
675  * First, 'make' will work as it worked before. Only when it is
676  * going to say "don't know how to make target" it sets second_pass to true and
677  * run 'doname' again but now trying to use Single Suffix Rules.
678  */
679                 if ((result == build_dont_know) && !automatic && (!implicit || s
680                     ((line == NULL) ||
681                      ((line->body.line.target != NULL) &&
682                       !line->body.line.target->has_regular_dependency))) {
683                         switch (find_suffix_rule(target,
684                                                  target,
685                                                  empty_name,
686                                                  &command,
687                                                  recheck_conditionals)) {
688                         case build_failed:
689                                 result = build_failed;
690                                 break;
691 #ifdef TEAMWARE_MAKE_CMN
692                         case build_running:
693                                 target->state = build_running;
694                                 add_pending(target,
695                                             --recursion_level,
696                                             do_get,
697                                             implicit,
698                                             false);
699                                 if (target->conditional_cnt > 0) {
700                                         reset_locals(target,
701                                                      old_locals,
702                                                      get_prop(target->prop,
703                                                               conditional_prop),
704                                                      0);
705                                 }
706                                 return build_running;
707 #endif
708                         }
709                 }
710                 /* Try to sccs get */
711                 if ((command == NULL) &&
712                     (result == build_dont_know) &&
713                     do_get) {
714                         result = sccs_get(target, &command);
715                 }
716
717                 /* Use .DEFAULT rule if it is defined. */
```

```
718                 if ((command == NULL) &&
719                     (result == build_dont_know) &&
720                     (true_target->colons == no_colon) &&
721                     default_rule &&
722                     !implicit) {
723                         /* Make sure we have a line prop */
724                         line = maybe_append_prop(target, line_prop);
725                         command = line;
726                         Boolean out_of_date;
727                         if (true_target->is_member) {
728                                 out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
729                                                                         line->bo
730                         } else {
731                                 out_of_date = (Boolean) OUT_OF_DATE(true_target-
732                                                                     line->body.l
733                         }
734                         if (build_unconditional || out_of_date) {
735                                 line->body.line.is_out_of_date = true;
736                                 if (debug_level > 0) {
737                                         (void) printf(catgets(catd, 1, 17, "%*sB
738                                                               recursion_level,
739                                                               "",
740                                                               true_target->string_mb);
741                                 }
742                         }
743                         line->body.line.sccs_command = false;
744                         line->body.line.command_template = default_rule;
745                         line->body.line.target = true_target;
746                         line->body.line.star = NULL;
747                         line->body.line.less = true_target;
748                         line->body.line.percent = NULL;
749                 }
750         }

752         /* We say "target up to date" if no cmd were executed for the target */
753         if (!target->is_double_colon_parent) {
754                 commands_done = false;
755         }

757         silent = silent_all;
758         ignore_errors = ignore_errors_all;
759         if  (posix)
760         {
761           if  (!silent)
762           {
763             silent = (Boolean) target->silent_mode;
764           }
765           if  (!ignore_errors)
766           {
767             ignore_errors = (Boolean) target->ignore_error_mode;
768           }
769         }

771         int doname_dyntarget = 0;
772 r_command:
773         /* Run commands if any. */
774         if ((command != NULL) &&
775             (command->body.line.command_template != NULL)) {
776                 if (result != build_failed) {
777                         result = run_command(command,
778                                              (Boolean) ((parallel || save_parall
779                 }
780                 switch (result) {
781 #ifdef TEAMWARE_MAKE_CMN
782                 case build_running:
783                         add_running(target,
```



```
784                                     true_target,
785                                     command,
786                                     --recursion_level,
787                                     auto_count,
788                                     automatics,
789                                     do_get,
790                                     implicit);
791                         target->state = build_running;
792                         if ((line = get_prop(target->prop,
793                                              line_prop)) != NULL) {
794                                 if (line->body.line.query != NULL) {
795                                         delete_query_chain(line->body.line.query
796                                 }
797                                 line->body.line.query = NULL;
798                         }
799                         if (target->conditional_cnt > 0) {
800                                 reset_locals(target,
801                                              old_locals,
802                                              get_prop(target->prop,
803                                                       conditional_prop),
804                                              0);
805                         }
806                         return build_running;
807                 case build_serial:
808                         add_serial(target,
809                                    --recursion_level,
810                                    do_get,
811                                    implicit);
812                         target->state = build_running;
813                         line = get_prop(target->prop, line_prop);
814                         if (line != NULL) {
815                                 if (line->body.line.query != NULL) {
816                                         delete_query_chain(line->body.line.query
817                                 }
818                                 line->body.line.query = NULL;
819                         }
820                         if (target->conditional_cnt > 0) {
821                                 reset_locals(target,
822                                              old_locals,
823                                              get_prop(target->prop,
824                                                       conditional_prop),
825                                              0);
826                         }
827                         return build_running;
828 #endif
829                 case build_ok:
830                         /* If all went OK set a nice timestamp */
831                         if (true_target->stat.time == file_doesnt_exist) {
832                                 true_target->stat.time = file_max_time;
833                         }
834                         break;
835                 }
836         } else {
837                 /*
838                  * If no command was found for the target, and it doesn't
839                  * exist, and it is mentioned as a target in the makefile,
840                  * we say it is extremely new and that it is OK.
841                  */
842                 if (target->colons != no_colon) {
843                         if (true_target->stat.time == file_doesnt_exist){
844                                 true_target->stat.time = file_max_time;
845                         }
846                         result = build_ok;
847                 }
848                 /*
849                  * Trying dynamic targets.
```

```
850                  */
851                 if(!doname_dyntarget) {
852                         doname_dyntarget = 1;
853                         Name dtarg = find_dyntarget(target);
854                         if(dtarg!=NULL) {
855                                 if (!target->has_depe_list_expanded) {
856                                         dynamic_dependencies(target);
857                                 }
858                                 if ((line = get_prop(target->prop, line_prop)) !
859                                         if (check_dependencies(&result,
860                                                                line,
861                                                                do_get,
862                                                                target,
863                                                                true_target,
864                                                                doing_subtree,
865                                                                &out_of_date_list
866                                                                old_locals,
867                                                                implicit,
868                                                                &command,
869                                                                less,
870                                                                rechecking_target
871                                                                recheck_condition
872                                         {
873                                                 return build_running;
874                                         }
875                                         if (line->body.line.query != NULL) {
876                                                 delete_query_chain(line->body.li
877                                         }
878                                         line->body.line.query = out_of_date_list
879                                 }
880                                 goto r_command;
881                         }
882                 }
883                 /*
884                  * If the file exists, it is OK that we couldnt figure
885                  * out how to build it.
886                  */
887                 (void) exists(target);
888                 if ((target->stat.time != file_doesnt_exist) &&
889                     (result == build_dont_know)) {
890                         result = build_ok;
891                 }
892         }

894         /*
895          * Some of the following is duplicated in the function finish_doname.
896          * If anything is changed here, check to see if it needs to be
897          * changed there.
898          */
899         if ((line = get_prop(target->prop, line_prop)) != NULL) {
900                 if (line->body.line.query != NULL) {
901                         delete_query_chain(line->body.line.query);
902                 }
903                 line->body.line.query = NULL;
904         }
905         target->state = result;
906         parallel = save_parallel;
907         if (target->conditional_cnt > 0) {
908                 reset_locals(target,
909                              old_locals,
910                              get_prop(target->prop, conditional_prop),
911                              0);
912         }
913         recursion_level--;
914         if (target->is_member) {
915                 Property member;
```



```
917                 /* Propagate the timestamp from the member file to the member*/
918                 if ((target->stat.time != file_max_time) &&
919                     ((member = get_prop(target->prop, member_prop)) != NULL) &&
920                     (exists(member->body.member.member) > file_doesnt_exist)) {
921                         target->stat.time =
922                           member->body.member.member->stat.time;
923                 }
924         }
925         /*
926          * Check if we found any new auto dependencies when we
927          * built the target.
928          */
929         if ((result == build_ok) && check_auto_dependencies(target,
930                                                             auto_count,
931                                                             automatics)) {
932                 if (debug_level > 0) {
933                         (void) printf(catgets(catd, 1, 18, "%*sTarget `%s' acqui
934                                       recursion_level,
935                                       "",
936                                       true_target->string_mb);
937                 }
938                 rechecking_target = true;
939                 saved_commands_done = commands_done;
940                 goto recheck_target;
941         }

943         if (rechecking_target && !commands_done) {
944                 commands_done = saved_commands_done;
945         }

947         return result;
948 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   98375 Wed May 20 11:42:02 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for PARALLEL (undefined, this relates to the old pmake)
**********************************************************
_____unchanged_portion_omitted_
 156 #endif

