

new/usr/src/cmd/make/bin/Makefile

1

```
*****
1729 Wed May 20 11:55:51 2015
new/usr/src/cmd/make/bin/Makefile
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
12 # Copyright 2015, Richard Lowe.
14 PROG=    make
15 OBJS=    ar.o          \
16          depvar.o     \
17          dist.o       \
18          doname.o     \
19          dosys.o      \
20          files.o      \
21          globals.o   \
22          implicit.o  \
23          macro.o      \
24          main.o       \
25          misc.o       \
26          nse_printdep.o \
27          parallel.o  \
28          pmake.o     \
29          read.o       \
30          read2.o      \
31          rep.o        \
32          state.o
33 include .././Makefile.cmd
34 include .././Makefile.com
36 LDLIBS += ../lib/mksh/libmksh.a ../lib/mksdmsil8n/libmksdmsil8n.a ../lib/vroot/1
37 LDLIBS += ../lib/bsd/libbsd.a -lc -lnsl -lumem
39 CPPFLAGS += -D_FILE_OFFSET_BITS=64
41 ROOTLINKS = $(ROOTCCSBIN)/make $(ROOTXPG4BIN)/make $(ROOTBIN)/dmake $(ROOTCCSLIB
42             $(ROOTLIB)/svr4.make
44 ROOTRULES = $(ROOTSHLIB)/make/make.rules $(ROOTSHLIB)/make/svr4.make.rules
46 all:      $(PROG)
48 install: all $(ROOTPROG) $(ROOTLINKS) $(ROOTRULES)
50 $(PROG):  $(OBJS)
51           $(LINK.cc) $(OBJS) -o $@ $(LDLIBS)
52           $(POST_PROCESS)
54 $(ROOTCCSBIN)/make:
55     -$(RM) $@; $(SYMLINK) .././bin/make $@
57 $(ROOTCCSLIB)/svr4.make:
58     -$(RM) $@; $(SYMLINK) .././bin/make $@
60 $(ROOTLIB)/svr4.make:
```

new/usr/src/cmd/make/bin/Makefile

2

```
61     -$(RM) $@; $(SYMLINK) ../bin/make $@
63 $(ROOTXPG4BIN)/make:
64     -$(RM) $@; $(SYMLINK) .././bin/make $@
66 $(ROOTBIN)/dmake:
67     -$(RM) $@; $(SYMLINK) ./make $@
69 $(ROOTRULES) := FILEMODE = 0444
71 $(ROOTRULES): $(ROOTSHLIB)/make
73 $(ROOTSHLIB)/make: FRC
74     $(INS.dir)
76 $(ROOTSHLIB)/make/%: %.file
77     $(INS.rename)
79 lint:
81 clean:
82     $(RM) $(OBJS)
84 FRC:
86 include .././Makefile.targ
```

```

*****
95642 Wed May 20 11:55:52 2015
new/usr/src/cmd/make/bin/doname.cc
make: undef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  * doname.c
28  *
29  * Figure out which targets are out of date and rebuild them
30  */

32 /*
33  * Included files
34  */
35 #include <alloca.h>          /* alloca() */

37 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
38 # include <avo/strings.h> /* AVO_STRDUP() */
39 # include <dm/Avo_MToolJobResultMsg.h>
40 # include <dm/Avo_MToolJobStartMsg.h>
41 # include <dm/Avo_MToolRsrcInfoMsg.h>
42 # include <dm/Avo_macro_defs.h> /* AVO_BLOCK_INTERRUPTS & AVO_UNBLOCK_INTER
43 # include <dmthread/Avo_ServerState.h>
44 # include <rw/pstream.h>
45 # include <rw/xdrstrea.h>
46 #endif

38 #include <fcntl.h>
39 #include <mk/defs.h>
40 #include <mksh/118n.h>          /* get_char_semantics_value() */
41 #include <mksh/macro.h>        /* getvar(), expand_value() */
42 #include <mksh/misc.h>        /* getmem() */
43 #include <poll.h>

46 #include <signal.h>

48 # include <stropts.h>

50 #include <sys/errno.h>
51 #include <sys/stat.h>

```

```

52 #include <sys/types.h>
53 #include <sys/utsname.h>      /* uname() */
54 #include <sys/wait.h>
55 #include <unistd.h>          /* close() */

57 /*
58  * Defined macros
59  */
60 # define LOCALHOST "localhost"

62 #define MAXRULES 100

74 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
75 #define SEND_MTOOL_MSG(cmds) \
76     if (send_mtool_msgs) { \
77         cmds \
78     }
79 #else
64 #define SEND_MTOOL_MSG(cmds)
81 #endif

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

69 /*
70  * typedefs & structs
71  */

73 /*
74  * Static variables
75  */
76 static char hostName[MAXNAMELEN] = "";
77 static char userName[MAXNAMELEN] = "";

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

80 static int second_pass = 0;

82 /*
83  * File table of contents
84  */
85 extern Doname doname_check(register Name target, register Boolean do_g
86 extern Doname doname(register Name target, register Boolean do_get, re
87 static Boolean check_dependencies(Doname *result, Property line, Boolea
88 void dynamic_dependencies(Name target);
89 static Doname run_command(register Property line, Boolean print_machin
90 extern Doname execute_serial(Property line);
91 extern Name vpath_translation(register Name cmd);
92 extern void check_state(Name temp_file_name);
93 static void read_dependency_file(register Name filename);
94 static void check_read_state_file(void);
95 static void do_assign(register Name line, register Name target);
96 static void build_command_strings(Name target, register Property lin
97 static Doname touch_command(register Property line, register Name targ
98 extern void update_target(Property line, Doname result);
99 static Doname sccs_get(register Name target, register Property *comman
100 extern void read_directory_of_file(register Name file);
101 static void add_pattern_conditionals(register Name target);
102 extern void set_locals(register Name target, register Property old_l
103 extern void reset_locals(register Name target, register Property old
104 extern Boolean check_auto_dependencies(Name target, int auto_count, Nam
105 static void delete_query_chain(Chain ch);

```

```

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

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

112 /*
113  * DONE.
114  *
115  * doname_check(target, do_get, implicit, automatic)
116  *
117  * Will call doname() and then inspect the return value
118  *
119  * Return value:
120  *                Indication if the build failed or not
121  *
122  * Parameters:
123  *     target      The target to build
124  *     do_get      Passed thru to doname()
125  *     implicit    Passed thru to doname()
126  *     automatic   Are we building a hidden dependency?
127  *
128  * Global variables used:
129  *     build_failed_seen    Set if -k is on and error occurs
130  *     continue_after_error Indicates that -k is on
131  *     report_dependencies  No error msg if -P is on
132  */
133 Doname
134 doname_check(register Name target, register Boolean do_get, register Boolean imp
135 {
136     int first_time = 1;
137     (void) fflush(stdout);
138 try_again:
139     switch (doname(target, do_get, implicit, automatic)) {
140     case build_ok:
141         second_pass = 0;
142         return build_ok;
143     case build_running:
144         second_pass = 0;
145         return build_running;
146     case build_failed:
147         if (!continue_after_error) {
148             fatal(catgets(catd, 1, 13, "Target '%s' not remade becau
149                 target->string_mb);
150         }
151         build_failed_seen = true;
152         second_pass = 0;
153         return build_failed;
154     case build_dont_know:
155         /*
156          * If we can't figure out how to build an automatic
157          * (hidden) dependency, we just ignore it.
158          * We later declare the target to be out of date just in
159          * case something changed.
160          * Also, don't complain if just reporting the dependencies
161          * and not building anything.
162          */
163         if (automatic || (report_dependencies_level > 0)) {

```

```

164         second_pass = 0;
165         return build_dont_know;
166     }
167     if (first_time) {
168         first_time = 0;
169         second_pass = 1;
170         goto try_again;
171     }
172     second_pass = 0;
173     if (continue_after_error && !svr4) {
174         warning(catgets(catd, 1, 14, "Don't know how to make tar
175             target->string_mb);
176         build_failed_seen = true;
177         return build_failed;
178     }
179     fatal(catgets(catd, 1, 15, "Don't know how to make target '%s'")
180         break;
181     }
182 #ifdef lint
183     return build_failed;
184 #endif
185 }
    unchanged_portion_omitted

917 /*
918  * DONE.
919  *
920  * check_dependencies(result, line, do_get,
921  *     target, true_target, doing_subtree, out_of_date_tail,
922  *     old_locals, implicit, command, less, rechecking_target)
923  *
924  * Return value:
925  *                True returned if some dependencies left running
926  *
927  * Parameters:
928  *     result      Pointer to cell we update if build failed
929  *     line        We get the dependencies from here
930  *     do_get      Allow use of sccs get in recursive doname()
931  *     target      The target to chase dependencies for
932  *     true_target The real one for :: and lib(member)
933  *     doing_subtree True if building a conditional macro subtree
934  *     out_of_date_tail Used to set the $? list
935  *     old_locals    Used for resetting the local macros
936  *     implicit      Called when scanning for implicit rules?
937  *     command      Place to stuff command
938  *     less         Set to $< value
939  *
940  * Global variables used:
941  *     command_changed Set if we suspect .make.state needs rewrite
942  *     debug_level     Should we trace actions?
943  *     force           The Name " FORCE", compared against
944  *     recursion_level Used for tracing
945  *     rewrite_statefile Set if .make.state needs rewriting
946  *     wait_name       The Name ".WAIT", compared against
947  */
948 static Boolean
949 #ifdef TEAMWARE_MAKE_CMN
950 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
951 #else
952 check_dependencies(Doname *result, Property line, Boolean do_get, Name target, N
953 #endif
954 {
955     Boolean      dependencies_running;
956     register Dependency      dependency;
957     Doname      dep_result;
958     Boolean      dependency_changed = false;

```

```

960     line->body.line.dependency_time = file_doesnt_exist;
961     if (line->body.line.query != NULL) {
962         delete_query_chain(line->body.line.query);
963     }
964     line->body.line.query = NULL;
965     line->body.line.is_out_of_date = false;
966     dependencies_running = false;
967     /*
968     * Run thru all the dependencies and call doname() recursively
969     * on each of them.
970     */
971     for (dependency = line->body.line.dependencies;
972          dependency != NULL;
973          dependency = dependency->next) {
974         Boolean this_dependency_changed = false;

976         if (!dependency->automatic &&
977             (rechecking_target || target->rechecking_target)) {
978             /*
979             * We only bother with the autos when rechecking
980             */
981             continue;
982         }

984         if (dependency->name == wait_name) {
985             /*
986             * The special target .WAIT means finish all of
987             * the prior dependencies before continuing.
988             */
989             if (dependencies_running) {
990                 break;
991             }
1022 #ifndef DISTRIBUTED
1023         } else if ((!parallel_ok(dependency->name, false)) &&
1024                  (dependencies_running)) {
1025             /*
1026             * If we can't execute the current dependency in
1027             * parallel, hold off the dependency processing
1028             * to preserve the order of the dependencies.
1029             */
1030             break;
1031 #endif
1032         } else {
1033             timestruc_t     depe_time = file_doesnt_exist;

996             if (true_target->is_member) {
997                 depe_time = exists(dependency->name);
998             }
999             if (dependency->built ||
1000                 (dependency->name->state == build_failed)) {
1001                 dep_result = (Doname) dependency->name->state;
1002             } else {
1003                 dep_result = doname_check(dependency->name,
1004                                          do_get,
1005                                          false,
1006                                          (Boolean) dependency->
1007             }
1008             if (true_target->is_member || dependency->name->is_membe
1009                 /* should compare only secs, cause lib members d
1010                 if (depe_time.tv_sec != dependency->name->stat.t
1011                     this_dependency_changed =
1012                     dependency_changed =
1013                     true;
1014         }

```

```

1015     } else {
1016         if (depe_time != dependency->name->stat.time) {
1017             this_dependency_changed =
1018             dependency_changed =
1019             true;
1020         }
1021     }
1022     dependency->built = true;
1023     switch (dep_result) {
1024     case build_running:
1025         dependencies_running = true;
1026         continue;
1027     case build_failed:
1028         *result = build_failed;
1029         break;
1030     case build_dont_know:
1031     /*
1032     * If make can't figure out how to make a dependency, maybe the dependency
1033     * is out of date. In this case, we just declare the target out of date
1034     * and go on. If we really need the dependency, the make'ing of the target
1035     * will fail. This will only happen for automatic (hidden) dependencies.
1036     */
1037         if (!recheck_conditionals) {
1038             line->body.line.is_out_of_date = true;
1039         }
1040         /*
1041         * Make sure the dependency is not saved
1042         * in the state file.
1043         */
1044         dependency->stale = true;
1045         rewrite_statefile =
1046         command_changed =
1047         true;
1048         if (debug_level > 0) {
1049             (void) printf(catgets(catd, 1, 19, "Targ
1050                             true_target->string_mb,
1051                             dependency->name->string_mb
1052             }
1053             break;
1054         }
1055         if (dependency->name->depends_on_conditional) {
1056             target->depends_on_conditional = true;
1057         }
1058         if (dependency->name == force) {
1059             target->stat.time =
1060             dependency->name->stat.time;
1061         }
1062         /*
1063         * Propagate new timestamp from "member" to
1064         * "lib.a(member)".
1065         */
1066         (void) exists(dependency->name);

1068         /* Collect the timestamp of the youngest dependency */
1069         line->body.line.dependency_time =
1070         MAX(dependency->name->stat.time,
1071            line->body.line.dependency_time);

1073         /* Correction: do not consider nanosecs for members */
1074         if (true_target->is_member || dependency->name->is_member
1075             line->body.line.dependency_time.tv_nsec = 0;
1076         }

1078         if (debug_level > 1) {
1079             (void) printf(catgets(catd, 1, 20, "%sDate(%s)=
1080                             recursion_level,

```

```

1081         "",
1082         dependency->name->string_mb,
1083         time_to_string(dependency->name->
1084             stat.time));
1085     if (dependency->name->stat.time > line->body.lin
1086         (void) printf(catgets(catd, 1, 21, "%*sD
1087             recursion_level,
1088             "",
1089             true_target->string_mb,
1090             time_to_string(line->body.
1091                 dependency_
1092         )
1093     }
1094 }
1095
1096 /* Build the $? list */
1097 if (true_target->is_member) {
1098     if (this_dependency_changed == true) {
1099         true_target->stat.time = dependency->nam
1100         true_target->stat.time.tv_sec--;
1101     } else {
1102         /* Dina:
1103          * The next statement is commented
1104          * out as a fix for bug #1051032.
1105          * if dependency hasn't changed
1106          * then there's no need to invalidate
1107          * true_target. This statemnt causes
1108          * make to take much longer to process
1109          * an already-built archive. Soren
1110          * said it was a quick fix for some
1111          * problem he doesn't remember.
1112          */
1113         true_target->stat.time = file_no_time;
1114         (void) exists(true_target);
1115     }
1116 } else {
1117     (void) exists(true_target);
1118 }
1119 Boolean out_of_date;
1120 if (true_target->is_member || dependency->name->is_membe
1121     out_of_date = (Boolean) OUT_OF_DATE_SEC(true_tar
1122     dependen
1123 } else {
1124     out_of_date = (Boolean) OUT_OF_DATE(true_target-
1125     dependency->
1126 }
1127 if ((build_unconditional || out_of_date) &&
1128     (dependency->name != force) &&
1129     (dependency->stale == false)) {
1130     *out_of_date_tail = ALLOC(Chain);
1131     if (dependency->name->is_member &&
1132         (get_prop(dependency->name->prop,
1133             member_prop) != NULL)) {
1134         (*out_of_date_tail)->name =
1135             get_prop(dependency->name->prop,
1136                 member_prop)->
1137                 body.member.member;
1138     } else {
1139         (*out_of_date_tail)->name =
1140             dependency->name;
1141     }
1142     (*out_of_date_tail)->next = NULL;
1143     out_of_date_tail = &(*out_of_date_tail)->next;
1144     if (debug_level > 0) {
1145         if (dependency->name->stat.time == file_
1146             (void) printf(catgets(catd, 1, 2