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

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

 162 static  void            append_makeflags_string(Name, String);
 163 static  void            doalarm(int);
 164 static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
 165 static  void            make_targets(int, char **, Boolean);
 166 static  int             parse_command_option(char);
 167 static  void            read_command_options(int, char **);
 168 static  void            read_environment(Boolean);
 169 static  void            read_files_and_state(int, char **);
 170 static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
 171 static  void            report_recursion(Name);
 172 static  void            set_sgs_support(void);
 173 static  void            setup_for_projectdir(void);
 174 static  void            setup_makeflags_argv(void);
 175 static  void            report_dir_enter_leave(Boolean entering);

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

 179 #ifdef DISTRIBUTED
 180         extern  int             dmake_ofd;
 181         extern  FILE*           dmake_ofp;
 182         extern  int             rxmPid;
 183         extern  XDR             xdrs_out;
 184 #endif
 185 #ifdef TEAMWARE_MAKE_CMN
 186         extern  char            verstring[];
 187 #endif

 189 jmp_buf jmpbuffer;
 190 extern nl_catd catd;

 192 /*
 193  *      main(argc, argv)
 194  *
 195  *      Parameters:
 196  *              argc                    You know what this is
 197  *              argv                    You know what this is
 198  *
 199  *      Static variables used:
 200  *              list_all_targets        make -T seen
 201  *              trace_status            make -p seen
 202  *
 203  *      Global variables used:
 204  *              debug_level             Should we trace make actions?
 205  *              keep_state              Set if .KEEP_STATE seen
 206  *              makeflags               The Name "MAKEFLAGS", used to get macro
 207  *              remote_command_name     Name of remote invocation cmd ("on")
 208  *              running_list            List of parallel running processes
 209  *              stdout_stderr_same      true if stdout and stderr are the same
 210  *              auto_dependencies       The Name "SUNPRO_DEPENDENCIES"
 211  *              temp_file_directory     Set to the dir where we create tmp file
 212  *              trace_reader            Set to reflect tracing status
 213  *              working_on_targets      Set when building user targets
 214  */
 215 int
```



```
 216 main(int argc, char *argv[])
 217 {
 218         /*
 219          * cp is a -> to the value of the MAKEFLAGS env var,
 220          * which has to be regular chars.
 221          */
 222         register char           *cp;
 223         char                    make_state_dir[MAXPATHLEN];
 224         Boolean                 parallel_flag = false;
 225         char                    *prognameptr;
 226         char                    *slash_ptr;
 227         mode_t                  um;
 228         int                     i;
 229 #ifdef TEAMWARE_MAKE_CMN
 230         struct itimerval        value;
 231         char                    def_dmakerc_path[MAXPATHLEN];
 232         Name                    dmake_name, dmake_name2;
 233         Name                    dmake_value, dmake_value2;
 234         Property                prop, prop2;
 235         struct stat             statbuf;
 236         int                     statval;
 237 #endif

 239 #ifndef PARALLEL
 239         struct stat             out_stat, err_stat;
 241 #endif
 240         hostid = gethostid();
 241 #ifdef TEAMWARE_MAKE_CMN
 242         avo_get_user(NULL, NULL); // Initialize user name
 243 #endif
 244         bsd_signals();

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


 249 #ifdef DMAKE_STATISTICS
 250         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
 251                 getname_stat = true;
 252         }
 253 #endif


 256         /*
 257          * avo_init() sets the umask to 0.  Save it here and restore
 258          * it after the avo_init() call.
 259          */
 260 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
 261         um = umask(0);
 262         avo_init(argv[0]);
 263         umask(um);

 265         cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
 266 #endif

 268 #if defined(TEAMWARE_MAKE_CMN)
 269         catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
 270         libcli_init();

 272 #ifdef _CHECK_UPDATE_H
 273         /* This is for dmake only (not for Solaris make).
 274          * Check (in background) if there is an update (dmake patch)
 275          * and inform user
 276          */
 277         {
 278                 Avo_err         *err;
 279                 char            *dir;
```

```
280                 err = avo_find_run_dir(&dir);
281                 if (AVO_OK == err) {
282                         AU_check_update_service(NOCATGETS("Dmake"), dir);
283                 }
284         }
285 #endif /* _CHECK_UPDATE_H */
286 #endif

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


291 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
292 /*
293  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
294  * from avo_util library.
295  */
296         libmksdmsi18n_init();
297 #endif


300 #ifndef TEAMWARE_MAKE_CMN
301         textdomain(NOCATGETS("SUNW_SPRO_MAKE"));
302 #endif /* TEAMWARE_MAKE_CMN */

304 #ifdef TEAMWARE_MAKE_CMN
305         g_argc = argc;
306         g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
307         for (i = 0; i < argc; i++) {
308                 g_argv[i] = argv[i];
309         }
310         g_argv[i] = NULL;
311 #endif /* TEAMWARE_MAKE_CMN */

313         /*
314          * Set argv_zero_string to some form of argv[0] for
315          * recursive MAKE builds.
316          */

318         if (*argv[0] == (int) slash_char) {
319                 /* argv[0] starts with a slash */
320                 argv_zero_string = strdup(argv[0]);
321         } else if (strchr(argv[0], (int) slash_char) == NULL) {
322                 /* argv[0] contains no slashes */
323                 argv_zero_string = strdup(argv[0]);
324         } else {
325                 /*
326                  * argv[0] contains at least one slash,
327                  * but doesn't start with a slash
328                  */
329                 char    *tmp_current_path;
330                 char    *tmp_string;

332                 tmp_current_path = get_current_path();
333                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
334                                     strlen(argv[0]) + 1);
335                 (void) sprintf(tmp_string,
336                                "%s/%s",
337                                tmp_current_path,
338                                argv[0]);
339                 argv_zero_string = strdup(tmp_string);
340                 retmem_mb(tmp_string);
341         }

343         /*
344          * The following flags are reset if we don't have the
345          * (.nse_depinfo or .make.state) files locked and only set
```



```
346          * AFTER the file has been locked. This ensures that if the user
347          * interrupts the program while file_lock() is waiting to lock
348          * the file, the interrupt handler doesn't remove a lock
349          * that doesn't belong to us.
350          */
351         make_state_lockfile = NULL;
352         make_state_locked = false;


355         /*
356          * look for last slash char in the path to look at the binary
357          * name. This is to resolve the hard link and invoke make
358          * in svr4 mode.
359          */

361         /* Sun OS make standart */
362         svr4 = false;
363         posix = false;
364         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
365                 svr4 = false;
366                 posix = true;
367         } else {
368                 prognameptr = strrchr(argv[0], '/');
369                 if(prognameptr) {
370                         prognameptr++;
371                 } else {
372                         prognameptr = argv[0];
373                 }
374                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
375                         svr4 = true;
376                         posix = false;
377                 }
378         }
379         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
380            svr4 = true;
381            posix = false;
382         }

384         /*
385          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
386          */
387         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
388         if (dmake_compat_mode_var != NULL) {
389                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
390                         gnu_style = true;
391                 }
392                 //svr4 = false;
393                 //posix = false;
394         }

396         /*
397          * Temporary directory set up.
398          */
399         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
400         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
401                 strcpy(mbs_buffer, tmpdir_var);
402                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
403                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
404                         *tmpdir_var = '\0');
405                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
406                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
407                                 mbs_buffer, getpid());
408                         int fd = mkstemp(mbs_buffer2);
409                         if (fd >= 0) {
410                                 close(fd);
411                                 unlink(mbs_buffer2);
```

```
412                                 tmpdir = strdup(mbs_buffer);
413                         }
414                 }
415         }

419 #ifndef PARALLEL
417         /* find out if stdout and stderr point to the same place */
418         if (fstat(1, &out_stat) < 0) {
419                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
420         }
421         if (fstat(2, &err_stat) < 0) {
422                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
423         }
424         if ((out_stat.st_dev == err_stat.st_dev) &&
425             (out_stat.st_ino == err_stat.st_ino)) {
426                 stdout_stderr_same = true;
427         } else {
428                 stdout_stderr_same = false;
429         }
433 #else
434         stdout_stderr_same = false;
435 #endif
430         /* Make the vroot package scan the path using shell semantics */
431         set_path_style(0);

433         setup_char_semantics();

435         setup_for_projectdir();

437         /*
438          * If running with .KEEP_STATE, curdir will be set with
439          * the connected directory.
440          */
441         (void) atexit(cleanup_after_exit);

443         load_cached_names();

445 /*
446  *      Set command line flags
447  */
448         setup_makeflags_argv();
449         read_command_options(mf_argc, mf_argv);
450         read_command_options(argc, argv);
451         if (debug_level > 0) {
452                 cp = getenv(makeflags->string_mb);
453                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
454         }

456         setup_interrupt(handle_interrupt);

458         read_files_and_state(argc, argv);

460 #ifdef TEAMWARE_MAKE_CMN
461         /*
462          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
463          */
464         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
465         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
466         prop2 = get_prop(dmake_name2->prop, macro_prop);
467         if (prop2 == NULL) {
468                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
469                 output_mode = txt1_mode;
470         } else {
471                 dmake_value2 = prop2->body.macro.value;
472                 if ((dmake_value2 == NULL) ||
473                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
```



```
474                         output_mode = txt1_mode;
475                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
476                         output_mode = txt2_mode;
477                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
478                         output_mode = html1_mode;
479                 } else {
480                         warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
481                               dmake_value2->string_mb);
482                 }
483         }
484         /*
485          * Find the dmake_mode: distributed, parallel, or serial.
486          */
487     if ((!pmake_cap_r_specified) &&
488         (!pmake_machinesfile_specified)) {
489         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
490         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
491         prop2 = get_prop(dmake_name2->prop, macro_prop);
492         if (prop2 == NULL) {
493                 /* DMAKE_MODE not defined, default to distributed mode */
494                 dmake_mode_type = distributed_mode;
495                 no_parallel = false;
496         } else {
497                 dmake_value2 = prop2->body.macro.value;
498                 if ((dmake_value2 == NULL) ||
499                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
500                         dmake_mode_type = distributed_mode;
501                         no_parallel = false;
502                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
503                         dmake_mode_type = parallel_mode;
504                         no_parallel = false;
505                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
506                         dmake_mode_type = serial_mode;
507                         no_parallel = true;
508                 } else {
509                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
510                 }
511         }

513         if ((!list_all_targets) &&
514             (report_dependencies_level == 0)) {
515                 /*
516                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
517                  * They could be defined in the env, in the makefile, or on the
518                  * command line.
519                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
520                  * then print a message, and default to parallel mode.
521                  */
522 #ifdef DISTRIBUTED
523                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
524                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
525                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
526                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
527                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
528                      ((dmake_value = prop->body.macro.value) == NULL)) &&
529                     (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
530                      ((dmake_value2 = prop2->body.macro.value) == NULL))) {
531                         Boolean empty_dmakerc = true;
532                         char *homedir = getenv(NOCATGETS("HOME"));
533                         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
534                                 sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
535                                 if ((((statval = stat(def_dmakerc_path, &statbuf
536                                       ((statval == 0) && (statbuf.st_size == 0
537                                 } else {
538                                       Avo_dmakerc     *rcfile = new Avo_dmaker
539                                       Avo_err         *err = rcfile->read(def_
```

```
540                                         if (err) {
541                                                 fatal(err->str);
542                                         }
543                                         empty_dmakerc = rcfile->was_empty();
544                                         delete rcfile;
545                                 }
546                         }
547                         if (empty_dmakerc) {
548                                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
549                                         putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
550                                         (void) fprintf(stdout, catgets(catd, 1,
551                                         (void) fprintf(stdout, catgets(catd, 1,
552                                 }
553                                 dmake_mode_type = parallel_mode;
554                                 no_parallel = false;
555                         }
556                 }
557 #else
558                 if(dmake_mode_type == distributed_mode) {
559                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
560                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
561                         dmake_mode_type = parallel_mode;
562                         no_parallel = false;
563                 }
564 #endif  /* DISTRIBUTED */
565         }
566     }
567 #endif

569 #ifdef TEAMWARE_MAKE_CMN
570         parallel_flag = true;
571         /* XXX - This is a major hack for DMake/Licensing. */
572         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
573                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
574                         /*
575                          * If the user can not get a TeamWare license,
576                          * default to serial mode.
577                          */
578                         dmake_mode_type = serial_mode;
579                         no_parallel = true;
580                 } else {
581                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
582                 }
583                 start_time = time(NULL);
584                 /*
585                  * XXX - Hack to disable SIGALRM's from licensing library's
586                  *       setitimer().
587                  */
588                 value.it_interval.tv_sec = 0;
589                 value.it_interval.tv_usec = 0;
590                 value.it_value.tv_sec = 0;
591                 value.it_value.tv_usec = 0;
592                 (void) setitimer(ITIMER_REAL, &value, NULL);
593         }

595 //
596 // If dmake is running with -t option, set dmake_mode_type to serial.
597 // This is done because doname() calls touch_command() that runs serially.
598 // If we do not do that, maketool will have problems.
599 //
600         if(touch) {
601                 dmake_mode_type = serial_mode;
602                 no_parallel = true;
603         }
604 #else
605         parallel_flag = false;
```



```
606 #endif

608 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
609         /*
610          * Check whether stdout and stderr are physically same.
611          * This is in order to decide whether we need to redirect
612          * stderr separately from stdout.
613          * This check is performed only if __DMAKE_SEPARATE_STDERR
614          * is not set. This variable may be used in order to preserve
615          * the 'old' behaviour.
616          */
617         out_err_same = true;
618         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
619         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
620                 struct stat stdout_stat;
621                 struct stat stderr_stat;
622                 if( (fstat(1, &stdout_stat) == 0)
623                  && (fstat(2, &stderr_stat) == 0) )
624                 {
625                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
626                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
627                         {
628                                 out_err_same = false;
629                         }
630                 }
631         }
632 #endif

634 #ifdef DISTRIBUTED
635         /*
636          * At this point, DMake should startup an rxm with any and all
637          * DMake command line options. Rxm will, among other things,
638          * read the rc file.
639          */
640         if ((!list_all_targets) &&
641             (report_dependencies_level == 0) &&
642             (dmake_mode_type == distributed_mode)) {
643                 startup_rxm();
644         }
645 #endif
646
647 /*
648  *      Enable interrupt handler for alarms
649  */
650         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

652 /*
653  *      Check if make should report
654  */
655         if (getenv(sunpro_dependencies->string_mb) != NULL) {
656                 FILE    *report_file;

658                 report_dependency("");
659                 report_file = get_report_file();
660                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
661                         (void) fprintf(report_file, "\n");
662                 }
663         }

665 /*
666  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
667  */
668         if (keep_state) {
669                 maybe_append_prop(sunpro_dependencies, macro_prop)->
670                   body.macro.exported = true;
671         } else {
```

```
672                 maybe_append_prop(sunpro_dependencies, macro_prop)->
673                   body.macro.exported = false;
674         }

676         working_on_targets = true;
677         if (trace_status) {
678                 dump_make_state();
679                 fclose(stdout);
680                 fclose(stderr);
681                 exit_status = 0;
682                 exit(0);
683         }
684         if (list_all_targets) {
685                 dump_target_list();
686                 fclose(stdout);
687                 fclose(stderr);
688                 exit_status = 0;
689                 exit(0);
690         }
691         trace_reader = false;

693         /*
694          * Set temp_file_directory to the directory the .make.state
695          * file is written to.
696          */
697         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
698                 temp_file_directory = strdup(get_current_path());
699         } else {
700                 *slash_ptr = (int) nul_char;
701                 (void) strcpy(make_state_dir, make_state->string_mb);
702                 *slash_ptr = (int) slash_char;
703                    /* when there is only one slash and it's the first
704                    ** character, make_state_dir should point to '/'.
705                    */
706                 if(make_state_dir[0] == '\0') {
707                    make_state_dir[0] = '/';
708                    make_state_dir[1] = '\0';
709                 }
710                 if (make_state_dir[0] == (int) slash_char) {
711                         temp_file_directory = strdup(make_state_dir);
712                 } else {
713                         char    tmp_current_path2[MAXPATHLEN];
714
715                         (void) sprintf(tmp_current_path2,
716                                        "%s/%s",
717                                        get_current_path(),
718                                        make_state_dir);
719                         temp_file_directory = strdup(tmp_current_path2);
720                 }
721         }

723 #ifdef DISTRIBUTED
724         building_serial = false;
725 #endif

727         report_dir_enter_leave(true);

729         make_targets(argc, argv, parallel_flag);

731         report_dir_enter_leave(false);

733         if (build_failed_ever_seen) {
734                 if (posix) {
735                         exit_status = 1;
736                 }
737                 exit(1);
```



```
738         }
739         exit_status = 0;
740         exit(0);
741         /* NOTREACHED */
742 }
```

_____***unchanged_portion_omitted_***