```

```

1147         "",
1148         true_target->strin
1149         dependency->name->
1150     } else {
1151         (void) printf(catgets(catd, 1, 2
1152             recursion_level,
1153             "",
1154             true_target->strin
1155         dependency->name->
1156     }
1157 }
1158 }
1159 if (dependency->name == force) {
1160     force->stat.time =
1161         file_max_time;
1162     force->state = build_dont_know;
1163 }
1164 }
1165 }
1166 #ifdef TEAMWARE_MAKE_CMN
1167 if (dependencies_running) {
1168     if (doing_subtree) {
1169         if (target->conditional_cnt > 0) {
1170             reset_locals(target,
1171                 old_locals,
1172                 get_prop(target->prop,
1173                     conditional_prop),
1174                 0);
1175         }
1176         return true;
1177     } else {
1178         target->state = build_running;
1179         add_pending(target,
1180             --recursion_level,
1181             do_get,
1182             implicit,
1183             false);
1184         if (target->conditional_cnt > 0) {
1185             reset_locals(target,
1186                 old_locals,
1187                 get_prop(target->prop,
1188                     conditional_prop),
1189                 0);
1190         }
1191         return true;
1192     }
1193 }
1194 #endif
1195 /*
1196  * Collect the timestamp of the youngest double colon target
1197  * dependency.
1198  */
1199 if (target->is_double_colon_parent) {
1200     for (dependency = line->body.line.dependencies;
1201         dependency != NULL;
1202         dependency = dependency->next) {
1203         Property tmp_line;
1204
1205         if ((tmp_line = get_prop(dependency->name->prop, line_pr
1206             if (tmp_line->body.line.dependency_time != file_m
1207                 target->stat.time =
1208                     MAX(tmp_line->body.line.dependency_tim
1209                         target->stat.time);
1210         }
1211     }
1212 }

```

```

1213     }
1214     if ((true_target->is_member) && (dependency_changed == true)) {
1215         true_target->stat.time = file_no_time;
1216     }
1217     /*
1218     * After scanning all the dependencies, we check the rule
1219     * if we found one.
1220     */
1221     if (line->body.line.command_template != NULL) {
1222         if (line->body.line.command_template_redefined) {
1223             warning(catgets(catd, 1, 24, "Too many rules defined for
1224             target->string_mb);
1225         }
1226         *command = line;
1227         /* Check if the target is out of date */
1228         Boolean out_of_date;
1229         if (true_target->is_member) {
1230             out_of_date = (Boolean) OUT_OF_DATE_SEC(true_target->sta
1231             line->body.line.
1232         } else {
1233             out_of_date = (Boolean) OUT_OF_DATE(true_target->stat.ti
1234             line->body.line.depe
1235         }
1236         if (build_unconditional || out_of_date){
1237             if(!recheck_conditionals) {
1238                 line->body.line.is_out_of_date = true;
1239             }
1240         }
1241         line->body.line.sccs_command = false;
1242         line->body.line.target = true_target;
1243         if(gnu_style) {
1244
1245             // set $< for explicit rule
1246             if(line->body.line.dependencies != NULL) {
1247                 less = line->body.line.dependencies->name;
1248             }
1249
1250             // set $* for explicit rule
1251             Name target_body;
1252             Name tt = true_target;
1253             Property member;
1254             register wchar_t *target_end;
1255             register Dependency suffix;
1256             register int suffix_length;
1257             Wstring targ_string;
1258             Wstring suf_string;
1259
1260             if (true_target->is_member &&
1261                 ((member = get_prop(target->prop, member_prop)) !=
1262                 NULL)) {
1263                 tt = member->body.member.member;
1264             }
1265             targ_string.init(tt);
1266             target_end = targ_string.get_string() + tt->hash.length;
1267             for (suffix = suffixes; suffix != NULL; suffix = suffix-
1268                 suffix_length = suffix->name->hash.length;
1269                 suf_string.init(suffix->name);
1270                 if (tt->hash.length < suffix_length) {
1271                     continue;
1272                 } else if (!IS_WEQUALN(suf_string.get_string(),
1273                     (target_end - suffix_length),
1274                     suffix_length)) {
1275                     continue;
1276                 }
1277             target_body = GETNAME(
1278                 targ_string.get_string(),

```

```

1279             (int)(tt->hash.length - suffix_length)
1280             );
1281             line->body.line.star = target_body;
1282         }
1283
1284         // set result = build_ok so that implicit rules are not
1285         if(*result == build_dont_know) {
1286             *result = build_ok;
1287         }
1288     }
1289     if (less != NULL) {
1290         line->body.line.less = less;
1291     }
1292 }
1293
1294     return false;
1295 }
1296
1297 unchanged_portion_omitted
1298
1299 /*
1300 * execute_serial(line)
1301 *
1302 * Runs thru the command line for the target and
1303 * executes the rules one by one.
1304 *
1305 * Return value:
1306 *
1307 * The result of the command build
1308 *
1309 * Parameters:
1310 * line The command to execute
1311 *
1312 * Static variables used:
1313 *
1314 * Global variables used:
1315 * continue_after_error -k flag
1316 * do_not_exec_rule -n flag
1317 * report_dependencies -P flag
1318 * silent Don't echo commands before executing
1319 * temp_file_name Temp file for auto dependencies
1320 * vpath_defined If true, translate path for command
1321 */
1322 Doname
1323 execute_serial(Property line)
1324 {
1325     int child_pid = 0;
1326 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
1327     Avo_MToolJobResultMsg *job_result_msg;
1328     RWCCollectable *xdr_msg;
1329 #endif
1330     Boolean printed_serial;
1331     Doname result = build_ok;
1332     Cmd_line rule, cmd_tail, command = NULL;
1333     char mbstring[MAXPATHLEN];
1334     int filed;
1335     Name target = line->body.line.target;
1336
1337     SEND_MTOOL_MSG(
1338         if (!sent_rsrc_info_msg) {
1339             if (userName[0] == '\0') {
1340                 avo_get_user(userName, NULL);
1341             }
1342             if (hostName[0] == '\0') {
1343                 strcpy(hostName, avo_hostname());
1344             }
1345             send_rsrc_info_msg(1, hostName, userName);
1346             sent_rsrc_info_msg = 1;

```

```

1824     }
1825     send_job_start_msg(line);
1826     job_result_msg = new Avo_MToolJobResultMsg();
1827 );

1829 target->has_recursive_dependency = false;
1830 // We have to create a copy of the rules chain for processing because
1831 // the original one can be destroyed during .make.state file rereading.
1832 for (rule = line->body.line.command_used;
1833      rule != NULL;
1834      rule = rule->next) {
1835     if (command == NULL) {
1836         command = cmd_tail = ALLOC(Cmd_line);
1837     } else {
1838         cmd_tail->next = ALLOC(Cmd_line);
1839         cmd_tail = cmd_tail->next;
1840     }
1841     *cmd_tail = *rule;
1842 }
1843 if (command) {
1844     cmd_tail->next = NULL;
1845 }
1846 for (rule = command; rule != NULL; rule = rule->next) {
1847     if (posix && (touch || quest) && !rule->always_exec) {
1848         continue;
1849     }
1850     if (vpath_defined) {
1851         rule->command_line =
1852             vpath_translation(rule->command_line);
1853     }
1854     /* Echo command line, maybe. */
1855     if ((rule->command_line->hash.length > 0) &&
1856         !silent &&
1857         (!rule->silent || do_not_exec_rule) &&
1858         (report_dependencies_level == 0)) {
1859         (void) printf("%s\n", rule->command_line->string_mb);
1860         SEND_MTOOL_MSG(
1861             job_result_msg->appendOutput(AVO_STRDUP(rule->co
1862             ));
1863     }
1864     if (rule->command_line->hash.length > 0) {
1865         SEND_MTOOL_MSG(
1866             (void) sprintf(mbstring,
1867                 NOCATGETS("%s/make.stdout.%d.%d.
1868                 tmpdir,
1869                 getpid(),
1870                 file_number++);

1872         int tmp_fd = mkstemp(mbstring);
1873         if(tmp_fd) {
1874             (void) close(tmp_fd);
1875         }

1877         stdout_file = strdup(mbstring);
1878         stderr_file = NULL;
1879         child_pid = pollResults(stdout_file,
1880                               (char *)NULL,
1881                               (char *)NULL);
1882     });
1883     /* Do assignment if command line prefixed with "=" */
1884     if (rule->assign) {
1885         result = build_ok;
1886         do_assign(rule->command_line, target);
1887     } else if (report_dependencies_level == 0) {
1888         /* Execute command line. */
1889     }
1933 #ifndef DISTRIBUTED

```

```

1934         setvar_envvar((Avo_DoJobMsg *)NULL);
1935 #else
1936         setvar_envvar();
1937 #endif
1938
1939 result = dosys(rule->command_line,
1940               (Boolean) rule->ignore_error,
1941               (Boolean) rule->make_refd,
1942               /* ds 98.04.23 bug #4085164. make
1943               false,
1944               /* BOOLEAN(rule->silent &&
1945               rule->ignore_error), */
1946               (Boolean) rule->always_exec,
1947               target,
1948               send_mtool_msgs);
1949
1950 check_state(temp_file_name);
1951 }
1952 SEND_MTOOL_MSG(
1953     append_job_result_msg(job_result_msg);
1954     if (child_pid > 0) {
1955         kill(child_pid, SIGUSR1);
1956         while (!(waitpid(child_pid, 0, 0) == -1
1957             && (errno == ECHILD)));
1958     }
1959     child_pid = 0;
1960     (void) unlink(stdout_file);
1961     retmem_mb(stdout_file);
1962     stdout_file = NULL;
1963 );
1964 } else {
1965     result = build_ok;
1966 }
1967 if (result == build_failed) {
1968     if (silent || rule->silent) {
1969         (void) printf(catgets(catd, 1, 242, "The followi
1970             rule->command_line->string_mb);
1971         SEND_MTOOL_MSG(
1972             job_result_msg->appendOutput(AVO_STRDUP(
1973             job_result_msg->appendOutput(AVO_STRDUP(
1974             ));
1975     }
1976     if (!rule->ignore_error && !ignore_errors) {
1977         if (!continue_after_error) {
1978             SEND_MTOOL_MSG(
1979                 job_result_msg->setResult(job_ms
1980                 xdr_msg = (RWCollectable*)
1981                 job_result_msg;
1982                 xdr(&xdrs, xdr_msg);
1983                 (void) fflush(mtool_msgs_fp);
1984                 delete job_result_msg;
1985             );
1986             fatal(catgets(catd, 1, 244, "Command fai
1987                 target->string_mb);
1988         }
1989         /*
1990         * Make sure a failing command is not
1991         * saved in .make.state.
1992         */
1993         line->body.line.command_used = NULL;
1994         break;
1995     } else {
1996         result = build_ok;
1997     }
1998 }
1999 }
2000 for (rule = command; rule != NULL; rule = cmd_tail) {
2001     cmd_tail = rule->next;

```

```

1952         free(rule);
1953     }
1954     command = NULL;
1955     SEND_MTOOL_MSG(
1956         job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
1957         xdr_msg = (RWCollectable*) job_result_msg;
1958         xdr(&xdrs, xdr_msg);
1959         (void) fflush(mtool_msgs_fp);

1961         delete job_result_msg;
1962     );
1963     if (temp_file_name != NULL) {
1964         free_name(temp_file_name);
1965     }
1966     temp_file_name = NULL;

1968     Property spro = get_prop(sunpro_dependencies->prop, macro_prop);
1969     if (spro != NULL) {
1970         Name val = spro->body.macro.value;
1971         if (val != NULL) {
1972             free_name(val);
1973             spro->body.macro.value = NULL;
1974         }
1975     }
1976     spro = get_prop(sunpro_dependencies->prop, env_mem_prop);
1977     if (spro) {
1978         char *val = spro->body.env_mem.value;
1979         if (val != NULL) {
1980             /*
1981              * Do not return memory allocated for SUNPRO_DEPENDENCIE
1982              * It will be returned in setvar_daemon() in macro.cc
1983              */
1984             //      retmem_mb(val);
1985             spro->body.env_mem.value = NULL;
1986         }
1987     }
1988     return result;
1989 }
1990 }

```

```
2041 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
```

```

2043 /*
2044  * Create and send an Avo_MToolRsrcInfoMsg.
2045  */
2046 void
2047 send_rsrc_info_msg(int max_jobs, char *hostname, char *username)
2048 {
2049     static int          first = 1;
2050     Avo_MToolRsrcInfoMsg *msg;
2051     RWListCollectables server_list;
2052     Avo_ServerState    *server_state;
2053     RWCollectable      *xdr_msg;

2055     if (!first) {
2056         return;
2057     }
2058     first = 0;

2060     create_xdrs_ptr();

2062     server_state = new Avo_ServerState(max_jobs, hostname, username);
2063     server_list.append(server_state);
2064     msg = new Avo_MToolRsrcInfoMsg(&server_list);

```

```

2066     xdr_msg = (RWCollectable *)msg;
2067     xdr(get_xdrs_ptr(), xdr_msg);
2068     (void) fflush(get_mtool_msgs_fp());

2070     delete server_state;
2071     delete msg;
2072 }

2074 /*
2075  * Create and send an Avo_MToolJobStartMsg.
2076  */
2077 void
2078 send_job_start_msg(Property line)
2079 {
2080     int          cmd_options = 0;
2081     Avo_MToolJobStartMsg *msg;
2082     Cmd_line     rule;
2083     Name         target = line->body.line.target;
2084     RWCollectable *xdr_msg;

2086     if (userName[0] == '\0') {
2087         avo_get_user(userName, NULL);
2088     }
2089     if (hostName[0] == '\0') {
2090         strcpy(hostName, avo_hostname());
2091     }

2093     msg = new Avo_MToolJobStartMsg();
2094     msg->setJobId(++job_msg_id);
2095     msg->setTarget(AVO_STRDUP(target->string_mb));
2096     msg->setHost(AVO_STRDUP(hostName));
2097     msg->setUser(AVO_STRDUP(userName));

2099     for (rule = line->body.line.command_used;
2100          rule != NULL;
2101          rule = rule->next) {
2102         if (posix && (touch || quest) && !rule->always_exec) {
2103             continue;
2104         }
2105         if (vpath_defined) {
2106             rule->command_line =
2107                 vpath_translation(rule->command_line);
2108         }
2109         cmd_options = 0;
2110         if (rule->ignore_error || ignore_errors) {
2111             cmd_options |= ignore_mask;
2112         }
2113         if (rule->silent || silent) {
2114             cmd_options |= silent_mask;
2115         }
2116         if (rule->command_line->meta) {
2117             cmd_options |= meta_mask;
2118         }
2119         if (!touch && (rule->command_line->hash.length > 0)) {
2120             msg->appendCmd(new Avo_DmakeCommand(rule->command_line->
2121             }
2122         }

2124     xdr_msg = (RWCollectable*) msg;
2125     xdr(&xdrs, xdr_msg);
2126     (void) fflush(mtool_msgs_fp);

2128     /* tolik, 08/39/2002.
2129      * I commented out this code because it causes using unallocated memory.
2130      * delete msg;
2131     */

```



```

2132 }
2133 /*
2134 * Append the stdout/err to Avo_MToolJobResultMsg.
2135 */
2136 static void
2137 append_job_result_msg(Avo_MToolJobResultMsg *job_result_msg)
2138 {
2139     FILE *fp;
2140     char line[MAXPATHLEN];
2141     char stdout_file2[MAXPATHLEN];
2142
2143     if (stdout_file != NULL) {
2144         fp = fopen(stdout_file, "r");
2145         if (fp == NULL) {
2146             /* Hmmmm... what should we do here? */
2147             warning(catgets(catd, 1, 326, "fopen() of stdout_file fa
2148             return;
2149         }
2150         while (fgets(line, MAXPATHLEN, fp) != NULL) {
2151             if (line[strlen(line) - 1] == '\n') {
2152                 line[strlen(line) - 1] = '\0';
2153             }
2154             job_result_msg->appendOutput(AVO_STRDUP(line));
2155         }
2156         (void) fclose(fp);
2157         us_sleep(STAT_RETRY_SLEEP_TIME);
2158     } else {
2159         /* Hmmmm... stdout_file shouldn't be NULL */
2160         warning(catgets(catd, 1, 327, "Internal stdout_file variable sho
2161     }
2162 }
2163 }
2164 #endif /* TEAMWARE_MAKE_CMN */

```

```

1994 /*
1995 * vpath_translation(cmd)
1996 *
1997 * Translates one command line by
1998 * checking each word. If the word has an alias it is translated.
1999 *
2000 * Return value:
2001 *     The translated command
2002 *
2003 * Parameters:
2004 *     cmd      Command to translate
2005 *
2006 * Global variables used:
2007 */
2008 Name
2009 vpath_translation(register Name cmd)
2010 {
2011     wchar_t buffer[STRING_BUFFER_LENGTH];
2012     String_rec new_cmd;
2013     wchar_t *p;
2014     wchar_t *start;
2015
2016     if (!vpath_defined || (cmd == NULL) || (cmd->hash.length == 0)) {
2017         return cmd;
2018     }
2019     INIT_STRING_FROM_STACK(new_cmd, buffer);
2020
2021     Wstring wcb(cmd);
2022     p = wcb.get_string();
2023
2024     while (*p != (int) nul_char) {
2025         while (iswspace(*p) && (*p != (int) nul_char)) {

```

```

2026         append_char(*p++, &new_cmd);
2027     }
2028     start = p;
2029     while (!iswspace(*p) && (*p != (int) nul_char)) {
2030         p++;
2031     }
2032     cmd = GETNAME(start, p - start);
2033     if (cmd->has_vpath_alias_prop) {
2034         cmd = get_prop(cmd->prop, vpath_alias_prop)->
2035             body.vpath_alias.alias;
2036     }
2037     APPEND_NAME(cmd,
2038                 &new_cmd,
2039                 (int) cmd->hash.length);
2040     } else {
2041         append_string(start, &new_cmd, p - start);
2042     }
2043     cmd = GETNAME(new_cmd.buffer.start, FIND_LENGTH);
2044     if (new_cmd.free_after_use) {
2045         retmem(new_cmd.buffer.start);
2046     }
2047     return cmd;
2048 }
2049
21000 unchanged_portion_omitted
21001
21002 2596 /*
21003 2597 * touch_command(line, target, result)
21004 2598 *
21005 2599 * If this is an "make -t" run we do this.
21006 2600 * We touch all targets in the target group ("foo + fie:") if any.
21007 2601 *
21008 2602 * Return value:
21009 2603 *     Indicates if the command failed or not
21010 2604 *
21011 2605 * Parameters:
21012 2606 *     line      The command line to update
21013 2607 *     target    The target we are touching
21014 2608 *     result    Initial value for the result we return
21015 2609 *
21016 2610 * Global variables used:
21017 2611 *     do_not_exec_rule Indicates that -n is on
21018 2612 *     silent         Do not echo commands
21019 2613 */
21020 2614 static Doname
21021 2615 touch_command(register Property line, register Name target, Doname result)
21022 2616 {
21023 2617     Name name;
21024 2618     register Chain target_group;
21025 2619     String_rec touch_string;
21026 2620     wchar_t buffer[MAXPATHLEN];
21027 2621     Name touch_cmd;
21028 2622     Cmd_line rule;
21029
21030 2796 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
21031 2797     Avo_MToolJobResultMsg *job_result_msg;
21032 2798     RWCollectable *xdr_msg;
21033 2799     int child_pid = 0;
21034 2800     wchar_t string[MAXPATHLEN];
21035 2801     char mbstring[MAXPATHLEN];
21036 2802     int filed;
21037 2803 #endif
21038
21039 2625     SEND_MTOOL_MSG(
21040 2626         if (!sent_rsrc_info_msg) {
21041 2627             if (userName[0] == '\0') {
21042 2628                 avo_get_user(userName, NULL);

```

```

2629     }
2630     if (hostName[0] == '\0') {
2631         strcpy(hostName, avo_hostname());
2632     }
2633     send_rsrc_info_msg(1, hostName, userName);
2634     sent_rsrc_info_msg = 1;
2635 }
2636 send_job_start_msg(line);
2637 job_result_msg = new Avo_MToolJobResultMsg();
2638 );
2639 for (name = target, target_group = NULL; name != NULL;) {
2640     if (!name->is_member) {
2641         /*
2642          * Build a touch command that can be passed
2643          * to dosys(). If KEEP_STATE is on, "make -t"
2644          * will save the proper command, not the
2645          * "touch" in .make.state.
2646          */
2647         INIT_STRING_FROM_STACK(touch_string, buffer);
2648         MBSTOWCS(wcs_buffer, NOCATGETS("touch "));
2649         append_string(wcs_buffer, &touch_string, FIND_LENGTH);
2650         touch_cmd = name;
2651         if (name->has_vpath_alias_prop) {
2652             touch_cmd = get_prop(name->prop,
2653                                 vpath_alias_prop->
2654                                 body.vpath_alias.alias);
2655         }
2656         APPEND_NAME(touch_cmd,
2657                     &touch_string,
2658                     FIND_LENGTH);
2659         touch_cmd = GETNAME(touch_string.buffer.start,
2660                             FIND_LENGTH);
2661         if (touch_string.free_after_use) {
2662             retmem(touch_string.buffer.start);
2663         }
2664         if (!silent ||
2665             do_not_exec_rule &&
2666             (target_group == NULL)) {
2667             (void) printf("%s\n", touch_cmd->string_mb);
2668             SEND_MTOOL_MSG(
2669                 job_result_msg->appendOutput(AVO_STRDUP(
2670                     ));
2671         }
2672         /* Run the touch command, or simulate it */
2673         if (!do_not_exec_rule) {
2674
2675             SEND_MTOOL_MSG(
2676                 (void) sprintf(mbstring,
2677                               NOCATGETS("%s/make.stdou
2678                               tmpdir,
2679                               getpid(),
2680                               file_number++);
2681
2682                 int tmp_fd = mkstemp(mbstring);
2683                 if(tmp_fd) {
2684                     (void) close(tmp_fd);
2685                 }
2686
2687                 stdout_file = strdup(mbstring);
2688                 stderr_file = NULL;
2689                 child_pid = pollResults(stdout_file,
2690                                       (char *)NULL,
2691                                       (char *)NULL);
2692             );
2693
2694             result = dosys(touch_cmd,

```

```

2695         false,
2696         false,
2697         false,
2698         false,
2699         name,
2700         send_mtool_msgs);
2701
2702         SEND_MTOOL_MSG(
2703             append_job_result_msg(job_result_msg);
2704             if (child_pid > 0) {
2705                 kill(child_pid, SIGUSR1);
2706                 while (!((waitpid(child_pid, 0,
2707                                 && (errno == ECHILD)))));
2708             }
2709             child_pid = 0;
2710             (void) unlink(stdout_file);
2711             retmem_mb(stdout_file);
2712             stdout_file = NULL;
2713         );
2714     } else {
2715         result = build_ok;
2716     }
2717 } else {
2718     result = build_ok;
2719 }
2720 if (target_group == NULL) {
2721     target_group = line->body.line.target_group;
2722 } else {
2723     target_group = target_group->next;
2724 }
2725 if (target_group != NULL) {
2726     name = target_group->name;
2727 } else {
2728     name = NULL;
2729 }
2730 }
2731 }
2732 SEND_MTOOL_MSG(
2733     job_result_msg->setResult(job_msg_id, (result == build_ok) ? 0 :
2734                             xdr_msg = (RWCollectable*) job_result_msg;
2735     xdr(&xdrs, xdr_msg);
2736     (void) fflush(mtool_msgs_fp);
2737     delete job_result_msg;
2738 );
2739 return result;
2740 }

```

unchanged portion omitted

```

3463 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
3464 void
3465 create_xdrs_ptr(void)
3466 {
3467     static int     xdrs_init = 0;
3468
3469     if (!xdrs_init) {
3470         xdrs_init = 1;
3471         mtool_msgs_fp = fdopen(mtool_msgs_fd, "a");
3472         xdrstdio_create(&xdrs,
3473                        mtool_msgs_fp,
3474                        XDR_ENCODE);
3475     }
3476 }
3477
3478 XDR *
3479 get_xdrs_ptr(void)
3480 {

```

```

3481     return &xdrs;
3482 }

3484 FILE *
3485 get_mtool_msgs_fp(void)
3486 {
3487     return mtool_msgs_fp;
3488 }

3490 int
3491 get_job_msg_id(void)
3492 {
3493     return job_msg_id;
3494 }

3496 // Continuously poll and show the results of remotely executing a job,
3497 // i.e., output the stdout and stderr files.

3499 static int
3500 pollResults(char *outFn, char *errFn, char *hostNm)
3501 {
3502     int child;

3504     child = fork();
3505     switch (child) {
3506     case -1:
3507         break;
3508     case 0:
3509         enable_interrupt((void (*) (int))SIG_DFL);
3510         (void) sigset(SIGUSR1, Avo_PollResultsAction_Sigusr1Handler);
3511         pollResultsAction(outFn, errFn);

3513         exit(0);
3514         break;
3515     default:
3516         break;
3517     }
3518     return child;
3519 }

3521 // This is the PollResultsAction SIGUSR1 handler.

3523 static bool_t pollResultsActionTimeToFinish = FALSE;

3525 extern "C" void
3526 Avo_PollResultsAction_Sigusr1Handler(int foo)
3527 {
3528     pollResultsActionTimeToFinish = TRUE;
3529 }

3531 static void
3532 pollResultsAction(char *outFn, char *errFn)
3533 {
3534     int fd;
3535     time_t file_time = 0;
3536     long file_time_nsec = 0;
3537     struct stat statbuf;
3538     int stat_rc;

3540     // Keep stat'ing until file exists.
3541     while ((stat_rc = stat(outFn, &statbuf)) != 0) &&
3542         (errno == ENOENT) &&
3543         !pollResultsActionTimeToFinish) {
3544         us_sleep(STAT_RETRY_SLEEP_TIME);
3545     }
3546     // The previous stat() could be failed due to EINTR

```

```

3547     // So one more try is needed
3548     if (stat_rc != 0 && stat(outFn, &statbuf) != 0) {
3549         // stat() failed
3550         warning(NOCATGETS("Internal error: stat(\"%s\", ...) failed: %s\
3551             outFn, strerror(errno));
3552         exit(1);
3553     }

3555     if ((fd = open(outFn, O_RDONLY)) < 0
3556         && (errno != EINTR || (fd = open(outFn, O_RDONLY)) < 0)) {
3557         // open() failed
3558         warning(NOCATGETS("Internal error: open(\"%s\", O_RDONLY) failed
3559             outFn, strerror(errno));
3560         exit(1);
3561     }

3563     while (!pollResultsActionTimeToFinish && stat(outFn, &statbuf) == 0) {
3564         if ((statbuf.st_mtim.tv_sec > file_time) ||
3565             ((statbuf.st_mtim.tv_sec == file_time) &&
3566              (statbuf.st_mtim.tv_nsec > file_time_nsec))
3567         ) {
3568             file_time = statbuf.st_mtim.tv_sec;
3569             file_time_nsec = statbuf.st_mtim.tv_nsec;
3570             rxmGetNextResultsBlock(fd);
3571         }
3572         us_sleep(STAT_RETRY_SLEEP_TIME);
3573     }
3574     // Check for the rest of output
3575     rxmGetNextResultsBlock(fd);

3577     (void) close(fd);
3578 }

3580 static void
3581 rxmGetNextResultsBlock(int fd)
3582 {
3583     size_t to_read = 8 * 1024;
3584     ssize_t bytes_read;
3585     ssize_t bytes_written;
3586     char results_buf[8 * 1024];
3587     sigset_t newset;
3588     sigset_t oldset;

3590     // Read some more from the output/results file.
3591     // Hopefully the kernel managed to prefetch the stuff.
3592     bytes_read = read(fd, results_buf, to_read);
3593     while (bytes_read > 0) {
3594         AVO_BLOCK_INTERRUPTS;
3595         bytes_written = write(1, results_buf, bytes_read);
3596         AVO_UNBLOCK_INTERRUPTS;
3597         if (bytes_written != bytes_read) {
3598             // write() failed
3599             warning(NOCATGETS("Internal error: write(1, ...) failed:
3600                 strerror(errno));
3601             exit(1);
3602         }
3603         bytes_read = read(fd, results_buf, to_read);
3604     }
3605 }

3607 // Generic, interruptable microsecond resolution sleep member function.

3609 static int
3610 us_sleep(unsigned int nusecs)
3611 {
3612     struct pollfd dummy;

```

```
3613     int timeout;
3615     if ((timeout = nusecs/1000) <= 0) {
3616         timeout = 1;
3617     }
3618     return poll(&dummy, 0, timeout);
3619 }
3620 #endif /* TEAMWARE_MAKE_CMN */

3284 // Recursively delete each of the Chain struct on the chain.

3286 static void
3287 delete_query_chain(Chain ch)
3288 {
3289     if (ch == NULL) {
3290         return;
3291     } else {
3292         delete_query_chain(ch->next);
3293         retmem_mb((char *) ch);
3294     }
3295 }
_____unchanged_portion_omitted_
```

```

*****
4182 Wed May 20 11:55:52 2015
new/usr/src/cmd/make/bin/dosys.cc
make: undef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      dosys.cc
28  *
29  *      Execute one commandline
30  */

32 /*
33  * Included files
34  */
35 #include <fcntl.h>          /* open() */
36 #include <mk/defs.h>
37 #include <mksh/dosys.h>    /* doshell(), doexec() */
38 #include <mksh/misc.h>    /* getmem() */
39 #include <sys/stat.h>     /* open() */
40 #include <unistd.h>       /* getpid() */

42 /*
43  * Defined macros
44  */

46 /*
47  * typedefs & structs
48  */

50 /*
51  * Static variables
52  */
53 static int      filter_file;
54 static char     *filter_file_name;

56 /*
57  * File table of contents
58  */
59 static void     redirect_stderr(void);

61 /*

```

```

62 *      dosys(command, ignore_error, call_make, silent_error, target)
63 *
64 *      Check if command string contains meta chars and dispatch to
65 *      the proper routine for executing one command line.
66 *
67 *      Return value:
68 *
69 *
70 *      Parameters:
71 *      command      The command to run
72 *      ignore_error Should make abort when an error is seen?
73 *      call_make    Did command reference $(MAKE) ?
74 *      silent_error Should error messages be suppressed for pmake?
75 *      target       Target we are building
76 *
77 *      Global variables used:
78 *      do_not_exec_rule Is -n on?
79 *      working_on_targets We started processing real targets
80 */
81 Doname
82 dosys(register Name command, register Boolean ignore_error, register Boolean cal
83 {
84     timestruc_t      before;
85     register int     length = command->hash.length;
86     Wstring          wcb(command);
87     register wchar_t *p = wcb.get_string();
88     register wchar_t *q;
89     Doname           result;

91     /* Strip spaces from head of command string */
92     while (iswspace(*p)) {
93         p++, length--;
94     }
95     if (*p == (int) nul_char) {
96         return build_failed;
97     }
98     /* If we are faking it we just return */
99     if (do_not_exec_rule &&
100         working_on_targets &&
101         !call_make &&
102         !always_exec) {
103         return build_ok;
104     }
105     /* no_action_was_taken is used to print special message */
106     no_action_was_taken = false;

108     /* Copy string to make it OK to write it. */
109     q = ALLOC_WC(length + 1);
110     (void) wscopy(q, p);
111     /* Write the state file iff this command uses make. */
112     if (call_make && command_changed) {
113         write_state_file(0, false);
114     }
115     make_state->stat.time = file_no_time;
116     (void)exists(make_state);
117     before = make_state->stat.time;
118     /*
119     * Run command directly if it contains no shell meta chars,
120     * else run it using the shell.
121     */
122     if (await(ignore_error,
123             silent_error,
124             target,
125             wcb.get_string(),
126             command->meta ?
127             doshell(q, ignore_error, redirect_out_err,