```
1116 /*
1117  *      read_command_options(argc, argv)
1118  *
1119  *      Scan the cmd line options and process the ones that start with "-"
1120  *
1121  *      Return value:
1122  *                              -M argument, if any
1123  *
1124  *      Parameters:
1125  *              argc            You know what this is
1126  *              argv            You know what this is
1127  *
1128  *      Global variables used:
1129  */
1130 static void
1131 read_command_options(register int argc, register char **argv)
1132 {
1133         register int            ch;
1134         int                     current_optind = 1;
1135         int                     last_optind_with_double_hyphen = 0;
1136         int                     last_optind;
1137         int                     last_current_optind;
1138         register int            i;
1139         register int            j;
1140         register int            k;
1141         register int            makefile_next = 0; /*
1142                                                     * flag to note options:
1143                                                     * -c, f, g, j, m, o
1144                                                     */
1145         const char              *tptr;
1146         const char              *CMD_OPTS;

1148         extern char             *optarg;
1149         extern int              optind, opterr, optopt;

1151 #define SUNPRO_CMD_OPTS "-~Bbc:Ddef:g:ij:K:kM:m:NnO:o:PpqRrSsTtuVvwx:"

1153 #ifdef TEAMWARE_MAKE_CMN
1154 #       define SVR4_CMD_OPTS   "-c:ef:g:ij:km:nO:o:pqrsTtVv"
1155 #else
1156 #       define SVR4_CMD_OPTS   "-ef:iknpqrstV"
1157 #endif

1159         /*
1160          * Added V in SVR4_CMD_OPTS also, which is going to be a hidden
1161          * option, just to make sure that the getopt doesn't fail when some
1162          * users leave their USE_SVR4_MAKE set and try to use the makefiles
1163          * that are designed to issue commands like $(MAKE) -V. Anyway it
1164          * sets the same flag but ensures that getopt doesn't fail.
1165          */

1167         opterr = 0;
1168         optind = 1;
1169         while (1) {
1170                 last_optind=optind;                     /* Save optind and curre
1171                 last_current_optind=current_optind;     /* in case we have to re
1172                 if (svr4) {
1173                         CMD_OPTS=SVR4_CMD_OPTS;
```

```
1174                    ch = getopt(argc, argv, SVR4_CMD_OPTS);
1175            } else {
1176                    CMD_OPTS=SUNPRO_CMD_OPTS;
1177                    ch = getopt(argc, argv, SUNPRO_CMD_OPTS);
1178            }
1179            if (ch == EOF) {
1180                    if(optind < argc) {
1181                            /*
1182                             * Fixing bug 4102537:
1183                             *    Strange behaviour of command make using --
1184                             * Not all argv have been processed
1185                             * Skip non-flag argv and continue processing.
1186                             */
1187                            optind++;
1188                            current_optind++;
1189                            continue;
1190                    } else {
1191                            break;
1192                    }

1194            }
1195            if (ch == '?') {
1196                    if (optopt == '-') {
1197                            /* Bug 5060758: getopt() changed behavior (s10_6
1198                             * and now we have to deal with cases when optio
1199                             * with double hyphen appear here, from -$(MAKEF
1200                             */
1201                            i = current_optind;
1202                            if (argv[i][0] == '-') {
1203                              if (argv[i][1] == '-') {
1204                                if (argv[i][2] != '\0') {
1205                                  /* Check if this option is allowed */
1206                                  tptr = strchr(CMD_OPTS, argv[i][2]);
1207                                  if (tptr) {
1208                                    if (last_optind_with_double_hyphen != cu
1209                                      /* This is first time we are trying to
1210                                       * problem with this option. If we com
1211                                       * time, we will go to fatal error.
1212                                       */
1213                                      last_optind_with_double_hyphen = curre
1214
1215                                      /* Eliminate first hyphen character */
1216                                      for (j=0; argv[i][j] != '\0'; j++) {
1217                                        argv[i][j] = argv[i][j+1];
1218                                      }
1219
1220                                      /* Repeat the processing of this argum
1221                                      optind=last_optind;
1222                                      current_optind=last_current_optind;
1223                                      continue;
1224                                    }
1225                                  }
1226                                }
1227                              }
1228                            }
1229                    }
1230            }

1232            if (ch == '?') {
1233                    if (svr4) {
1234 #ifdef TEAMWARE_MAKE_CMN
1235                            fprintf(stderr,
1236                                    catgets(catd, 1, 267, "Usage : dmake [ -
1237                            fprintf(stderr,
1238                                    catgets(catd, 1, 268, "             [ -
1239                            fprintf(stderr,
```



```
1240                                    catgets(catd, 1, 269, "             [ -
1241 #else
1242                            fprintf(stderr,
1243                                    catgets(catd, 1, 270, "Usage : make [ -f
1244                            fprintf(stderr,
1245                                    catgets(catd, 1, 271, "             [ -s
1246 #endif
1247                            tptr = strchr(SVR4_CMD_OPTS, optopt);
1248                    } else {
1249 #ifdef TEAMWARE_MAKE_CMN
1250                            fprintf(stderr,
1251                                    catgets(catd, 1, 272, "Usage : dmake [ -
1252                            fprintf(stderr,
1253                                    catgets(catd, 1, 273, "             [ -
1254                            fprintf(stderr,
1255                                    catgets(catd, 1, 274, "             [ -
1256                            fprintf(stderr,
1257                                    catgets(catd, 1, 275, "             [ -
1258 #else
1259                            fprintf(stderr,
1260                                    catgets(catd, 1, 276, "Usage : make [ -f
1261                            fprintf(stderr,
1262                                    catgets(catd, 1, 277, "             [ -e
1263                            fprintf(stderr,
1264                                    catgets(catd, 1, 278, "             [ -u
1265 #endif
1266                            tptr = strchr(SUNPRO_CMD_OPTS, optopt);
1267                    }
1268                    if (!tptr) {
1269                            fatal(catgets(catd, 1, 279, "Unknown option '-%c
1270                    } else {
1271                            fatal(catgets(catd, 1, 280, "Missing argument af
1272                    }
1273            }



1277            makefile_next |= parse_command_option(ch);
1278            /*
1279             * If we're done processing all of the options of
1280             * ONE argument string...
1281             */
1282            if (current_optind < optind) {
1283                    i = current_optind;
1284                    k = 0;
1285                    /* If there's an argument for an option... */
1286                    if ((optind - current_optind) > 1) {
1287                            k = i + 1;
1288                    }
1289                    switch (makefile_next) {
1290                    case 0:
1291                            argv[i] = NULL;
1292                            /* This shouldn't happen */
1293                            if (k) {
1294                                    argv[k] = NULL;
1295                            }
1296                            break;
1297                    case 1: /* -f seen */
1298                            argv[i] = (char *)NOCATGETS("-f");
1299                            break;
1300                    case 2: /* -c seen */
1301                            argv[i] = (char *)NOCATGETS("-c");
1302 #ifndef TEAMWARE_MAKE_CMN
1303                            warning(catgets(catd, 1, 281, "Ignoring Distribu
1304 #endif
1305                            break;
```

```
1306                            case 4: /* -g seen */
1307                                    argv[i] = (char *)NOCATGETS("-g");
1308 #ifndef TEAMWARE_MAKE_CMN
1309                                    warning(catgets(catd, 1, 282, "Ignoring Distribu
1310 #endif
1311                                    break;
1312                            case 8: /* -j seen */
1313                                    argv[i] = (char *)NOCATGETS("-j");
1314 #ifndef TEAMWARE_MAKE_CMN
1315                                    warning(catgets(catd, 1, 283, "Ignoring Distribu
1316 #endif
1317                                    break;
1318                            case 16: /* -M seen */
1319                                    argv[i] = (char *)NOCATGETS("-M");
1320 #ifndef TEAMWARE_MAKE_CMN
1321                                    warning(catgets(catd, 1, 284, "Ignoring Parallel
1322 #endif
1323                                    break;
1324                            case 32: /* -m seen */
1325                                    argv[i] = (char *)NOCATGETS("-m");
1326 #ifndef TEAMWARE_MAKE_CMN
1327                                    warning(catgets(catd, 1, 285, "Ignoring Distribu
1328 #endif
1329                                    break;
1336 #ifndef PARALLEL
1330                            case 128: /* -O seen */
1331                                    argv[i] = (char *)NOCATGETS("-O");
1332                                    break;
1340 #endif
1333                            case 256: /* -K seen */
1334                                    argv[i] = (char *)NOCATGETS("-K");
1335                                    break;
1336                            case 512:       /* -o seen */
1337                                    argv[i] = (char *)NOCATGETS("-o");
1338 #ifndef TEAMWARE_MAKE_CMN
1339                                    warning(catgets(catd, 1, 311, "Ignoring Distribu
1340 #endif
1341                                    break;
1342                            case 1024: /* -x seen */
1343                                    argv[i] = (char *)NOCATGETS("-x");
1344 #ifndef TEAMWARE_MAKE_CMN
1345                                    warning(catgets(catd, 1, 353, "Ignoring Distribu
1346 #endif
1347                                    break;
1348                            default: /* > 1 of -c, f, g, j, K, M, m, O, o, x seen */
1349                                    fatal(catgets(catd, 1, 286, "Illegal command lin
1350                            }