```

```
128         stdout_file, stderr_file, 0) :
129         doexec(q, ignore_error, redirect_out_err,
130         stdout_file, stderr_file,
131         vroot_path, 0),
132         send_mtool_msgs,
133 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
134         get_xdrs_ptr(),
135         get_job_msg_id()
136 #else
137         NULL,
138         -1
139 #endif
140     )) {
141         result = build_ok;
142     } else {
143         result = build_failed;
144     }
145     retmem(q);
146
147     if ((report_dependencies_level == 0) &&
148         call_make) {
149         make_state->stat.time = file_no_time;
150         (void)exists(make_state);
151         if (before == make_state->stat.time) {
152             return result;
153         }
154         makefile_type = reading_statefile;
155         if (read_trace_level > 1) {
156             trace_reader = true;
157         }
158         temp_file_number++;
159         (void) read_simple_file(make_state,
160                               false,
161                               false,
162                               false,
163                               false,
164                               false,
165                               true);
166         trace_reader = false;
167     }
168     return result;
169 }
```

unchanged portion omitted

```

*****
4711 Wed May 20 11:55:53 2015
new/usr/src/cmd/make/bin/globals.cc
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      globals.cc
28  *
29  *      This declares all global variables
30  */

32 /*
33  * Included files
34  */
35 #include <nl_types.h>
36 #include <mk/defs.h>
37 #include <sys/stat.h>

39 /*
40  * Defined macros
41  */

43 /*
44  * typedefs & structs
45  */

47 /*
48  * Global variables used by make only
49  */
50 FILE          *dependency_report_file;

52 /*
53  * Global variables used by make
54  */
55 Boolean      allrules_read=false;
56 Name        posix_name;
57 Name        svr4_name;
58 Boolean      sdot_target; /* used to identify s.m(/M)akefile */
59 Boolean      all_parallel; /* TEAMWARE_MAKE_CMN */
60 Boolean      assign_done;
61 int foo;

```

```

62 Boolean      build_failed_seen;
63 #ifdef DISTRIBUTED
64 Boolean      building_serial;
65 #endif
66 Name        built_last_make_run;
67 Name        c_at;
68 #ifdef DISTRIBUTED
69 Boolean      called_make = false;
70 #endif
71 Boolean      cleanup;
72 Boolean      close_report;
73 Boolean      command_changed;
74 Boolean      commands_done;
75 Chain       conditional_targets;
76 Name        conditionals;
77 Boolean     continue_after_error; /* '-k' */
78 Property    current_line;
79 Name        current_make_version;
80 Name        current_target;
81 short       debug_level;
82 Cmd_line    default_rule;
83 Name        default_rule_name;
84 Name        default_target_to_build;
85 Name        dmake_group;
86 Name        dmake_max_jobs;
87 Name        dmake_mode;
88 DMake_mode  dmake_mode_type;
89 Name        dmake_output_mode;
90 DMake_output_mode output_mode = txt1_mode;
91 Name        dmake_odir;
92 Name        dmake_rcfile;
93 Name        done;
94 Name        dot;
95 Name        dot_keep_state;
96 Name        dot_keep_state_file;
97 Name        empty_name;
98 Boolean     fatal_in_progress;
99 int         file_number;
100 #if 0
101 Boolean     filter_stderr; /* '-X' */
102 #endif
103 Name        force;
104 Name        ignore_name;
105 Boolean     ignore_errors; /* '-i' */
106 Boolean     ignore_errors_all; /* '-i' */
107 Name        init;
108 int         job_msg_id;
109 Boolean     keep_state;
110 Name        make_state;
111 #ifdef TEAMWARE_MAKE_CMN
112 timestruc_t make_state_before;
113 #endif
114 Dependency  makefiles_used;
115 Name        makeflags;
116 Boolean     make_state_locked; // Moved to lib/mksh
117 Name        make_version;
118 char        mbs_buffer2[(MAXPATHLEN * MB_LEN_MAX)];
119 char        *mbs_ptr;
120 char        *mbs_ptr2;
121 int         mtool_msgs_fd;
122 Boolean     depinfo_already_read = false;
123 Boolean     no_action_was_taken = true; /* true if we've not **
                                          ** run any command */
124 Boolean     no_parallel = false; /* TEAMWARE_MAKE_CMN */
125 Name        no_parallel_name;

```

```

122     Name           not_auto;
123     Boolean        only_parallel;           /* TEAMWARE_MAKE_CMN */
124     Boolean        parallel;               /* TEAMWARE_MAKE_CMN */
125     Name           parallel_name;
126     Name           localhost_name;
127     int            parallel_process_cnt;
128     Percent        percent_list;
129     Dyntarget      dyntarget_list;
130     Name           plus;
131     Name           pmake_machinesfile;
132     Name           precious;
133     Name           primary_makefile;
134     Boolean        quest;                  /* '-q' */
135     short          read_trace_level;
136     Boolean        reading_dependencies = false;
137     Name           recursive_name;
138     int            recursion_level;
139     short          report_dependencies_level = 0; /* -P */
140     Boolean        report_pwd;
141     Boolean        rewrite_statefile;
142     Running        running_list;
143     char           *sccs_dir_path;
144     Name           sccs_get_name;
145     Name           sccs_get_posix_name;
146     Cmd_line       sccs_get_rule;
147     Cmd_line       sccs_get_org_rule;
148     Cmd_line       sccs_get_posix_rule;
149     Name           get_name;
150     Cmd_line       get_rule;
151     Name           get_posix_name;
152     Cmd_line       get_posix_rule;
153     Boolean        send_mtool_msgs;       /* '-K' */
154     Boolean        all_precious;
155     Boolean        silent_all;           /* '-s' */
156     Boolean        report_cwd;           /* '-w' */
157     Boolean        silent;               /* '-s' */
158     Name           silent_name;
159     char           *stderr_file = NULL;
160     char           *stdout_file = NULL;
161     Boolean        stdout_stderr_same;
162     Dependency     suffixes;
163     Name           suffixes_name;
164     Name           sunpro_dependencies;
165     Boolean        target_variants;
166     const char     *tmpdir = NOCATGETS("/tmp");
167     const char     *temp_file_directory = NOCATGETS(".");
168     Name           temp_file_name;
169     short          temp_file_number;
170     time_t         timing_start;
171     wchar_t        *top_level_target;
172     Boolean        touch;                 /* '-t' */
173     Boolean        trace_reader;          /* '-D' */
174     Boolean        build_unconditional;   /* '-u' */
175     pathpt        vroot_path = VROOT_DEFAULT;
176     Name           wait_name;
177     wchar_t        wcs_buffer2[MAXPATHLEN];
178     wchar_t        *wcs_ptr;
179     wchar_t        *wcs_ptr2;
180     nl_catd        catd;
181     long int       hostid;

183 /*
184 * File table of contents
185 */

```



```
*****
```

```
4000 Wed May 20 11:55:55 2015
```

```
new/usr/src/cmd/make/bin/macro.cc
```

```
make: undef for MAKETOOL and DISTRIBUTED (undefined)
```

```
*****
```

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      macro.cc
28  *
29  *      Handle expansion of make macros
30  */

32 /*
33  * Included files
34  */
35 #ifdef DISTRIBUTED
36 #include <avo/strings.h>      /* AVO_STRDUP() */
37 #include <dm/Avo_DoJobMsg.h>
38 #endif
39 #include <mk/defs.h>
40 #include <mksh/macro.h>      /* getvar(), expand_value() */
41 #include <mksh/misc.h>      /* getmem() */

39 /*
40  * Defined macros
41  */

43 /*
44  * typedefs & structs
45  */

47 /*
48  * Static variables
49  */

51 /*
52  * File table of contents
53  */

55 void
56 setvar_append(register Name name, register Name value)
57 {
```

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

66     if(macro_apx == NULL) {
67         macro_apx = append_prop(name, macro_append_prop);
68         if(macro != NULL) {
69             macro_apx->body.macro_appendix.value = macro->body.macro
70         }
71     }

73     val = macro_apx->body.macro_appendix.value_to_append;

75     INIT_STRING_FROM_STACK(destination, buffer);
76     buffer[0] = 0;
77     if (val != NULL) {
78         APPEND_NAME(val,
79                     &destination,
80                     (int) val->hash.length);
81         if (value != NULL) {
82             MBTOWC(wcs_buffer, " ");
83             append_char(wcs_buffer[0], &destination);
84         }
85     }
86     if (value != NULL) {
87         APPEND_NAME(value,
88                     &destination,
89                     (int) value->hash.length);
90     }
91     value = GETNAME(destination.buffer.start, FIND_LENGTH);
92     if (destination.free_after_use) {
93         retmem(destination.buffer.start);
94     }
95     macro_apx->body.macro_appendix.value_to_append = value;

97     SETVAR(name, empty_name, true);
98 }

100 /*
101  *      setvar_envvar()
102  *
103  *      This function scans the list of environment variables that have
104  *      dynamic values and sets them.
105  *
106  *      Parameters:
107  *
108  *      Global variables used:
109  *          envvar      A list of environment vars with $ in value
110  */
111 void
112 #ifdef DISTRIBUTED
113 setvar_envvar(Avo_DoJobMsg *dmake_job_msg)
114 #else
115 setvar_envvar(void)
116 #endif
117 {
118     wchar_t               buffer[STRING_BUFFER_LENGTH];
119     int                    length;
120 #ifdef DISTRIBUTED
121     Property              macro;
122 #endif
123     register              char *mbs, *tmp_mbs_buffer = NULL;
```

```

117     register      char      *env, *tmp_mbs_buffer2 = NULL;
118     Envvar        p;
119     String_rec    value;

121     for (p = envvar; p != NULL; p = p->next) {
122         if (p->already_put
134 #ifdef DISTRIBUTED
135         && !dmake_job_msg
136 #endif
123         ) {
124             continue;
125         }
126         INIT_STRING_FROM_STACK(value, buffer);
127         expand_value(p->value, &value, false);
128         if ((length = wslen(value.buffer.start)) >= MAXPATHLEN) {
129             mbs = tmp_mbs_buffer = getmem((length + 1) * MB_LEN_MAX)
130             (void) wcstombs(mbs,
131                 value.buffer.start,
132                 (length + 1) * MB_LEN_MAX);
133         } else {
134             mbs = mbs_buffer;
135             WCSTOMBS(mbs, value.buffer.start);
136         }
137         length = 2 + strlen(p->name->string_mb) + strlen(mbs);
138         if (!p->already_put || length > (MAXPATHLEN * MB_LEN_MAX)) {
139             env = tmp_mbs_buffer = getmem(length);
140         } else {
141             env = mbs_buffer2;
142         }
143         (void) sprintf(env,
144             "%s=%s",
145             p->name->string_mb,
146             mbs);
147         if (!p->already_put) {
148             (void) putenv(env);
149             p->already_put = true;
150             if (p->env_string) {
151                 retmem_mb(p->env_string);
152             }
153             p->env_string = env;
154             tmp_mbs_buffer2 = NULL; // We should not return this mem
155         }
170 #ifdef DISTRIBUTED
171         if (dmake_job_msg) {
172             dmake_job_msg->appendVar(env);
173         }
174 #endif
156         if (tmp_mbs_buffer2) {
157             retmem_mb(tmp_mbs_buffer2);
158             tmp_mbs_buffer2 = NULL;
159         }
160         if (tmp_mbs_buffer) {
161             retmem_mb(tmp_mbs_buffer);
162             tmp_mbs_buffer = NULL;
163         }
164     }
184 #ifdef DISTRIBUTED
185     /* Append SUNPRO_DEPENDENCIES to the dmake_job_msg. */
186     if (keep_state && dmake_job_msg) {
187         macro = get_prop(sunpro_dependencies->prop, macro_prop);
188         length = 2 +
189             strlen(sunpro_dependencies->string_mb) +
190             strlen(macro->body.macro.value->string_mb);
191         if (length > (MAXPATHLEN * MB_LEN_MAX)) {
192             env = tmp_mbs_buffer2 = getmem(length);
193         } else {

```

```

194             env = mbs_buffer2;
195         }
196         (void) sprintf(env,
197             "%s=%s",
198             sunpro_dependencies->string_mb,
199             macro->body.macro.value->string_mb);
200         dmake_job_msg->appendVar(env);
201         if (tmp_mbs_buffer2) {
202             retmem_mb(tmp_mbs_buffer2);
203             tmp_mbs_buffer2 = NULL;
204         }
205     }
206 #endif
165 }
_____unchanged_portion_omitted_____

```

new/usr/src/cmd/make/bin/main.cc

1

```
*****
94173 Wed May 20 11:55:55 2015
new/usr/src/cmd/make/bin/main.cc
make: undef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      main.cc
28  *
29  *      make program main routine plus some helper routines
30  */
31
32 /*
33  * Included files
34  */
35 #if defined(TEAMWARE_MAKE_CMN)
36 #   include <avo/int1.h>
37 #endif

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

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

42 #include <locale.h>          /* setlocale() */
43 #include <mk/defs.h>
44 #include <mksdms18n/mksdms18n.h> /* libmksdms18n_init() */
45 #include <mksh/macro.h>      /* getvar() */
46 #include <mksh/misc.h>       /* getmem(), setup_char_semantics() */

48 #if defined(TEAMWARE_MAKE_CMN)
49 #endif

51 #include <pwd.h>              /* getpwnam() */
52 #include <setjmp.h>
53 #include <signal.h>
54 #include <stdlib.h>
55 #include <sys/errno.h>        /* ENOENT */
56 #include <sys/stat.h>        /* fstat() */
```

new/usr/src/cmd/make/bin/main.cc

2

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

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

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

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

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

73 #ifdef MAXJOBS_ADJUST_RFE4694000
74 extern void job_adjust_fini();
75 #endif /* MAXJOBS_ADJUST_RFE4694000 */
76 #endif /* TEAMWARE_MAKE_CMN */

79 /*
80  * Defined macros
81  */
82 #define MAKE_PREFIX           NOCATGETS("/usr")
83 #define LD_SUPPORT_ENV_VAR    NOCATGETS("SGS_SUPPORT_32")
84 #define LD_SUPPORT_ENV_VAR_32 NOCATGETS("SGS_SUPPORT_32")
85 #define LD_SUPPORT_ENV_VAR_64 NOCATGETS("SGS_SUPPORT_64")
86 #define LD_SUPPORT_MAKE_LIB   NOCATGETS("libmakestate.so.1")
87 #define LD_SUPPORT_MAKE_LIB_DIR NOCATGETS("/lib")
88 #define LD_SUPPORT_MAKE_LIB_DIR_64 NOCATGETS("/64")

90 /*
91  * typedefs & structs
92  */

94 /*
95  * Static variables
96  */
97 static char      *argv_zero_string;
98 static Boolean   build_failed_ever_seen;
99 static Boolean   continue_after_error_ever_seen; /* '-k' */
100 static Boolean   dmake_group_specified;          /* '-g' */
101 static Boolean   dmake_max_jobs_specified;       /* '-j' */
102 static Boolean   dmake_mode_specified;          /* '-m' */
103 static Boolean   dmake_add_mode_specified;       /* '-x' */
104 static Boolean   dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE */
105 static Boolean   dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M */
106 static Boolean   dmake_odir_specified;          /* '-o' */
107 static Boolean   dmake_rcfile_specified;        /* '-c' */
108 static Boolean   env_wins;                       /* '-e' */
109 static Boolean   ignore_default_mk;             /* '-r' */
110 static Boolean   list_all_targets;              /* '-T' */
111 static int       mf_argc;
112 static char      **mf_argv;
113 static Dependency_rec not_auto_depen_struct;
114 static Dependency   not_auto_depen = &not_auto_depen_struct;
115 static Boolean   pmake_cap_r_specified;         /* '-R' */
116 static Boolean   pmake_machinesfile_specified; /* '-M' */
117 static Boolean   stop_after_error_ever_seen;   /* '-S' */
118 static Boolean   trace_status;                  /* '-p' */

120 #ifndef DMAKE_STATISTICS
121 static Boolean   getname_stat = false;
122 #endif
```

```

124 #if defined(TEAMWARE_MAKE_CMN)
125     static time_t      start_time;
126     static int         g_argc;
127     static char        **g_argv;
128 #endif

130 /*
131  * File table of contents
132  */
133     extern "C" void      cleanup_after_exit(void);

135 #ifndef TEAMWARE_MAKE_CMN
136 extern "C" {
137     extern void          dmake_exit_callback(void);
138     extern void          dmake_message_callback(char *);
139 }
140 #endif

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

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

146 static void             append_makeflags_string(Name, String);
147 static void             doalarm(int);
148 static void             enter_argv_values(int , char **, ASCII_Dyn_Array *);
149 static void             make_targets(int, char **, Boolean);
150 static int              parse_command_option(char);
151 static void             read_command_options(int, char **);
152 static void             read_environment(Boolean);
153 static void             read_files_and_state(int, char **);
154 static Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
155 static void             report_recursion(Name);
156 static void             set_sgs_support(void);
157 static void             setup_for_projectdir(void);
158 static void             setup_makeflags_argv(void);
159 static void             report_dir_enter_leave(Boolean entering);

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

168 #ifndef DISTRIBUTED
169     extern int           dmake_ofd;
170     extern FILE*         dmake_ofp;
171     extern int           rxmPid;
172     extern XDR           xdrs_out;
173 #endif
163 #ifndef TEAMWARE_MAKE_CMN
164     static const char   verstring[] = "illumos make";
165 #endif

167 jmp_buf jmpbuffer;
168 extern nl_catd catd;

170 /*
171  *   main(argc, argv)
172  *
173  *   Parameters:
174  *       argc           You know what this is
175  *       argv           You know what this is
176  *
177  *   Static variables used:
178  *       list_all_targets   make -T seen
179  *       trace_status       make -p seen
180  *
181  *   Global variables used:
182  *       debug_level       Should we trace make actions?

```

```

183  *           keep_state           Set if .KEEP_STATE seen
184  *           makeflags           The Name "MAKEFLAGS", used to get macro
185  *           remote_command_name Name of remote invocation cmd ("on")
186  *           running_list        List of parallel running processes
187  *           stdout_stderr_same  true if stdout and stderr are the same
188  *           auto_dependencies   The Name "SUNPRO_DEPENDENCIES"
189  *           temp_file_directory Set to the dir where we create tmp file
190  *           trace_reader        Set to reflect tracing status
191  *           working_on_targets  Set when building user targets
192  */
193 int
194 main(int argc, char *argv[])
195 {
196     /*
197      * cp is a -> to the value of the MAKEFLAGS env var,
198      * which has to be regular chars.
199      */
200     register char        *cp;
201     char                 make_state_dir[MAXPATHLEN];
202     Boolean              parallel_flag = false;
203     char                 *progrnameptr;
204     char                 *slash_ptr;
205     mode_t               um;
206     int                  i;
207 #ifndef TEAMWARE_MAKE_CMN
208     struct itimerval     value;
209     char                 def_dmakerc_path[MAXPATHLEN];
210     Name                 dmake_name, dmake_name2;
211     Name                 dmake_value, dmake_value2;
212     Property             prop, prop2;
213     struct stat          statbuf;
214     int                  statval;
215 #endif

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

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

224 #ifndef DMAKE_STATISTICS
225     if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
226         getname_stat = true;
227     }
228 #endif

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

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

237 #if defined(TEAMWARE_MAKE_CMN)
238 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
239  /*
240  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
241  * from avo_util library.
242  */
243 #endif
244     libmksdmsi18n_init();
245 #endif

246 #ifndef TEAMWARE_MAKE_CMN
247     textdomain(NOCATGETS("SUNW_SPRO_MAKE"));

```

```

248 #endif /* TEAMWARE_MAKE_CMN */

250 #ifdef TEAMWARE_MAKE_CMN
251     g_argc = argc;
252     g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
253     for (i = 0; i < argc; i++) {
254         g_argv[i] = argv[i];
255     }
256     g_argv[i] = NULL;
257 #endif /* TEAMWARE_MAKE_CMN */

259 /*
260  * Set argv_zero_string to some form of argv[0] for
261  * recursive MAKE builds.
262  */

264 if (*argv[0] == (int) slash_char) {
265     /* argv[0] starts with a slash */
266     argv_zero_string = strdup(argv[0]);
267 } else if (strchr(argv[0], (int) slash_char) == NULL) {
268     /* argv[0] contains no slashes */
269     argv_zero_string = strdup(argv[0]);
270 } else {
271     /*
272      * argv[0] contains at least one slash,
273      * but doesn't start with a slash
274      */
275     char *tmp_current_path;
276     char *tmp_string;

278     tmp_current_path = get_current_path();
279     tmp_string = getmem(strlen(tmp_current_path) + 1 +
280                       strlen(argv[0]) + 1);
281     (void) sprintf(tmp_string,
282                  "%s/%s",
283                  tmp_current_path,
284                  argv[0]);
285     argv_zero_string = strdup(tmp_string);
286     retmem_mb(tmp_string);
287 }

289 /*
290  * The following flags are reset if we don't have the
291  * (.nse_depinfo or .make.state) files locked and only set
292  * AFTER the file has been locked. This ensures that if the user
293  * interrupts the program while file_lock() is waiting to lock
294  * the file, the interrupt handler doesn't remove a lock
295  * that doesn't belong to us.
296  */
297 make_state_lockfile = NULL;
298 make_state_locked = false;

301 /*
302  * look for last slash char in the path to look at the binary
303  * name. This is to resolve the hard link and invoke make
304  * in svr4 mode.
305  */

307 /* Sun OS make standart */
308 svr4 = false;
309 posix = false;
310 if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
311     svr4 = false;
312     posix = true;
313 } else {

```

```

314     prognameptr = strrchr(argv[0], '/');
315     if(prognameptr) {
316         prognameptr++;
317     } else {
318         prognameptr = argv[0];
319     }
320     if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
321         svr4 = true;
322         posix = false;
323     }
324 }
325 if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
326     svr4 = true;
327     posix = false;
328 }

330 /*
331  * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
332  */
333 char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"));
334 if (dmake_compat_mode_var != NULL) {
335     if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
336         gnu_style = true;
337     }
338     //svr4 = false;
339     //posix = false;
340 }

342 /*
343  * Temporary directory set up.
344  */
345 char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
346 if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
347     strcpy(mbs_buffer, tmpdir_var);
348     for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
349         *--tmpdir_var == '/' && tmpdir_var > mbs_buffer;
350         *tmpdir_var = '\0');
351     if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dmake
352         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
353             mbs_buffer, getpid());
354         int fd = mkstemp(mbs_buffer2);
355         if (fd >= 0) {
356             close(fd);
357             unlink(mbs_buffer2);
358             tmpdir = strdup(mbs_buffer);
359         }
360     }
361 }

363 /* find out if stdout and stderr point to the same place */
364 if (fstat(1, &out_stat) < 0) {
365     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
366         );
367 }
368 if (fstat(2, &err_stat) < 0) {
369     fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s")
370         );
371 }
372 if ((out_stat.st_dev == err_stat.st_dev) &&
373     (out_stat.st_ino == err_stat.st_ino)) {
374     stdout_stderr_same = true;
375 } else {
376     stdout_stderr_same = false;
377 }
378 /* Make the vroot package scan the path using shell semantics */
379 set_path_style(0);

390 setup_char_semantics();

```

```

381     setup_for_projectdir();

383     /*
384     * If running with .KEEP_STATE, curdir will be set with
385     * the connected directory.
386     */
387     (void) atexit(cleanup_after_exit);

389     load_cached_names();

391 /*
392 *
393 */
394     setup_makeflags_argv();
395     read_command_options(mf_argc, mf_argv);
396     read_command_options(argc, argv);
397     if (debug_level > 0) {
398         cp = getenv(makeflags->string_mb);
399         (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp);
400     }

402     setup_interrupt(handle_interrupt);

404     read_files_and_state(argc, argv);

406 #ifdef TEAMWARE_MAKE_CMN
407     /*
408     * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
409     */
410     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
411     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
412     prop2 = get_prop(dmake_name2->prop, macro_prop);
413     if (prop2 == NULL) {
414         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
415         output_mode = txt1_mode;
416     } else {
417         dmake_value2 = prop2->body.macro.value;
418         if ((dmake_value2 == NULL) ||
419             (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
420             output_mode = txt1_mode;
421         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
422             output_mode = txt2_mode;
423         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1")))
424             output_mode = html1_mode;
425         } else {
426             warning(catgets(catd, 1, 352, "Unsupported value '%s' fo
427                 dmake_value2->string_mb);
428         }
429     }
430     /*
431     * Find the dmake_mode: distributed, parallel, or serial.
432     */
433     if ((!pmake_cap_r_specified) &&
434         (!pmake_machinesfile_specified)) {
435         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
436         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
437         prop2 = get_prop(dmake_name2->prop, macro_prop);
438         if (prop2 == NULL) {
439             /* DMAKE_MODE not defined, default to distributed mode */
440             dmake_mode_type = distributed_mode;
441             no_parallel = false;
442         } else {
443             dmake_value2 = prop2->body.macro.value;
444             if ((dmake_value2 == NULL) ||
445                 (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))

```

```

446         dmake_mode_type = distributed_mode;
447         no_parallel = false;
448     } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
449         dmake_mode_type = parallel_mode;
450         no_parallel = false;
451     } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
452         dmake_mode_type = serial_mode;
453         no_parallel = true;
454     } else {
455         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
456     }
457 }

459     if ((!list_all_targets) &&
460         (report_dependencies_level == 0)) {
461         /*
462         * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
463         * They could be defined in the env, in the makefile, or on the
464         * command line.
465         * If neither is defined, and $(HOME)/.dmakerc does not exists,
466         * then print a message, and default to parallel mode.
467         */
468     #ifdef DISTRIBUTED
469         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
470         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
471         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
472         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
473         if (((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
474             ((dmake_value = prop->body.macro.value) == NULL)) &&
475             (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
476             ((dmake_value2 = prop2->body.macro.value) == NULL))) {
477             Boolean empty_dmaker = true;
478             char *homedir = getenv(NOCATGETS("HOME"));
479             if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
480                 sprintf(def_dmaker_path, NOCATGETS("%s/.dmakerc
481                 if (((statval = stat(def_dmaker_path, &statbuf
482                     ((statval == 0) && (statbuf.st_size == 0
483                 } else {
484                     Avo_dmaker      *rcfile = new Avo_dmaker
485                     Avo_err        *err = rcfile->read(def_
486                     if (err) {
487                         fatal(err->str);
488                     }
489                     empty_dmaker = rcfile->was_empty();
490                     delete rcfile;
491                 }
492             }
493             if (empty_dmaker) {
494                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
495                     putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
496                     (void) fprintf(stdout, catgets(catd, 1,
497                     (void) fprintf(stdout, catgets(catd, 1,
498                 }
499                 dmake_mode_type = parallel_mode;
500                 no_parallel = false;
501             }
502         }
503     }
504     #endif
505     #endif /* DISTRIBUTED */
506 }
507 #endif

```

```

476 #ifdef TEAMWARE_MAKE_CMN
477     parallel_flag = true;
478     putenv(strdup(NOCATGETS("DMAKE_CHILD=TRUE")));
480 //
481 // If dmake is running with -t option, set dmake_mode_type to serial.
482 // This is done because doname() calls touch_command() that runs serially.
483 // If we do not do that, maketool will have problems.
484 //
485     if(touch) {
486         dmake_mode_type = serial_mode;
487         no_parallel = true;
488     }
489 #else
490     parallel_flag = false;
491 #endif
493 #if defined (TEAMWARE_MAKE_CMN)
494     /*
495     * Check whether stdout and stderr are physically same.
496     * This is in order to decide whether we need to redirect
497     * stderr separately from stdout.
498     * This check is performed only if __DMAKE_SEPARATE_STDERR
499     * is not set. This variable may be used in order to preserve
500     * the 'old' behaviour.
501     */
502     out_err_same = true;
503     char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
504     if (dmake_sep_var == NULL || (0 != strcmp(dmake_sep_var, NOCATGETS("
505         struct stat stdout_stat;
506         struct stat stderr_stat;
507         if( (fstat(1, &stdout_stat) == 0)
508             && (fstat(2, &stderr_stat) == 0) )
509         {
510             if( (stdout_stat.st_dev != stderr_stat.st_dev)
511                 || (stdout_stat.st_ino != stderr_stat.st_ino) )
512             {
513                 out_err_same = false;
514             }
515         }
516     }
517 #endif
567 #ifdef DISTRIBUTED
568     /*
569     * At this point, DMake should startup an rxm with any and all
570     * DMake command line options. Rxm will, among other things,
571     * read the rc file.
572     */
573     if ((!list_all_targets) &&
574         (report_dependencies_level == 0) &&
575         (dmake_mode_type == distributed_mode)) {
576         startup_rxm();
577     }
578 #endif
519
520 /*
521 * Enable interrupt handler for alarms
522 */
523     (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);
525 /*
526 * Check if make should report
527 */
528     if (getenv(sunpro_dependencies->string_mb) != NULL) {

```

```

529         FILE *report_file;
531         report_dependency("");
532         report_file = get_report_file();
533         if ((report_file != NULL) && (report_file != (FILE*)-1)) {
534             (void) fprintf(report_file, "\n");
535         }
536     }
538 /*
539 * Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
540 */
541     if (keep_state) {
542         maybe_append_prop(sunpro_dependencies, macro_prop)->
543             body.macro.exported = true;
544     } else {
545         maybe_append_prop(sunpro_dependencies, macro_prop)->
546             body.macro.exported = false;
547     }
549     working_on_targets = true;
550     if (trace_status) {
551         dump_make_state();
552         fclose(stdout);
553         fclose(stderr);
554         exit_status = 0;
555         exit(0);
556     }
557     if (list_all_targets) {
558         dump_target_list();
559         fclose(stdout);
560         fclose(stderr);
561         exit_status = 0;
562         exit(0);
563     }
564     trace_reader = false;
566     /*
567     * Set temp_file_directory to the directory the .make.state
568     * file is written to.
569     */
570     if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
571         temp_file_directory = strdup(get_current_path());
572     } else {
573         *slash_ptr = (int) nul_char;
574         (void) strcpy(make_state_dir, make_state->string_mb);
575         *slash_ptr = (int) slash_char;
576         /* when there is only one slash and it's the first
577         ** character, make_state_dir should point to '/'.
578         */
579         if (make_state_dir[0] == '\0') {
580             make_state_dir[0] = '/';
581             make_state_dir[1] = '\0';
582         }
583         if (make_state_dir[0] == (int) slash_char) {
584             temp_file_directory = strdup(make_state_dir);
585         } else {
586             char tmp_current_path2[MAXPATHLEN];
587
588             (void) sprintf(tmp_current_path2,
589                 "%s/%s",
590                 get_current_path(),
591                 make_state_dir);
592             temp_file_directory = strdup(tmp_current_path2);
593         }
594     }

```

```

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

597     report_dir_enter_leave(true);

599     make_targets(argc, argv, parallel_flag);

601     report_dir_enter_leave(false);

603     if (build_failed_ever_seen) {
604         if (posix) {
605             exit_status = 1;
606         }
607         exit(1);
608     }
609     exit_status = 0;
610     exit(0);
611     /* NOTREACHED */
612 }
unchanged portion omitted
682 #endif

747 /*
748 #ifdef DISTRIBUTED
749     if (get_parent() == TRUE) {
750 #endif
751 */

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

691             MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
692             failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
693             if ((exit_status != 0) && (failed_name->prop != NULL)) {
694 #ifdef TEAMWARE_MAKE_CMN
695                 /*
696                  * [tolik] switch DMake to serial mode
697                  */
698                 dmake_mode_type = serial_mode;
699                 no_parallel = true;
700 #endif
701                 (void) doname(failed_name, false, true);
702             } else {
703                 if (!trace_status) {
704 #ifdef TEAMWARE_MAKE_CMN
705                     /*
706                      * Switch DMake to serial mode
707                      */
708                     dmake_mode_type = serial_mode;
709                     no_parallel = true;
710 #endif
711                 }
712                 (void) doname(done, false, true);
713             }
714         }
715     }
716     /*
717     * Remove the temp file utilities report dependencies thru if it
718     * is still around
719     */

```

```

720     if (temp_file_name != NULL) {
721         (void) unlink(temp_file_name->string_mb);
722     }
723     /*
724     * Do not save the current command in .make.state if make
725     * was interrupted.
726     */
727     if (current_line != NULL) {
728         command_changed = true;
729         current_line->body.line.command_used = NULL;
730     }
731     /*
732     * For each parallel build process running, remove the temp files
733     * and zap the command line so it won't be put in .make.state
734     */
735     for (rp = running_list; rp != NULL; rp = rp->next) {
736         if (rp->temp_file != NULL) {
737             (void) unlink(rp->temp_file->string_mb);
738         }
739         if (rp->stdout_file != NULL) {
740             (void) unlink(rp->stdout_file);
741             retmem_mb(rp->stdout_file);
742             rp->stdout_file = NULL;
743         }
744         if (rp->stderr_file != NULL) {
745             (void) unlink(rp->stderr_file);
746             retmem_mb(rp->stderr_file);
747             rp->stderr_file = NULL;
748         }
749         command_changed = true;
750     /*
751     line = get_prop(rp->target->prop, line_prop);
752     if (line != NULL) {
753         line->body.line.command_used = NULL;
754     }
755     */
756     }
757     /* Remove the statefile lock file if the file has been locked */
758     if ((make_state_lockfile != NULL) && (make_state_locked)) {
759         (void) unlink(make_state_lockfile);
760         make_state_lockfile = NULL;
761         make_state_locked = false;
762     }
763     /* Write .make.state */
764     write_state_file(1, (Boolean) 1);
765     /*
766     #ifdef DISTRIBUTED
767     }
768     #endif
769     */