1352                            makefile_next = 0;
1353                            current_optind = optind;
1354                    }
1355            }
1356 }
```

unchanged_portion_omitted_

```
1538 /*
1539  *      parse_command_option(ch)
1540  *
1541  *      Parse make command line options.
1542  *
1543  *      Return value:
1544  *                              Indicates if any -f -c or -M were seen
1545  *
1546  *      Parameters:
1547  *              ch              The character to parse
1548  *
```



```
1549  *      Static variables used:
1550  *              dmake_group_specified   Set for make -g
1551  *              dmake_max_jobs_specified        Set for make -j
1552  *              dmake_mode_specified    Set for make -m
1553  *              dmake_add_mode_specified        Set for make -x
1554  *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
1555  *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
1556  *              dmake_odir_specified    Set for make -o
1557  *              dmake_rcfile_specified  Set for make -c
1558  *              env_wins                Set for make -e
1559  *              ignore_default_mk       Set for make -r
1560  *              trace_status            Set for make -p
1561  *
1562  *      Global variables used:
1563  *              .make.state path & name set for make -K
1564  *              continue_after_error    Set for make -k
1565  *              debug_level             Set for make -d
1566  *              do_not_exec_rule        Set for make -n
1567  *              filter_stderr           Set for make -X
1568  *              ignore_errors_all       Set for make -i
1569  *              no_parallel             Set for make -R
1570  *              quest                   Set for make -q
1571  *              read_trace_level        Set for make -D
1572  *              report_dependencies     Set for make -P
1573  *              send_mtool_msgs         Set for make -K
1574  *              silent_all              Set for make -s
1575  *              touch                   Set for make -t
1576  */
1577 static int
1578 parse_command_option(register char ch)
1579 {
1580         static int              invert_next = 0;
1581         int                     invert_this = invert_next;

1583         invert_next = 0;
1584         switch (ch) {
1585         case '-':                       /* Ignore "--" */
1586                 return 0;
1587         case '~':                       /* Invert next option */
1588                 invert_next = 1;
1589                 return 0;
1590         case 'B':                       /* Obsolete */
1591                 return 0;
1592         case 'b':                       /* Obsolete */
1593                 return 0;
1594         case 'c':                       /* Read alternative dmakerc file */
1595                 if (invert_this) {
1596                         dmake_rcfile_specified = false;
1597                 } else {
1598                         dmake_rcfile_specified = true;
1599                 }
1600                 return 2;
1601         case 'D':                       /* Show lines read */
1602                 if (invert_this) {
1603                         read_trace_level--;
1604                 } else {
1605                         read_trace_level++;
1606                 }
1607                 return 0;
1608         case 'd':                       /* Debug flag */
1609                 if (invert_this) {
1610                         debug_level--;
1611                 } else {
1612                         debug_level++;
1613                 }
1614                 return 0;
```

```
1615         case 'e':                          /* Environment override flag */
1616                 if (invert_this) {
1617                         env_wins = false;
1618                 } else {
1619                         env_wins = true;
1620                 }
1621                 return 0;
1622         case 'f':                          /* Read alternative makefile(s) */
1623                 return 1;
1624         case 'g':                          /* Use alternative DMake group */
1625                 if (invert_this) {
1626                         dmake_group_specified = false;
1627                 } else {
1628                         dmake_group_specified = true;
1629                 }
1630                 return 4;
1631         case 'i':                          /* Ignore errors */
1632                 if (invert_this) {
1633                         ignore_errors_all = false;
1634                 } else {
1635                         ignore_errors_all = true;
1636                 }
1637                 return 0;
1638         case 'j':                          /* Use alternative DMake max jobs */
1639                 if (invert_this) {
1640                         dmake_max_jobs_specified = false;
1641                 } else {
1642                         dmake_max_jobs_specified = true;
1643                 }
1644                 return 8;
1645         case 'K':                          /* Read alternative .make.state */
1646                 return 256;
1647         case 'k':                          /* Keep making even after errors */
1648                 if (invert_this) {
1649                         continue_after_error = false;
1650                 } else {
1651                         continue_after_error = true;
1652                         continue_after_error_ever_seen = true;
1653                 }
1654                 return 0;
1655         case 'M':                          /* Read alternative make.machines file
1656                 if (invert_this) {
1657                         pmake_machinesfile_specified = false;
1658                 } else {
1659                         pmake_machinesfile_specified = true;
1660                         dmake_mode_type = parallel_mode;
1661                         no_parallel = false;
1662                 }
1663                 return 16;
1664         case 'm':                          /* Use alternative DMake build mode */
1665                 if (invert_this) {
1666                         dmake_mode_specified = false;
1667                 } else {
1668                         dmake_mode_specified = true;
1669                 }
1670                 return 32;
1671         case 'x':                          /* Use alternative DMake mode */
1672                 if (invert_this) {
1673                         dmake_add_mode_specified = false;
1674                 } else {
1675                         dmake_add_mode_specified = true;
1676                 }
1677                 return 1024;
1678         case 'N':                          /* Reverse -n */
1679                 if (invert_this) {
1680                         do_not_exec_rule = true;
```



```
1681                 } else {
1682                         do_not_exec_rule = false;
1683                 }
1684                 return 0;
1685         case 'n':                          /* Print, not exec commands */
1686                 if (invert_this) {
1687                         do_not_exec_rule = false;
1688                 } else {
1689                         do_not_exec_rule = true;
1690                 }
1691                 return 0;
1700 #ifndef PARALLEL
1692         case 'O':                          /* Send job start & result msgs */
1693                 if (invert_this) {
1694                         send_mtool_msgs = false;
1695                 } else {
1696 #ifdef DISTRIBUTED
1697                         send_mtool_msgs = true;
1698 #endif
1699                 }
1700                 return 128;
1710 #endif
1701         case 'o':                          /* Use alternative dmake output dir */
1702                 if (invert_this) {
1703                         dmake_odir_specified = false;
1704                 } else {
1705                         dmake_odir_specified = true;
1706                 }
1707                 return 512;
1708         case 'P':                          /* Print for selected targets */
1709                 if (invert_this) {
1710                         report_dependencies_level--;
1711                 } else {
1712                         report_dependencies_level++;
1713                 }
1714                 return 0;
1715         case 'p':                          /* Print description */
1716                 if (invert_this) {
1717                         trace_status = false;
1718                         do_not_exec_rule = false;
1719                 } else {
1720                         trace_status = true;
1721                         do_not_exec_rule = true;
1722                 }
1723                 return 0;
1724         case 'q':                          /* Question flag */
1725                 if (invert_this) {
1726                         quest = false;
1727                 } else {
1728                         quest = true;
1729                 }
1730                 return 0;
1731         case 'R':                          /* Don't run in parallel */
1732 #ifdef TEAMWARE_MAKE_CMN
1733                 if (invert_this) {
1734                         pmake_cap_r_specified = false;
1735                         no_parallel = false;
1736                 } else {
1737                         pmake_cap_r_specified = true;
1738                         dmake_mode_type = serial_mode;
1739                         no_parallel = true;
1740                 }
1741 #else
1742                 warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option")
1743 #endif
1744                 return 0;
```

```
1745         case 'r':                          /* Turn off internal rules */
1746                 if (invert_this) {
1747                         ignore_default_mk = false;
1748                 } else {
1749                         ignore_default_mk = true;
1750                 }
1751                 return 0;
1752         case 'S':                          /* Reverse -k */
1753                 if (invert_this) {
1754                         continue_after_error = true;
1755                 } else {
1756                         continue_after_error = false;
1757                         stop_after_error_ever_seen = true;
1758                 }
1759                 return 0;
1760         case 's':                          /* Silent flag */
1761                 if (invert_this) {
1762                         silent_all = false;
1763                 } else {
1764                         silent_all = true;
1765                 }
1766                 return 0;
1767         case 'T':                          /* Print target list */
1768                 if (invert_this) {
1769                         list_all_targets = false;
1770                         do_not_exec_rule = false;
1771                 } else {
1772                         list_all_targets = true;
1773                         do_not_exec_rule = true;
1774                 }
1775                 return 0;
1776         case 't':                          /* Touch flag */
1777                 if (invert_this) {
1778                         touch = false;
1779                 } else {
1780                         touch = true;
1781                 }
1782                 return 0;
1783         case 'u':                          /* Unconditional flag */
1784                 if (invert_this) {
1785                         build_unconditional = false;
1786                 } else {
1787                         build_unconditional = true;
1788                 }
1789                 return 0;
1790         case 'V':                         /* SVR4 mode */
1791                 svr4 = true;
1792                 return 0;
1793         case 'v':                         /* Version flag */
1794                 if (invert_this) {
1795                 } else {
1796 #ifdef TEAMWARE_MAKE_CMN
1797                         fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1798                         exit_status = 0;
1799                         exit(0);
1800 #else
1801                         warning(catgets(catd, 1, 324, "Ignoring DistributedMake
1802 #endif
1803                 }
1804                 return 0;
1805         case 'w':                          /* Unconditional flag */
1806                 if (invert_this) {
1807                         report_cwd = false;
1808                 } else {
1809                         report_cwd = true;
1810                 }
```