770 #ifdef TEAMWARE_MAKE_CMN
771     catclose(catd);
772 #endif
773 #ifdef DISTRIBUTED
774     if (rxmPid > 0) {
775         /* Tell rxm to exit by sending it an Avo_AcknowledgeMsg
776         Avo_AcknowledgeMsg acknowledgeMsg;
777         RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;
778     }
779     int xdrResult = xdr(&xdrs_out, msg);

```



```

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

864     waitpid(rxmPid, NULL, 0);
865     rxmPid = 0;
866 }
867 #endif
873 }

775 /*
776 *   handle_interrupt()
777 *
778 *   This is where C-C traps are caught.
779 *
780 *   Parameters:
781 *
782 *   Global variables used (except DMake 1.0):
783 *       current_target      Sometimes the current target is removed
784 *       do_not_exec_rule   But not if -n is on
785 *       quest               or -q
786 *       running_list       List of parallel running processes
787 *       touch               Current target is not removed if -t on
788 */
789 void
790 handle_interrupt(int)
791 {
792     Property      member;
793     Running       rp;

795     (void) fflush(stdout);
891 #ifndef DISTRIBUTED
892     if (rxmPid > 0) {
893         // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
894         Avo_AcknowledgeMsg acknowledgeMsg;
895         RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

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

899         if (xdrResult) {
900             fflush(dmake_ofp);
901         } else {
902             kill(rxmPid, SIGTERM);
903             rxmPid = 0;
904         }
905     }
906 #endif
926     if (childPid > 0) {
927         kill(childPid, SIGTERM);
928         childPid = -1;
929     }
930     for (rp = running_list; rp != NULL; rp = rp->next) {
931         if (rp->state != build_running) {
932             continue;
933         }
934         if (rp->pid > 0) {
935             kill(rp->pid, SIGTERM);
936             rp->pid = -1;
937         }
938     }
939     if (getpid() == getppid()) {

```

```

810         bsd_signal(SIGTERM, SIG_IGN);
811         kill (-getpid(), SIGTERM);
812     }
813 #ifndef TEAMWARE_MAKE_CMN
814     /* Clean up all parallel/distributed children already finished */
815     finish_children(false);
816 #endif

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

820     while (wait((int *) NULL) != -1);
821     /* Delete the current targets unless they are precious */
822     if ((current_target != NULL) &&
823         current_target->is_member &&
824         ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
825         current_target = member->body.member.library;
826     }
827     if (!do_not_exec_rule &&
828         !touch &&
829         !quest &&
830         (current_target != NULL) &&
831         !(current_target->stat.is_precious || all_precious)) {

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

837     if (exists(current_target) != file_doesnt_exist) {
838         (void) fprintf(stderr,
839             "\n*** %s ",
840             current_target->string_mb);
841         if (current_target->stat.is_dir) {
842             (void) fprintf(stderr,
843                 catgets(catd, 1, 168, "not remove
844                 current_target->string_mb);
845             } else if (unlink(current_target->string_mb) == 0) {
846                 (void) fprintf(stderr,
847                     catgets(catd, 1, 169, "removed.\n
848                     current_target->string_mb);
849             } else {
850                 (void) fprintf(stderr,
851                     catgets(catd, 1, 170, "could not
852                     current_target->string_mb,
853                     errmsg(errno));
854             }
855         }
856     }
857     for (rp = running_list; rp != NULL; rp = rp->next) {
858         if (rp->state != build_running) {
859             continue;
860         }
861         if (rp->target->is_member &&
862             ((member = get_prop(rp->target->prop, member_prop)) !=
863              NULL)) {
864             rp->target = member->body.member.library;
865         }
866         if (!do_not_exec_rule &&
867             !touch &&
868             !quest &&
869             !(rp->target->stat.is_precious || all_precious)) {

871             rp->target->stat.time = file_no_time;
872             if (exists(rp->target) != file_doesnt_exist) {
873                 (void) fprintf(stderr,
874                     "\n*** %s ",
875                     rp->target->string_mb);

```

```

876     if (rp->target->stat.is_dir) {
877         (void) fprintf(stderr,
878             catgets(catd, 1, 171, "no
879             rp->target->string_mb);
880     } else if (unlink(rp->target->string_mb) == 0) {
881         (void) fprintf(stderr,
882             catgets(catd, 1, 172, "re
883             rp->target->string_mb);
884     } else {
885         (void) fprintf(stderr,
886             catgets(catd, 1, 173, "co
887             rp->target->string_mb,
888             errmsg(errno));
889     }
890 }
891 }
892 }

```

```

895 /* Have we locked .make.state or .nse_depinfo? */
896 if ((make_state_lockfile != NULL) && (make_state_locked)) {
897     unlink(make_state_lockfile);
898     make_state_lockfile = NULL;
899     make_state_locked = false;
900 }
901 /*
902  * Re-read .make.state file (it might be changed by recursive make)
903  */
904 check_state(NULL);

906 report_dir_enter_leave(false);

908 exit_status = 2;
909 exit(2);
910 }

```

unchanged portion omitted

```

1353 /*
1354 * parse_command_option(ch)
1355 *
1356 * Parse make command line options.
1357 *
1358 * Return value:
1359 *
1360 * Indicates if any -f -c or -M were seen
1361 *
1362 * Parameters:
1363 *     ch          The character to parse
1364 *
1365 * Static variables used:
1366 *     dmake_group_specified  Set for make -g
1367 *     dmake_max_jobs_specified  Set for make -j
1368 *     dmake_mode_specified  Set for make -m
1369 *     dmake_add_mode_specified  Set for make -x
1370 *     dmake_compat_mode_specified  Set for make -x SUN_MAKE_COMPAT_
1371 *     dmake_output_mode_specified  Set for make -x DMAKE_OUTPUT_MOD
1372 *     dmake_odir_specified  Set for make -o
1373 *     dmake_rcfile_specified  Set for make -c
1374 *     env_wins  Set for make -e
1375 *     ignore_default_mk  Set for make -r
1376 *     trace_status  Set for make -p
1377 *
1378 * Global variables used:
1379 *     .make.state path & name  set for make -K
1380 *     continue_after_error  Set for make -k
1381 *     debug_level  Set for make -d
1382 *     do_not_exec_rule  Set for make -n

```

```

1382 *     filter_stderr  Set for make -X
1383 *     ignore_errors_all  Set for make -i
1384 *     no_parallel  Set for make -R
1385 *     quest  Set for make -q
1386 *     read_trace_level  Set for make -D
1387 *     report_dependencies  Set for make -P
1388 *     send_mtool_msgs  Set for make -K
1389 *     silent_all  Set for make -s
1390 *     touch  Set for make -t
1391 */
1392 static int
1393 parse_command_option(register char ch)
1394 {
1395     static int          invert_next = 0;
1396     int                invert_this = invert_next;

1398     invert_next = 0;
1399     switch (ch) {
1400     case '-': /* Ignore "--" */
1401         return 0;
1402     case '~': /* Invert next option */
1403         invert_next = 1;
1404         return 0;
1405     case 'B': /* Obsolete */
1406         return 0;
1407     case 'b': /* Obsolete */
1408         return 0;
1409     case 'c': /* Read alternative dmake.rc file */
1410         if (invert_this) {
1411             dmake_rcfile_specified = false;
1412         } else {
1413             dmake_rcfile_specified = true;
1414         }
1415         return 2;
1416     case 'D': /* Show lines read */
1417         if (invert_this) {
1418             read_trace_level--;
1419         } else {
1420             read_trace_level++;
1421         }
1422         return 0;
1423     case 'd': /* Debug flag */
1424         if (invert_this) {
1425             debug_level--;
1426         } else {
1427             debug_level++;
1428         }
1429         return 0;
1430     case 'e': /* Environment override flag */
1431         if (invert_this) {
1432             env_wins = false;
1433         } else {
1434             env_wins = true;
1435         }
1436         return 0;
1437     case 'f': /* Read alternative makefile(s) */
1438         return 1;
1439     case 'g': /* Use alternative DMake group */
1440         if (invert_this) {
1441             dmake_group_specified = false;
1442         } else {
1443             dmake_group_specified = true;
1444         }
1445         return 4;
1446     case 'i': /* Ignore errors */
1447         if (invert_this) {

```

```

1448         ignore_errors_all = false;
1449     } else {
1450         ignore_errors_all = true;
1451     }
1452     return 0;
1453 case 'j':
1454     /* Use alternative DMake max jobs */
1455     if (invert_this) {
1456         dmake_max_jobs_specified = false;
1457     } else {
1458         dmake_max_jobs_specified = true;
1459     }
1460     return 8;
1461 case 'K':
1462     /* Read alternative .make.state */
1463     return 256;
1464 case 'k':
1465     /* Keep making even after errors */
1466     if (invert_this) {
1467         continue_after_error = false;
1468     } else {
1469         continue_after_error = true;
1470         continue_after_error_ever_seen = true;
1471     }
1472     return 0;
1473 case 'M':
1474     /* Read alternative make.machines file */
1475     if (invert_this) {
1476         pmake_machinesfile_specified = false;
1477     } else {
1478         pmake_machinesfile_specified = true;
1479         dmake_mode_type = parallel_mode;
1480         no_parallel = false;
1481     }
1482     return 16;
1483 case 'm':
1484     /* Use alternative DMake build mode */
1485     if (invert_this) {
1486         dmake_mode_specified = false;
1487     } else {
1488         dmake_mode_specified = true;
1489     }
1490     return 32;
1491 case 'x':
1492     /* Use alternative DMake mode */
1493     if (invert_this) {
1494         dmake_add_mode_specified = false;
1495     } else {
1496         dmake_add_mode_specified = true;
1497     }
1498     return 1024;
1499 case 'N':
1500     /* Reverse -n */
1501     if (invert_this) {
1502         do_not_exec_rule = true;
1503     } else {
1504         do_not_exec_rule = false;
1505     }
1506     return 0;
1507 case 'n':
1508     /* Print, not exec commands */
1509     if (invert_this) {
1510         do_not_exec_rule = false;
1511     } else {
1512         do_not_exec_rule = true;
1513     }
1514     return 0;
1515 case 'O':
1516     /* Send job start & result msgs */
1517     if (invert_this) {
1518         send_mtool_msgs = false;
1519     } else {
1520         send_mtool_msgs = true;
1521     }
1522 #ifdef DISTRIBUTED
1523     send_mtool_msgs = true;
1524 #endif

```

```

1511     }
1512     return 128;
1513 case 'o':
1514     /* Use alternative dmake output dir */
1515     if (invert_this) {
1516         dmake_odir_specified = false;
1517     } else {
1518         dmake_odir_specified = true;
1519     }
1520     return 512;
1521 case 'P':
1522     /* Print for selected targets */
1523     if (invert_this) {
1524         report_dependencies_level--;
1525     } else {
1526         report_dependencies_level++;
1527     }
1528     return 0;
1529 case 'p':
1530     /* Print description */
1531     if (invert_this) {
1532         trace_status = false;
1533         do_not_exec_rule = false;
1534     } else {
1535         trace_status = true;
1536         do_not_exec_rule = true;
1537     }
1538     return 0;
1539 case 'q':
1540     /* Question flag */
1541     if (invert_this) {
1542         quest = false;
1543     } else {
1544         quest = true;
1545     }
1546     return 0;
1547 case 'R':
1548     /* Don't run in parallel */
1549 #ifdef TEAMWARE_MAKE_CMN
1550     if (invert_this) {
1551         pmake_cap_r_specified = false;
1552         no_parallel = false;
1553     } else {
1554         pmake_cap_r_specified = true;
1555         dmake_mode_type = serial_mode;
1556         no_parallel = true;
1557     }
1558 #else
1559     warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option"))
1560 #endif
1561     return 0;
1562 case 'r':
1563     /* Turn off internal rules */
1564     if (invert_this) {
1565         ignore_default_mk = false;
1566     } else {
1567         ignore_default_mk = true;
1568     }
1569     return 0;
1570 case 'S':
1571     /* Reverse -k */
1572     if (invert_this) {
1573         continue_after_error = true;
1574     } else {
1575         continue_after_error = false;
1576         stop_after_error_ever_seen = true;
1577     }
1578     return 0;
1579 case 's':
1580     /* Silent flag */
1581     if (invert_this) {
1582         silent_all = false;
1583     } else {
1584         silent_all = true;
1585     }

```

```

1577     }
1578     return 0;
1579     case 'T':                /* Print target list */
1580         if (invert_this) {
1581             list_all_targets = false;
1582             do_not_exec_rule = false;
1583         } else {
1584             list_all_targets = true;
1585             do_not_exec_rule = true;
1586         }
1587     return 0;
1588     case 't':                /* Touch flag */
1589         if (invert_this) {
1590             touch = false;
1591         } else {
1592             touch = true;
1593         }
1594     return 0;
1595     case 'u':                /* Unconditional flag */
1596         if (invert_this) {
1597             build_unconditional = false;
1598         } else {
1599             build_unconditional = true;
1600         }
1601     return 0;
1602     case 'V':                /* SVR4 mode */
1603         svr4 = true;
1604         return 0;
1605     case 'v':                /* Version flag */
1606         if (invert_this) {
1607             } else {
1608 #ifdef TEAMWARE_MAKE_CMN
1609             fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1610             exit_status = 0;
1611             exit(0);
1612 #else
1613             warning(catgets(catd, 1, 324, "Ignoring DistributedMake
1614 #endif
1615         }
1616     return 0;
1617     case 'w':                /* Unconditional flag */
1618         if (invert_this) {
1619             report_cwd = false;
1620         } else {
1621             report_cwd = true;
1622         }
1623     return 0;
1624 #if 0
1625     case 'X':                /* Filter stdout */
1626         if (invert_this) {
1627             filter_stderr = false;
1628         } else {
1629             filter_stderr = true;
1630         }
1631     return 0;
1632 #endif
1633     default:
1634         break;
1635     }
1636     return 0;
1637 }
1638
1639 unchanged_portion_omitted
1640 #endif
1641
1642 #ifdef DISTRIBUTED
1643 /*

```

```

3414 * Returns whether -c is set or not.
3415 */
3416 Boolean
3417 get_dmake_rcfile_specified(void)
3418 {
3419     return(dmake_rcfile_specified);
3420 }
3421
3422 /*
3423 * Returns whether -g is set or not.
3424 */
3425 Boolean
3426 get_dmake_group_specified(void)
3427 {
3428     return(dmake_group_specified);
3429 }
3430
3431 /*
3432 * Returns whether -j is set or not.
3433 */
3434 Boolean
3435 get_dmake_max_jobs_specified(void)
3436 {
3437     return(dmake_max_jobs_specified);
3438 }
3439
3440 /*
3441 * Returns whether -m is set or not.
3442 */
3443 Boolean
3444 get_dmake_mode_specified(void)
3445 {
3446     return(dmake_mode_specified);
3447 }
3448
3449 /*
3450 * Returns whether -o is set or not.
3451 */
3452 Boolean
3453 get_dmake_odir_specified(void)
3454 {
3455     return(dmake_odir_specified);
3456 }
3457
3458 #endif
3459
3460 static void
3461 report_dir_enter_leave(Boolean entering)
3462 {
3463     char    rcwd[MAXPATHLEN];
3464     static char * mlev = NULL;
3465     char *  make_level_str = NULL;
3466     int     make_level_val = 0;
3467
3468     make_level_str = getenv(NOCATGETS("MAKELEVEL"));
3469     if(make_level_str) {
3470         make_level_val = atoi(make_level_str);
3471     }
3472     if(mlev == NULL) {
3473         mlev = (char*) malloc(MAXPATHLEN);
3474     }
3475     if(entering) {
3476         sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val + 1);
3477     } else {
3478         make_level_val--;
3479         sprintf(mlev, NOCATGETS("MAKELEVEL=%d"), make_level_val);
3480     }

```