```
1811                 return 0;
1812 #if 0
1813         case 'X':                          /* Filter stdout */
1814                 if (invert_this) {
1815                         filter_stderr = false;
1816                 } else {
1817                         filter_stderr = true;
1818                 }
1819                 return 0;
1820 #endif
1821         default:
1822                 break;
1823         }
1824         return 0;
1825 }
______unchanged_portion_omitted_

1938 /*
1939  *      read_files_and_state(argc, argv)
1940  *
1941  *      Read the makefiles we care about and the environment
1942  *      Also read the = style command line options
1943  *
1944  *      Parameters:
1945  *              argc            You know what this is
1946  *              argv            You know what this is
1947  *
1948  *      Static variables used:
1949  *              env_wins        make -e, determines if env vars are RO
1950  *              ignore_default_mk make -r, determines if make.rules is read
1951  *              not_auto_depen  dwight
1952  *
1953  *      Global variables used:
1954  *              default_target_to_build Set to first proper target from file
1955  *              do_not_exec_rule Set to false when makfile is made
1956  *              dot             The Name ".", used to read current dir
1957  *              empty_name      The Name "", use as macro value
1958  *              keep_state      Set if KEEP_STATE is in environment
1959  *              make_state      The Name ".make.state", used to read file
1960  *              makefile_type   Set to type of file being read
1961  *              makeflags       The Name "MAKEFLAGS", used to set macro value
1962  *              not_auto        dwight
1963  *              read_trace_level Checked to se if the reader should trace
1964  *              report_dependencies If -P is on we do not read .make.state
1965  *              trace_reader    Set if reader should trace
1966  *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
1967  */
1968 static void
1969 read_files_and_state(int argc, char **argv)
1970 {
1971         wchar_t                 buffer[1000];
1972         wchar_t                 buffer_posix[1000];
1973         register char           ch;
1974         register char           *cp;
1975         Property                def_make_macro = NULL;
1976         Name                    def_make_name;
1977         Name                    default_makefile;
1978         String_rec              dest;
1979         wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
1980         register int            i;
1981         register int            j;
1982         Name                    keep_state_name;
1983         int                     length;
1984         Name                    Makefile;
1985         register Property       macro;
1986         struct stat             make_state_stat;
```

```
1987         Name                    makefile_name;
1988         register int            makefile_next = 0;
1989         register Boolean        makefile_read = false;
1990         String_rec              makeflags_string;
1991         String_rec              makeflags_string_posix;
1992         String_rec *            makeflags_string_current;
1993         Name                    makeflags_value_saved;
1994         register Name           name;
1995         Name                    new_make_value;
1996         Boolean                 save_do_not_exec_rule;
1997         Name                    sdotMakefile;
1998         Name                    sdotmakefile_name;
1999         static wchar_t          state_file_str;
2000         static char             state_file_str_mb[MAXPATHLEN];
2001         static struct _Name     state_filename;
2002         Boolean                 temp;
2003         char                    tmp_char;
2004         wchar_t                 *tmp_wcs_buffer;
2005         register Name           value;
2006         ASCII_Dyn_Array         makeflags_and_macro;
2007         Boolean                 is_xpg4;

2009 /*
2010  *      Remember current mode. It may be changed after reading makefile
2011  *      and we will have to correct MAKEFLAGS variable.
2012  */
2013         is_xpg4 = posix;

2015         MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
2016         keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
2017         MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
2018         Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2019         MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
2020         makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2021         MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
2022         sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2023         MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
2024         sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

2026 /*
2027  *      Set flag if NSE is active
2028  */

2030 /*
2031  *      initialize global dependency entry for .NOT_AUTO
2032  */
2033         not_auto_depen->next = NULL;
2034         not_auto_depen->name = not_auto;
2035         not_auto_depen->automatic = not_auto_depen->stale = false;

2037 /*
2038  *      Read internal definitions and rules.
2039  */
2040         if (read_trace_level > 1) {
2041                 trace_reader = true;
2042         }
2043         if (!ignore_default_mk) {
2044                 if (svr4) {
2045                         MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
2046                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2047                 } else {
2048                         MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
2049                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2050                 }
2051                 default_makefile->stat.is_file = true;
```



```
2053                 (void) read_makefile(default_makefile,
2054                                      true,
2055                                      false,
2056                                      true);
2057         }

2059         /*
2060          * If the user did not redefine the MAKE macro in the
2061          * default makefile (make.rules), then we'd like to
2062          * change the macro value of MAKE to be some form
2063          * of argv[0] for recursive MAKE builds.
2064          */
2065         MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
2066         def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2067         def_make_macro = get_prop(def_make_name->prop, macro_prop);
2068         if ((def_make_macro != NULL) &&
2069             (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
2070                       NOCATGETS("make")))) {
2071                 MBSTOWCS(wcs_buffer, argv_zero_string);
2072                 new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2073                 (void) SETVAR(def_make_name,
2074                               new_make_value,
2075                               false);
2076         }

2078         default_target_to_build = NULL;
2079         trace_reader = false;

2081 /*
2082  *      Read environment args. Let file args which follow override unless
2083  *      -e option seen. If -e option is not mentioned.
2084  */
2085         read_environment(env_wins);
2086         if (getvar(virtual_root)->hash.length == 0) {
2087                 maybe_append_prop(virtual_root, macro_prop)
2088                   ->body.macro.exported = true;
2089                 MBSTOWCS(wcs_buffer, "/");
2090                 (void) SETVAR(virtual_root,
2091                               GETNAME(wcs_buffer, FIND_LENGTH),
2092                               false);
2093         }

2095 /*
2096  * We now scan mf_argv and argv to see if we need to set
2097  * any of the DMake-added options/variables in MAKEFLAGS.
2098  */

2100         makeflags_and_macro.start = 0;
2101         makeflags_and_macro.size = 0;
2102         enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
2103         enter_argv_values(argc, argv, &makeflags_and_macro);

2105 /*
2106  *      Set MFLAGS and MAKEFLAGS
2107  *
2108  *      Before reading makefile we do not know exactly which mode
2109  *      (posix or not) is used. So prepare two MAKEFLAGS strings
2110  *      for both posix and solaris modes because they are different.
2111  */
2112         INIT_STRING_FROM_STACK(makeflags_string, buffer);
2113         INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
2114         append_char((int) hyphen_char, &makeflags_string);
2115         append_char((int) hyphen_char, &makeflags_string_posix);

2117         switch (read_trace_level) {
2118         case 2:
```

```
2119                    append_char('D', &makeflags_string);
2120                    append_char('D', &makeflags_string_posix);
2121            case 1:
2122                    append_char('D', &makeflags_string);
2123                    append_char('D', &makeflags_string_posix);
2124            }
2125            switch (debug_level) {
2126            case 2:
2127                    append_char('d', &makeflags_string);
2128                    append_char('d', &makeflags_string_posix);
2129            case 1:
2130                    append_char('d', &makeflags_string);
2131                    append_char('d', &makeflags_string_posix);
2132            }
2133            if (env_wins) {
2134                    append_char('e', &makeflags_string);
2135                    append_char('e', &makeflags_string_posix);
2136            }
2137            if (ignore_errors_all) {
2138                    append_char('i', &makeflags_string);
2139                    append_char('i', &makeflags_string_posix);
2140            }
2141            if (continue_after_error) {
2142                    if (stop_after_error_ever_seen) {
2143                            append_char('S', &makeflags_string_posix);
2144                            append_char((int) space_char, &makeflags_string_posix);
2145                            append_char((int) hyphen_char, &makeflags_string_posix);
2146                    }
2147                    append_char('k', &makeflags_string);
2148                    append_char('k', &makeflags_string_posix);
2149            } else {
2150                    if (stop_after_error_ever_seen
2151                        && continue_after_error_ever_seen) {
2152                            append_char('k', &makeflags_string_posix);
2153                            append_char((int) space_char, &makeflags_string_posix);
2154                            append_char((int) hyphen_char, &makeflags_string_posix);
2155                            append_char('S', &makeflags_string_posix);
2156                    }
2157            }
2158            if (do_not_exec_rule) {
2159                    append_char('n', &makeflags_string);
2160                    append_char('n', &makeflags_string_posix);
2161            }
2162            switch (report_dependencies_level) {
2163            case 4:
2164                    append_char('P', &makeflags_string);
2165                    append_char('P', &makeflags_string_posix);
2166            case 3:
2167                    append_char('P', &makeflags_string);
2168                    append_char('P', &makeflags_string_posix);
2169            case 2:
2170                    append_char('P', &makeflags_string);
2171                    append_char('P', &makeflags_string_posix);
2172            case 1:
2173                    append_char('P', &makeflags_string);
2174                    append_char('P', &makeflags_string_posix);
2175            }
2176            if (trace_status) {
2177                    append_char('p', &makeflags_string);
2178                    append_char('p', &makeflags_string_posix);
2179            }
2180            if (quest) {
2181                    append_char('q', &makeflags_string);
2182                    append_char('q', &makeflags_string_posix);
2183            }
2184            if (silent_all) {
```