```
3319     }
3320     putenv(mlev);
3321
3322     if(report_cwd) {
3323         if(make_level_val <= 0) {
3324             if(entering) {
3325 #ifdef TEAMWARE_MAKE_CMN
3326                 sprintf( rcwd
3327                     , catgets(catd, 1, 329, "dmake: Entering
3328                     , get_current_path());
3329 #else
3330                 sprintf( rcwd
3331                     , catgets(catd, 1, 330, "make: Entering d
3332                     , get_current_path());
3333 #endif
3334             } else {
3335 #ifdef TEAMWARE_MAKE_CMN
3336                 sprintf( rcwd
3337                     , catgets(catd, 1, 331, "dmake: Leaving d
3338                     , get_current_path());
3339 #else
3340                 sprintf( rcwd
3341                     , catgets(catd, 1, 332, "make: Leaving di
3342                     , get_current_path());
3343 #endif
3344             }
3345         } else {
3346             if(entering) {
3347 #ifdef TEAMWARE_MAKE_CMN
3348                 sprintf( rcwd
3349                     , catgets(catd, 1, 333, "dmake[%d]: Enter
3350                     , make_level_val, get_current_path());
3351 #else
3352                 sprintf( rcwd
3353                     , catgets(catd, 1, 334, "make[%d]: Enteri
3354                     , make_level_val, get_current_path());
3355 #endif
3356             } else {
3357 #ifdef TEAMWARE_MAKE_CMN
3358                 sprintf( rcwd
3359                     , catgets(catd, 1, 335, "dmake[%d]: Leavi
3360                     , make_level_val, get_current_path());
3361 #else
3362                 sprintf( rcwd
3363                     , catgets(catd, 1, 336, "make[%d]: Leavin
3364                     , make_level_val, get_current_path());
3365 #endif
3366             }
3367         }
3368         printf(NOCATGETS("%s"), rcwd);
3369     }
3370 }
unchanged_portion_omitted_
```

```

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

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

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

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

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

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

```

```

164         (parallel_process_cnt > 0)) {
165             (void) fprintf(stderr,
166                 catgets(catd, 1, 157, "Waiting for %d %s to finish
167                 parallel_process_cnt,
168                 parallel_process_cnt == 1 ?
169                 catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
170                 (void) fflush(stderr);
171         }

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

187 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
188     job_adjust_fini();
189 #endif

191     exit_status = 1;
192     exit(1);
193 }

195 /*
196 *   warning(format, args...)
197 *
198 *   Print a message and continue.
199 *
200 *   Parameters:
201 *       format      printf type format string
202 *       args        Arguments to match the format
203 *
204 *   Global variables used:
205 *       report_pwd      Should we report the current path?
206 */
207 /*VARARGS*/
208 void
209 warning(char * message, ...)
210 {
211     va_list args;

213     va_start(args, message);
214     (void) fflush(stdout);
215 #ifdef DISTRIBUTED
216     (void) fprintf(stderr, catgets(catd, 1, 264, "dmake: Warning: "));
217 #else
218     (void) fprintf(stderr, catgets(catd, 1, 265, "make: Warning: "));
219 #endif
220     (void) vfprintf(stderr, message, args);
221     (void) fprintf(stderr, "\n");
222     va_end(args);
223     if (report_pwd) {
224         (void) fprintf(stderr,
225             catgets(catd, 1, 161, "Current working directory
226             get_current_path());
227     }
228     (void) fflush(stderr);
229 }
_unchanged_portion_omitted_

```

```
251 /*
252 *   get_current_path()
253 *
254 *   Stuff current_path with the current path if it isnt there already.
255 *
256 *   Parameters:
257 *
258 *   Global variables used:
259 */
260 char *
261 get_current_path(void)
262 {
263     char          pwd[(MAXPATHLEN * MB_LEN_MAX)];
264     static char   *current_path;
265
266     if (current_path == NULL) {
267         getcwd(pwd, sizeof(pwd));
268         if (pwd[0] == (int) nul_char) {
269             pwd[0] = (int) slash_char;
270             pwd[1] = (int) nul_char;
271
272 #ifdef DISTRIBUTED
273             current_path = strdup(pwd);
274         } else if (IS_EQUALN(pwd, NOCATGETS("/tmp_mnt"), 8)) {
275             current_path = strdup(pwd + 8);
276         } else {
277             current_path = strdup(pwd);
278         }
279 #else
280         current_path = strdup(pwd);
281 #endif
282     }
283     return current_path;
284 }
285
286 unchanged_portion_omitted
```

new/usr/src/cmd/make/bin/parallel.cc

1

46861 Wed May 20 11:55:56 2015

new/usr/src/cmd/make/bin/parallel.cc

make: unifdef for MAKE TOOL and DISTRIBUTED (undefined)

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
```

```
26 #ifdef TEAMWARE_MAKE_CMN
```

```
28 /*
29  * parallel.cc
30  *
31  * Deal with the parallel processing
32  */
```

```
34 /*
35  * Included files
36  */
```

```
37 #ifndef DISTRIBUTED
38 #include <avo/strings.h> /* AVO_STRDUP() */
39 #include <dm/Avo_DoJobMsg.h>
40 #include <dm/Avo_MToolJobResultMsg.h>
41 #endif
42 #include <errno.h> /* errno */
43 #include <fcntl.h>
44 #include <mk/defs.h>
45 #include <mksh/dosys.h> /* redirect_io() */
46 #include <mksh/macro.h> /* expand_value() */
47 #include <mksh/misc.h> /* getmem() */
48 #include <sys/signal.h>
49 #include <sys/stat.h>
50 #include <sys/types.h>
51 #include <sys/utsname.h>
52 #include <sys/wait.h>
53 #include <unistd.h>
54 #include <netdb.h>
```

```
53 /*
54  * Defined macros
55  */
```

```
56 #define MAXRULES 100
```

new/usr/src/cmd/make/bin/parallel.cc

2

```
58 /*
59  * This const should be in avo_dms/include/AvoDmakeCommand.h
60  */
61 const int local_host_mask = 0x20;
```

```
64 /*
65  * typedefs & structs
66  */
```

```
69 /*
70  * Static variables
71  */
72 #ifdef TEAMWARE_MAKE_CMN
73 static Boolean just_did_subtree = false;
74 static char local_host[MAXNAMELEN] = "";
75 static char user_name[MAXNAMELEN] = "";
76 #endif
77 static int pmake_max_jobs = 0;
78 static pid_t process_running = -1;
79 static Running *running_tail = &running_list;
80 static Name subtree_conflict;
81 static Name subtree_conflict2;
```

```
84 /*
85  * File table of contents
86  */
```

```
92 #ifdef DISTRIBUTED
93 static void append_dmake_cmd(Avo_DoJobMsg *dmake_job_msg, char *orig
94 static void append_job_result_msg(Avo_MToolJobResultMsg *msg, char *
95 static void send_job_result_msg(Running rp);
96 #endif
97 static void delete_running_struct(Running rp);
98 static Boolean dependency_conflict(Name target);
99 static Doname distribute_process(char **commands, Property line);
100 static void doname_subtree(Name target, Boolean do_get, Boolean impl
101 static void dump_out_file(char *filename, Boolean err);
102 static void finish_doname(Running rp);
103 static void maybe_reread_make_state(void);
104 static void process_next(void);
105 static void reset_conditionals(int cnt, Name *targets, Property *loc
106 static pid_t run_rule_commands(char *host, char **commands);
107 static Property *set_conditionals(int cnt, Name *targets);
108 static void store_conditionals(Running rp);
```

```
101 /*
102  * execute_parallel(line, waitflg)
103  *
104  * DMake 2.x:
105  * parallel mode: spawns a parallel process to execute the command group.
106  * distributed mode: sends the command group down the pipe to rxm.
107  *
108  * Return value:
109  * The result of the execution
110  *
111  * Parameters:
112  * line The command group to execute
113  */
114 Doname
115 execute_parallel(Property line, Boolean waitflg, Boolean local)
116 {
117     int argcnt;
```



```

118     int          cmd_options = 0;
119     char         *commands[MAXRULES + 5];
120     char         *cp;
131 #ifndef DISTRIBUTED
132     Avo_DoJobMsg *dmake_job_msg = NULL;
133 #endif
121     Name          dmake_name;
122     Name          dmake_value;
123     int          ignore;
124     Name          make_machines_name;
125     char         **p;
126     Property     prop;
127     Doname       result = build_ok;
128     Cmd_line     rule;
129     Boolean      silent_flag;
130     Name         target = line->body.line.target;
131     Boolean      wrote_state_file = false;

133     if ((pmake_max_jobs == 0) &&
134         (dmake_mode_type == parallel_mode)) {
135         if (local_host[0] == '\0') {
136             (void) gethostname(local_host, MAXNAMELEN);
137         }
138         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
139         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
140         if ((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
141             ((dmake_value = prop->body.macro.value) != NULL) {
142             pmake_max_jobs = atoi(dmake_value->string_mb);
143             if (pmake_max_jobs <= 0) {
144                 warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
145 warning(catgets(catd, 1, 309, "setting DMAKE_MAX
146 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
147         }
148     } else {
149         /*
150          * For backwards compatibility w/ PMake 1.x, when
151          * DMake 2.x is being run in parallel mode, DMake
152          * should parse the PMake startup file
153          * $(HOME)/.make.machines to get the pmake_max_jobs.
154          */
155         MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
156         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
157         if ((prop = get_prop(dmake_name->prop, macro_prop)) !=
158             ((dmake_value = prop->body.macro.value) != NULL)) {
159             make_machines_name = dmake_value;
160         } else {
161             make_machines_name = NULL;
162         }
163         if ((pmake_max_jobs = read_make_machines(make_machines_n
164 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
165     }
166 }
180 #ifndef DISTRIBUTED
181     if (send_mtool_msgs) {
182         send_rsrc_info_msg(pmake_max_jobs, local_host, user_name
183     }
184 #endif
167 }

169     if ((dmake_mode_type == serial_mode) ||
170         ((dmake_mode_type == parallel_mode) && (waitflg))) {
171         return (execute_serial(line));
172     }

192 #ifndef DISTRIBUTED
193     if (dmake_mode_type == distributed_mode) {

```

```

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

178     argcnt = 0;
179     for (rule = line->body.line.command_used;
180         rule != NULL;
181         rule = rule->next) {
182         if (posix && (touch || quest) && !rule->always_exec) {
183             continue;
184         }
185         if (vpath_defined) {
186             rule->command_line =
187                 vpath_translation(rule->command_line);
188         }
189         if (dmake_mode_type == distributed_mode) {
190             cmd_options = 0;
191             if (local) {
192                 cmd_options |= local_host_mask;
193             }
194         } else {
195             silent_flag = false;
196             ignore = 0;
197         }
198         if (rule->command_line->hash.length > 0) {
199             if (++argcnt == MAXRULES) {
200                 if (dmake_mode_type == distributed_mode) {
201                     /* XXX - tell rxm to execute on local ho
202                     /* I WAS HERE!!! */
203                 } else {
204                     /* Too many rules, run serially instead.
205                     return build_serial;
206                 }
207             }
208         }
209 #ifndef DISTRIBUTED
210         if (dmake_mode_type == distributed_mode) {
211             /*
212             * XXX - set assign_mask to tell rxm
213             * to do the following.
214             */
215             /* From execute_serial():
216             if (rule->assign) {
217                 result = build_ok;
218                 do_assign(rule->command_line, target);
219             }
220             */
221             if (0) {
222             } else if (report_dependencies_level == 0) {
223                 if (rule->ignore_error) {
224                     cmd_options |= ignore_mask;
225                 }
226                 if (rule->silent) {
227                     cmd_options |= silent_mask;
228                 }
229                 if (rule->command_line->meta) {
230                     cmd_options |= meta_mask;

```

```

260     }
261     if (rule->make_refd) {
262         cmd_options |= make_refd_mask;
263     }
264     if (do_not_exec_rule) {
265         cmd_options |= do_not_exec_mask;
266     }
267     append_dmake_cmd(dmake_job_msg,
268                     rule->command_line->str
269                     cmd_options);
270     /* Copying dosys()... */
271     if (rule->make_refd) {
272         if (waitflg) {
273             dmake_job_msg->setImmedi
274         }
275         called_make = true;
276         if (command_changed &&
277             !wrote_state_file) {
278             write_state_file(0, fals
279             wrote_state_file = true;
280         }
281     }
282     } else
283 }
284 #endif
285 {
286     if (rule->silent && !silent) {
287         silent_flag = true;
288     }
289     if (rule->ignore_error) {
290         ignore++;
291     }
292     /* XXX - need to add support for + prefix */
293     if (silent_flag || ignore) {
294         *p = getmem((silent_flag ? 1 : 0) +
295                  ignore +
296                  (strlen(rule->
297                       command_line->
298                       string_mb) +
299                  1));
300         cp = *p++;
301         if (silent_flag) {
302             *cp++ = (int) at_char;
303         }
304         if (ignore) {
305             *cp++ = (int) hyphen_char;
306         }
307         (void) strcpy(cp, rule->command_line->st
308     } else {
309         *p++ = rule->command_line->string_mb;
310     }
311 }
312 }
313 }
314 }
315 }
316 #ifndef DISTRIBUTED
317     if (dmake_job_msg) {
318         delete dmake_job_msg;
319     }
320 #endif
321     return build_ok;
322 }
323 #ifndef DISTRIBUTED
324     if (dmake_mode_type == distributed_mode) {
325         // Send a DoJob message to the rxm process.

```

```

326     distribute_rxm(dmake_job_msg);
327 }
328 // Wait for an acknowledgement.
329 Avo_AcknowledgeMsg *ackMsg = getAcknowledgeMsg();
330 if (ackMsg) {
331     delete ackMsg;
332 }
333 }
334 if (waitflg) {
335     // Wait for, and process a job result.
336     result = await_dist(waitflg);
337     if (called_make) {
338         maybe_reread_make_state();
339     }
340     check_state(temp_file_name);
341     if (result == build_failed) {
342         if (!continue_after_error) {
343             return result;
344 #ifndef PRINT_EXIT_STATUS
345             warning(NOCATGETS("I'm in execute_parall
346 #endif
347
348             fatal(catgets(catd, 1, 252, "Command fai
349                 target->string_mb);
350         }
351     /*
352     * Make sure a failing command is not
353     * saved in .make.state.
354     */
355     line->body.line.command_used = NULL;
356 }
357 if (temp_file_name != NULL) {
358     free_name(temp_file_name);
359 }
360 temp_file_name = NULL;
361 Property spro = get_prop(sunpro_dependencies->prop, macr
362 if(spro != NULL) {
363     Name val = spro->body.macro.value;
364     if(val != NULL) {
365         free_name(val);
366         spro->body.macro.value = NULL;
367     }
368 }
369 spro = get_prop(sunpro_dependencies->prop, env_mem_prop)
370 if(spro) {
371     char *val = spro->body.env_mem.value;
372     if(val != NULL) {
373         retmem_mb(val);
374         spro->body.env_mem.value = NULL;
375     }
376 }
377 return result;
378 } else {
379     parallel_process_cnt++;
380     return build_running;
381 }
382 } else
383 #endif
384 {
385     *p = NULL;
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

249     /*
250     * Return only those memory that were specially allocated
251     * for part of commands.
252     */
253     for (int i = 0; commands[i] != NULL; i++) {
254         if ((commands[i][0] == (int) at_char) ||
255             (commands[i][0] == (int) hyphen_char)) {
256             retmem_mb(commands[i]);
257         }
258     }
259     return res;
260 }
261 }

406 #ifdef DISTRIBUTED
407 /*
408 *   append_dmake_cmd()
409 *
410 *   Replaces all escaped newline's (\<cr>)
411 *   in the original command line with space's,
412 *   then append the new command line to the DoJobMsg object.
413 */
414 static void
415 append_dmake_cmd(Avo_DoJobMsg *dmake_job_msg,
416                 char *orig_cmd_line,
417                 int cmd_options)
418 {
419 /*
420     Avo_DmakeCommand      *tmp_dmake_command;

422     tmp_dmake_command = new Avo_DmakeCommand(orig_cmd_line, cmd_options);
423     dmake_job_msg->appendCmd(tmp_dmake_command);
424     delete tmp_dmake_command;
425 */
426     dmake_job_msg->appendCmd(new Avo_DmakeCommand(orig_cmd_line, cmd_options
427 )
428 #endif

264 #ifdef TEAMWARE_MAKE_CMN
265 #define MAXJOBS_ADJUST_RFE4694000

267 #ifdef MAXJOBS_ADJUST_RFE4694000

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

275 /*
276 *   adjust_pmake_max_jobs (int pmake_max_jobs)
277 *
278 *   Parameters:
279 *       pmake_max_jobs - max jobs limit set by user
280 *
281 *   External functions used:
282 *       sysconf()
283 *       getloadavg()
284 */
285 static int
286 adjust_pmake_max_jobs (int pmake_max_jobs)
287 {
288     static int      ncpu = 0;
289     double          loadavg[3];
290     int             adjustment;
291     int             adjusted_max_jobs;