```
2185                    append_char('s', &makeflags_string);
2186                    append_char('s', &makeflags_string_posix);
2187            }
2188            if (touch) {
2189                    append_char('t', &makeflags_string);
2190                    append_char('t', &makeflags_string_posix);
2191            }
2192            if (build_unconditional) {
2193                    append_char('u', &makeflags_string);
2194                    append_char('u', &makeflags_string_posix);
2195            }
2196            if (report_cwd) {
2197                    append_char('w', &makeflags_string);
2198                    append_char('w', &makeflags_string_posix);
2199            }
2210 #ifndef PARALLEL
2200            /* -c dmake_rcfile */
2201            if (dmake_rcfile_specified) {
2202                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2203                    dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
2204                    append_makeflags_string(dmake_rcfile, &makeflags_string);
2205                    append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
2206            }
2207            /* -g dmake_group */
2208            if (dmake_group_specified) {
2209                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2210                    dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
2211                    append_makeflags_string(dmake_group, &makeflags_string);
2212                    append_makeflags_string(dmake_group, &makeflags_string_posix);
2213            }
2214            /* -j dmake_max_jobs */
2215            if (dmake_max_jobs_specified) {
2216                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
2217                    dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
2218                    append_makeflags_string(dmake_max_jobs, &makeflags_string);
2219                    append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
2220            }
2221            /* -m dmake_mode */
2222            if (dmake_mode_specified) {
2223                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2224                    dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2225                    append_makeflags_string(dmake_mode, &makeflags_string);
2226                    append_makeflags_string(dmake_mode, &makeflags_string_posix);
2227            }
2228            /* -x dmake_compat_mode */
2229 //         if (dmake_compat_mode_specified) {
2230 //                 MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2231 //                 dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2232 //                 append_makeflags_string(dmake_compat_mode, &makeflags_string);
2233 //                 append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
2234 //         }
2235            /* -x dmake_output_mode */
2236            if (dmake_output_mode_specified) {
2237                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2238                    dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2239                    append_makeflags_string(dmake_output_mode, &makeflags_string);
2240                    append_makeflags_string(dmake_output_mode, &makeflags_string_pos
2241            }
2242            /* -o dmake_odir */
2243            if (dmake_odir_specified) {
2244                    MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2245                    dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2246                    append_makeflags_string(dmake_odir, &makeflags_string);
2247                    append_makeflags_string(dmake_odir, &makeflags_string_posix);
2248            }
2249            /* -M pmake_machinesfile */
```

```
2250         if (pmake_machinesfile_specified) {
2251                 MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2252                 pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2253                 append_makeflags_string(pmake_machinesfile, &makeflags_string);
2254                 append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2255         }
2256         /* -R */
2257         if (pmake_cap_r_specified) {
2258                 append_char((int) space_char, &makeflags_string);
2259                 append_char((int) hyphen_char, &makeflags_string);
2260                 append_char('R', &makeflags_string);
2261                 append_char((int) space_char, &makeflags_string_posix);
2262                 append_char((int) hyphen_char, &makeflags_string_posix);
2263                 append_char('R', &makeflags_string_posix);
2264         }
2276 #endif

2266 /*
2267  *      Make sure MAKEFLAGS is exported
2268  */
2269         maybe_append_prop(makeflags, macro_prop)->
2270           body.macro.exported = true;

2272         if (makeflags_string.buffer.start[1] != (int) nul_char) {
2273                 if (makeflags_string.buffer.start[1] != (int) space_char) {
2274                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2275                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2276                                       GETNAME(makeflags_string.buffer.start,
2277                                               FIND_LENGTH),
2278                                       false);
2279                 } else {
2280                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2281                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2282                                       GETNAME(makeflags_string.buffer.start + 2,
2283                                               FIND_LENGTH),
2284                                       false);
2285                 }
2286         }

2288 /*
2289  *      Add command line macro to POSIX makeflags_string
2290  */
2291         if (makeflags_and_macro.start) {
2292                 tmp_char = (char) space_char;
2293                 cp = makeflags_and_macro.start;
2294                 do {
2295                         append_char(tmp_char, &makeflags_string_posix);
2296                 } while ( tmp_char = *cp++ );
2297                 retmem_mb(makeflags_and_macro.start);
2298         }

2300 /*
2301  *      Now set the value of MAKEFLAGS macro in accordance
2302  *      with current mode.
2303  */
2304         macro = maybe_append_prop(makeflags, macro_prop);
2305         temp = (Boolean) macro->body.macro.read_only;
2306         macro->body.macro.read_only = false;
2307         if(posix || gnu_style) {
2308                 makeflags_string_current = &makeflags_string_posix;
2309         } else {
2310                 makeflags_string_current = &makeflags_string;
2311         }
2312         if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2313                 makeflags_value_saved =
2314                         GETNAME( makeflags_string_current->buffer.start + 1
```



```
2315                                 , FIND_LENGTH
2316                                 );
2317         } else {
2318                 if (makeflags_string_current->buffer.start[1] != (int) space_cha
2319                         makeflags_value_saved =
2320                                 GETNAME( makeflags_string_current->buffer.start
2321                                         , FIND_LENGTH
2322                                         );
2323                 } else {
2324                         makeflags_value_saved =
2325                                 GETNAME( makeflags_string_current->buffer.start
2326                                         , FIND_LENGTH
2327                                         );
2328                 }
2329         }
2330         (void) SETVAR( makeflags
2331                      , makeflags_value_saved
2332                      , false
2333                      );
2334         macro->body.macro.read_only = temp;

2336 /*
2337  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2338  */
2339         save_do_not_exec_rule = do_not_exec_rule;
2340         do_not_exec_rule = false;
2341         if (read_trace_level > 0) {
2342                 trace_reader = true;
2343         }

2345         for (i = 1; i < argc; i++) {
2346                 if (argv[i] &&
2347                     (argv[i][0] == (int) hyphen_char) &&
2348                     (argv[i][1] == 'f') &&
2349                     (argv[i][2] == (int) nul_char)) {
2350                         argv[i] = NULL;         /* Remove -f */
2351                         if (i >= argc - 1) {
2352                                 fatal(catgets(catd, 1, 190, "No filename argumen
2353                         }
2354                         MBSTOWCS(wcs_buffer, argv[++i]);
2355                         primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2356                         (void) read_makefile(primary_makefile, true, true, true)
2357                         argv[i] = NULL;         /* Remove filename */
2358                         makefile_read = true;
2359                 } else if (argv[i] &&
2360                            (argv[i][0] == (int) hyphen_char) &&
2361                            (argv[i][1] == 'c' ||
2362                             argv[i][1] == 'g' ||
2363                             argv[i][1] == 'j' ||
2364                             argv[i][1] == 'K' ||
2365                             argv[i][1] == 'M' ||
2366                             argv[i][1] == 'm' ||
2367                             argv[i][1] == 'O' ||
2368                             argv[i][1] == 'o') &&
2369                           (argv[i][2] == (int) nul_char)) {
2370                         argv[i] = NULL;
2371                         argv[++i] = NULL;
2372                 }
2373         }

2375 /*
2376  *      If no command line "-f" args then look for "makefile", and then for
2377  *      "Makefile" if "makefile" isn't found.
2378  */
2379         if (!makefile_read) {
2380                 (void) read_dir(dot,
```

```
2381                                (wchar_t *) NULL,
2382                                (Property) NULL,
2383                                (wchar_t *) NULL);
2384            if (!posix) {
2385                if (makefile_name->stat.is_file) {
2386                        if (Makefile->stat.is_file) {
2387                                warning(catgets(catd, 1, 310, "Both ‘makefile’ a
2388                        }
2389                        primary_makefile = makefile_name;
2390                        makefile_read = read_makefile(makefile_name,
2391                                                      false,
2392                                                      false,
2393                                                      true);
2394                }
2395                if (!makefile_read &&
2396                    Makefile->stat.is_file) {
2397                        primary_makefile = Makefile;
2398                        makefile_read = read_makefile(Makefile,
2399                                                      false,
2400                                                      false,
2401                                                      true);
2402                }
2403            } else {

2405                enum sccs_stat save_m_has_sccs = NO_SCCS;
2406                enum sccs_stat save_M_has_sccs = NO_SCCS;

2408                if (makefile_name->stat.is_file) {
2409                        if (Makefile->stat.is_file) {
2410                                warning(catgets(catd, 1, 191, "Both ‘makefile’ a
2411                        }
2412                }
2413                if (makefile_name->stat.is_file) {
2414                        if (makefile_name->stat.has_sccs == NO_SCCS) {
2415                           primary_makefile = makefile_name;
2416                           makefile_read = read_makefile(makefile_name,
2417                                                         false,
2418                                                         false,
2419                                                         true);
2420                        } else {
2421                          save_m_has_sccs = makefile_name->stat.has_sccs;
2422                          makefile_name->stat.has_sccs = NO_SCCS;
2423                          primary_makefile = makefile_name;
2424                          makefile_read = read_makefile(makefile_name,
2425                                                        false,
2426                                                        false,
2427                                                        true);
2428                        }
2429                }
2430                if (!makefile_read &&
2431                    Makefile->stat.is_file) {
2432                        if (Makefile->stat.has_sccs == NO_SCCS) {
2433                           primary_makefile = Makefile;
2434                           makefile_read = read_makefile(Makefile,
2435                                                         false,
2436                                                         false,
2437                                                         true);
2438                        } else {
2439                          save_M_has_sccs = Makefile->stat.has_sccs;
2440                          Makefile->stat.has_sccs = NO_SCCS;
2441                          primary_makefile = Makefile;
2442                          makefile_read = read_makefile(Makefile,
2443                                                        false,
2444                                                        false,
2445                                                        true);
2446                        }
```



```
2447                }
2448                if (!makefile_read &&
2449                        makefile_name->stat.is_file) {
2450                           makefile_name->stat.has_sccs = save_m_has_sccs;
2451                           primary_makefile = makefile_name;
2452                           makefile_read = read_makefile(makefile_name,
2453                                                         false,
2454                                                         false,
2455                                                         true);
2456                }
2457                if (!makefile_read &&
2458                    Makefile->stat.is_file) {
2459                           Makefile->stat.has_sccs = save_M_has_sccs;
2460                           primary_makefile = Makefile;
2461                           makefile_read = read_makefile(Makefile,
2462                                                         false,
2463                                                         false,
2464                                                         true);
2465                }
2466            }
2467        }
2468        do_not_exec_rule = save_do_not_exec_rule;
2469        allrules_read = makefile_read;
2470        trace_reader = false;

2472 /*
2473  *      Now get current value of MAKEFLAGS and compare it with
2474  *      the saved value we set before reading makefile.
2475  *      If they are different then MAKEFLAGS is subsequently set by
2476  *      makefile, just leave it there. Otherwise, if make mode
2477  *      is changed by using .POSIX target in makefile we need
2478  *      to correct MAKEFLAGS value.
2479  */
2480        Name mf_val = getvar(makeflags);
2481        if( (posix != is_xpg4)
2482         && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2483        {
2484                if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2485                        (void) SETVAR(makeflags,
2486                                      GETNAME(makeflags_string_posix.buffer.star
2487                                              FIND_LENGTH),
2488                                      false);
2489                } else {
2490                        if (makeflags_string_posix.buffer.start[1] != (int) spac
2491                                (void) SETVAR(makeflags,
2492                                              GETNAME(makeflags_string_posix.buf
2493                                                      FIND_LENGTH),
2494                                              false);
2495                        } else {
2496                                (void) SETVAR(makeflags,
2497                                              GETNAME(makeflags_string_posix.buf
2498                                                      FIND_LENGTH),
2499                                              false);
2500                        }
2501                }
2502        }

2504        if (makeflags_string.free_after_use) {
2505                retmem(makeflags_string.buffer.start);
2506        }
2507        if (makeflags_string_posix.free_after_use) {
2508                retmem(makeflags_string_posix.buffer.start);
2509        }
2510        makeflags_string.buffer.start = NULL;
2511        makeflags_string_posix.buffer.start = NULL;
```

```
2513         if (posix) {
2514                 /*
2515                  * If the user did not redefine the ARFLAGS macro in the
2516                  * default makefile (make.rules), then we'd like to
2517                  * change the macro value of ARFLAGS to be in accordance
2518                  * with "POSIX" requirements.
2519                  */
2520                 MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2521                 name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2522                 macro = get_prop(name->prop, macro_prop);
2523                 if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2524                     (IS_EQUAL(macro->body.macro.value->string_mb,
2525                               NOCATGETS("rv")))) {
2526                         MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2527                         value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2528                         (void) SETVAR(name,
2529                                       value,
2530                                       false);
2531                 }
2532         }

2534         if (!posix && !svr4) {
2535                 set_sgs_support();
2536         }


2539 /*
2540  *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2541  */
2542         macro = get_prop(keep_state_name->prop, macro_prop);
2543         if ((macro != NULL) &&
2544             macro->body.macro.exported) {
2545                 keep_state = true;
2546         }
2547         if (keep_state) {
2548                 if (macro == NULL) {
2549                         macro = maybe_append_prop(keep_state_name,
2550                                                   macro_prop);
2551                 }
2552                 macro->body.macro.exported = true;
2553                 (void) SETVAR(keep_state_name,
2554                               empty_name,
2555                               false);

2557                 /*
2558                  *      Read state file
2559                  */

2561                 /* Before we read state, let's make sure we have
2562                 ** right state file.
2563                 */
2564                 /* just in case macro references are used in make_state file
2565                 ** name, we better expand them at this stage using expand_value.
2566                 */
2567                 INIT_STRING_FROM_STACK(dest, destbuffer);
2568                 expand_value(make_state, &dest, false);

2570                 make_state = GETNAME(dest.buffer.start, FIND_LENGTH);

2572                 if(!stat(make_state->string_mb, &make_state_stat)) {
2573                    if(!(make_state_stat.st_mode & S_IFREG) ) {
2574                         /* copy the make_state structure to the other
2575                         ** and then let make_state point to the new
2576                         ** one.
2577                         */
2578                      memcpy(&state_filename, make_state,sizeof(state_filename))
```



```
2579                         state_filename.string_mb = state_file_str_mb;
2580                 /* Just a kludge to avoid two slashes back to back */
2581                         if((make_state->hash.length == 1)&&
2582                                 (make_state->string_mb[0] == '/')) {
2583                            make_state->hash.length = 0;
2584                            make_state->string_mb[0] = '\0';
2585                         }
2586                         sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2587                          make_state->string_mb,NOCATGETS("/.make.state"));
2588                         make_state = &state_filename;
2589                           /* adjust the length to reflect the appended string */
2590                         make_state->hash.length += 12;
2591                      }
2592                 } else { /* the file doesn't exist or no permission */
2593                    char tmp_path[MAXPATHLEN];
2594                    char *slashp;

2596                    if (slashp = strrchr(make_state->string_mb, '/')) {
2597                       strncpy(tmp_path, make_state->string_mb,
2598                                 (slashp - make_state->string_mb));
2599                         tmp_path[slashp - make_state->string_mb]=0;
2600                       if(strlen(tmp_path)) {
2601                         if(stat(tmp_path, &make_state_stat)) {
2602                           warning(catgets(catd, 1, 192, "directory %s for .KEEP_
2603                         }
2604                         if (access(tmp_path, F_OK) != 0) {
2605                           warning(catgets(catd, 1, 193, "can't access dir %s"),t
2606                         }
2607                       }
2608                    }
2609                 }
2610                 if (report_dependencies_level != 1) {
2611                         Makefile_type   makefile_type_temp = makefile_type;
2612                         makefile_type = reading_statefile;
2613                         if (read_trace_level > 1) {
2614                                 trace_reader = true;
2615                         }
2616                         (void) read_simple_file(make_state,
2617                                                 false,
2618                                                 false,
2619                                                 false,
2620                                                 false,
2621                                                 false,
2622                                                 true);
2623                         trace_reader = false;
2624                         makefile_type = makefile_type_temp;
2625                 }
2626         }
2627 }
```

*_____unchanged_portion_omitted_*