```

```

293     if (ncpu <= 0) {
294         if ((ncpu = sysconf(_SC_NPROCESSORS_ONLN)) <= 0) {
295             ncpu = 1;
296         }
297     }
298     if (getloadavg(loadavg, 3) != 3) return(pmake_max_jobs);
299     adjustment = ((int)loadavg[LOADAVG_1MIN]);
300     if (adjustment < 2) return(pmake_max_jobs);
301     if (ncpu > 1) {
302         adjustment = adjustment / ncpu;
303     }
304     adjusted_max_jobs = pmake_max_jobs - adjustment;
305     if (adjusted_max_jobs < 1) adjusted_max_jobs = 1;
306     return(adjusted_max_jobs);
307 }

unchanged_portion_omitted_

553 #endif /* MAXJOBS_ADJUST_RFE4694000 */
554 #endif /* TEAMWARE_MAKE_CMN */

556 /*
557 *   distribute_process(char **commands, Property line)
558 *
559 *   Parameters:
560 *       commands          argv vector of commands to execute
561 *
562 *   Return value:
563 *
564 *       The result of the execution

565 *   Static variables used:
566 *       process_running Set to the pid of the process set running
567 * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
568 *       job_adjust_mode Current job adjust mode
569 * #endif
570 */
571 static Doname
572 distribute_process(char **commands, Property line)
573 {
574     static unsigned file_number = 0;
575     wchar_t         string[MAXPATHLEN];
576     char            mbstring[MAXPATHLEN];
577     int             filed;
578     int             res;
579     int             tmp_index;
580     char            *tmp_index_str_ptr;

582 #if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
583     while (parallel_process_cnt >= pmake_max_jobs) {
584         await_parallel(false);
585         finish_children(true);
586     }
587 #else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
588     /* initialize adjust mode, if not initialized */
589     if (job_adjust_mode == ADJUST_UNKNOWN) {
590         job_adjust_init();
591     }

593     /* actions depend on adjust mode */
594     switch (job_adjust_mode) {
595     case ADJUST_M1:
596         while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
597             await_parallel(false);
598             finish_children(true);
599         }
600     break;

```

```

601     case ADJUST_M2:
602         if ((res = m2_acquire_job()) == 0) {
603             if (parallel_process_cnt > 0) {
604                 await_parallel(false);
605                 finish_children(true);
606
607                 if ((res = m2_acquire_job()) == 0) {
608                     return build_serial;
609                 }
610             } else {
611                 return build_serial;
612             }
613         }
614         if (res < 0) {
615             /* job adjustment error */
616             job_adjust_error();
617
618             /* no adjustment */
619             while (parallel_process_cnt >= pmake_max_jobs) {
620                 await_parallel(false);
621                 finish_children(true);
622             }
623         }
624         break;
625     default:
626         while (parallel_process_cnt >= pmake_max_jobs) {
627             await_parallel(false);
628             finish_children(true);
629         }
630     }
631 #endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
632 #ifdef DISTRIBUTED
633     if (send_mtool_msgs) {
634         send_job_start_msg(line);
635     }
636 #endif
637 #ifdef DISTRIBUTED
638     setvar_envvar((Avo_DoJobMsg *)NULL);
639 #else
640     setvar_envvar();
641 #endif
642
643 /*
644  * Tell the user what DMake is doing.
645  */
646 if (!silent && output_mode != txt2_mode) {
647     /*
648      * Print local_host --> x job(s).
649      */
650     (void) fprintf(stdout,
651         catgets(catd, 1, 325, "%s --> %d %s\n"),
652         local_host,
653         parallel_process_cnt + 1,
654         (parallel_process_cnt == 0) ? catgets(catd, 1, 12
655
656 /* Print command line(s). */
657 tmp_index = 0;
658 while (commands[tmp_index] != NULL) {
659     /* No @ char. */
660     /* XXX - need to add [2] when + prefix is added */
661     if ((commands[tmp_index][0] != (int) at_char) &&
662         (commands[tmp_index][1] != (int) at_char)) {
663         tmp_index_str_ptr = commands[tmp_index];
664         if (*tmp_index_str_ptr == (int) hyphen_char) {
665             tmp_index_str_ptr++;
666         }
667         (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);

```

```

658     }
659     tmp_index++;
660 }
661 (void) fflush(stdout);
662 }
663
664 (void) sprintf(mbstring,
665     NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
666     tmpdir,
667     getpid(),
668     file_number++);
669
670 mktemp(mbstring);
671
672 stdout_file = strdup(mbstring);
673 stderr_file = NULL;
674 #if defined (TEAMWARE_MAKE_CMN)
675 if (!out_err_same) {
676     (void) sprintf(mbstring,
677         NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
678         tmpdir,
679         getpid(),
680         file_number++);
681
682     mktemp(mbstring);
683
684     stderr_file = strdup(mbstring);
685 }
686 #endif
687
688 process_running = run_rule_commands(local_host, commands);
689
690 return build_running;
691 }

```

unchanged portion omitted

```

759 /*
760 * finish_running()
761 *
762 * Keeps processing until the running_list is emptied out.
763 *
764 * Parameters:
765 *
766 * Global variables used:
767 *     running_list     The list of running processes
768 */
769 void
770 finish_running(void)
771 {
772     while (running_list != NULL) {
773         #ifdef DISTRIBUTED
774             if (dmake_mode_type == distributed_mode) {
775                 if ((just_did_subtree) ||
776                     (parallel_process_cnt == 0)) {
777                     just_did_subtree = false;
778                     (void) await_dist(false);
779                     finish_children(true);
780                 } else {
781                 }
782             } else {
783             #endif
784             {
785                 await_parallel(false);
786                 finish_children(true);
787             }
788             if (running_list != NULL) {

```

```

778         process_next();
779     }
780 }
781 }

783 /*
784 * process_next()
785 *
786 * Searches the running list for any targets which can start processing.
787 * This can be a pending target, a serial target, or a subtree target.
788 *
789 * Parameters:
790 *
791 * Static variables used:
792 *     running_tail      The end of the list of running procs
793 *     subtree_conflict  A target which conflicts with a subtree
794 *     subtree_conflict2 The other target which conflicts
795 *
796 * Global variables used:
797 *     commands_done     True if commands executed
798 *     debug_level       Controls debug output
799 *     parallel_process_cnt Number of parallel process running
800 *     recursion_level   Indentation for debug output
801 *     running_list      List of running processes
802 */
803 static void
804 process_next(void)
805 {
806     Running      rp;
807     Running      *rp_prev;
808     Property     line;
809     Chain        target_group;
810     Dependency    dep;
811     Boolean      quiescent = true;
812     Running      *subtree_target;
813     Boolean      saved_commands_done;
814     Property     *conditionals;

816     subtree_target = NULL;
817     subtree_conflict = NULL;
818     subtree_conflict2 = NULL;
819     /*
820     * If nothing currently running, build a serial target, if any.
821     */
822 start_loop_1:
823     for (rp_prev = &running_list, rp = running_list;
824          rp != NULL && parallel_process_cnt == 0;
825          rp = rp->next) {
826         if (rp->state == build_serial) {
827             *rp_prev = rp->next;
828             if (rp->next == NULL) {
829                 running_tail = rp_prev;
830             }
831             recursion_level = rp->recursion_level;
832             rp->target->state = build_pending;
833             (void) doname_check(rp->target,
834                                rp->do_get,
835                                rp->implicit,
836                                false);
837             quiescent = false;
838             delete_running_struct(rp);
839             goto start_loop_1;
840         } else {
841             rp_prev = &rp->next;
842         }
843     }

```

```

844     /*
845     * Find a target to build. The target must be pending, have all
846     * its dependencies built, and not be in a target group with a target
847     * currently building.
848     */
849 start_loop_2:
850     for (rp_prev = &running_list, rp = running_list;
851          rp != NULL;
852          rp = rp->next) {
853         if (!(rp->state == build_pending ||
854             rp->state == build_subtree)) {
855             quiescent = false;
856             rp_prev = &rp->next;
857         } else if (rp->state == build_pending) {
858             line = get_prop(rp->target->prop, line_prop);
859             for (dep = line->body.line.dependencies;
860                  dep != NULL;
861                  dep = dep->next) {
862                 if (dep->name->state == build_running ||
863                     dep->name->state == build_pending ||
864                     dep->name->state == build_serial) {
865                     break;
866                 }
867             }
868             if (dep == NULL) {
869                 for (target_group = line->body.line.target_group;
870                      target_group != NULL;
871                      target_group = target_group->next) {
872                     if (is_running(target_group->name)) {
873                         break;
874                     }
875                 }
876                 if (target_group == NULL) {
877                     *rp_prev = rp->next;
878                     if (rp->next == NULL) {
879                         running_tail = rp_prev;
880                     }
881                     recursion_level = rp->recursion_level;
882                     rp->target->state = rp->redo ?
883                         build_dont_know : build_pending;
884                     saved_commands_done = commands_done;
885                     conditionals =
886                         set_conditionals
887                             (rp->conditional_cnt,
888                              rp->conditional_targets);
889                     rp->target->dont_activate_cond_values =
890                         if ((doname_check(rp->target,
891                                             rp->do_get,
892                                             rp->implicit,
893                                             rp->target->has_target
894                                             build_running) &&
895                             !commands_done) {
896                             commands_done =
897                                 saved_commands_done;
898                         }
899                     rp->target->dont_activate_cond_values =
900                         reset_conditionals
901                             (rp->conditional_cnt,
902                              rp->conditional_targets,
903                              conditionals);
904                     quiescent = false;
905                     delete_running_struct(rp);
906                     goto start_loop_2;
907                 } else {
908                     rp_prev = &rp->next;
909                 }

```

```

910         } else {
911             rp_prev = &rp->next;
912         }
913     } else {
914         rp_prev = &rp->next;
915     }
916 }
917 /*
918  * If nothing has been found to build and there exists a subtree
919  * target with no dependency conflicts, build it.
920  */
921 if (quiescent) {
922 start_loop_3:
923     for (rp_prev = &running_list, rp = running_list;
924          rp != NULL;
925          rp = rp->next) {
926         if (rp->state == build_subtree) {
927             if (!dependency_conflict(rp->target)) {
928                 *rp_prev = rp->next;
929                 if (rp->next == NULL) {
930                     running_tail = rp_prev;
931                 }
932                 recursion_level = rp->recursion_level;
933                 doname_subtree(rp->target,
934                               rp->do_get,
935                               rp->implicit);
1122 #ifdef DISTRIBUTED
1123                 just_did_subtree = true;
1124 #endif
936                 quiescent = false;
937                 delete_running_struct(rp);
938                 goto start_loop_3;
939             } else {
940                 subtree_target = rp_prev;
941                 rp_prev = &rp->next;
942             }
943         } else {
944             rp_prev = &rp->next;
945         }
946     }
947 }
948 /*
949  * If still nothing found to build, we either have a deadlock
950  * or a subtree with a dependency conflict with something waiting
951  * to build.
952  */
953 if (quiescent) {
954     if (subtree_target == NULL) {
955         fatal(catgets(catd, 1, 126, "Internal error: deadlock de
956     } else {
957         rp = *subtree_target;
958         if (debug_level > 0) {
959             warning(catgets(catd, 1, 127, "Conditional macro
960                 subtree_conflict2->string_mb,
961                 rp->target->string_mb,
962                 subtree_conflict->string_mb);
963         }
964         *subtree_target = (*subtree_target)->next;
965         if (rp->next == NULL) {
966             running_tail = subtree_target;
967         }
968         recursion_level = rp->recursion_level;
969         doname_subtree(rp->target, rp->do_get, rp->implicit);
1159 #ifdef DISTRIBUTED
1160         just_did_subtree = true;
1161 #endif

```

```

970         delete_running_struct(rp);
971     }
972 }
973 }
_____unchanged_portion_omitted_____
1186 /*
1187  * finish_children(docheck)
1188  *
1189  * Finishes the processing for all targets which were running
1190  * and have now completed.
1191  *
1192  * Parameters:
1193  *     docheck           Completely check the finished target
1194  *
1195  * Static variables used:
1196  *     running_tail     The tail of the running list
1197  *
1198  * Global variables used:
1199  *     continue_after_error -k flag
1200  *     fatal_in_progress   True if we are finishing up after fatal err
1201  *     running_list       List of running processes
1202  */
1203 void
1204 finish_children(Boolean docheck)
1205 {
1206     int             cmds_length;
1207     Property        line;
1208     Property        line2;
1209     struct stat     out_buf;
1210     Running         rp;
1211     Running         *rp_prev;
1212     Cmd_line        rule;
1213     Boolean         silent_flag;
1214
1215     for (rp_prev = &running_list, rp = running_list;
1216          rp != NULL;
1217          rp = rp->next) {
1218 bypass_for_loop_inc_4:
1219         /*
1220          * If the state is ok or failed, then this target has
1221          * finished building.
1222          * In parallel_mode, output the accumulated stdout/stderr.
1223          * Read the auto dependency stuff, handle a failed build,
1224          * update the target, then finish the doname process for
1225          * that target.
1226          */
1227         if (rp->state == build_ok || rp->state == build_failed) {
1228             *rp_prev = rp->next;
1229             if (rp->next == NULL) {
1230                 running_tail = rp_prev;
1231             }
1232             if ((line2 = rp->command) == NULL) {
1233                 line2 = get_prop(rp->target->prop, line_prop);
1234             }
1235             if (dmake_mode_type == distributed_mode) {
1236                 if (rp->make_refd) {
1237                     maybe_reread_make_state();
1238                 }
1239             } else {
1240                 /*
1433                  * Send an Avo_MToolJobResultMsg to maketool.
1434                  */
1435 #ifdef DISTRIBUTED
1436                 if (send_mtool_msgs) {
1437                     send_job_result_msg(rp);

```

```

1438     }
1439 #endif
1440 /*
1441  * Check if there were any job output
1442  * from the parallel build.
1443  */
1444 if (rp->stdout_file != NULL) {
1445     if (stat(rp->stdout_file, &out_buf) < 0)
1446         fatal(catgets(catd, 1, 130, "sta
1447             rp->stdout_file,
1448             errmsg(errno));
1449     }
1450     if ((line2 != NULL) &&
1451         (out_buf.st_size > 0)) {
1452         cmds_length = 0;
1453         for (rule = line2->body.line.com
1454             silent_flag = silent;
1455             rule != NULL;
1456             rule = rule->next) {
1457             cmds_length += rule->com
1458             silent_flag = BOOLEAN(si
1459         }
1460         if (out_buf.st_size != cmds_leng
1461             output_mode == txt2_mode) {
1462             dump_out_file(rp->stdout
1463         }
1464     }
1465     (void) unlink(rp->stdout_file);
1466     retmem_mb(rp->stdout_file);
1467     rp->stdout_file = NULL;
1468 }
1469
1470 if (!out_err_same && (rp->stderr_file != NULL))
1471     if (stat(rp->stderr_file, &out_buf) < 0)
1472         fatal(catgets(catd, 1, 130, "sta
1473             rp->stderr_file,
1474             errmsg(errno));
1475     }
1476     if ((line2 != NULL) &&
1477         (out_buf.st_size > 0)) {
1478         dump_out_file(rp->stderr_file, t
1479     }
1480     (void) unlink(rp->stderr_file);
1481     retmem_mb(rp->stderr_file);
1482     rp->stderr_file = NULL;
1483 }
1484 }
1485 check_state(rp->temp_file);
1486 if (rp->temp_file != NULL) {
1487     free_name(rp->temp_file);
1488 }
1489 rp->temp_file = NULL;
1490 if (rp->state == build_failed) {
1491     line = get_prop(rp->target->prop, line_prop);
1492     if (line != NULL) {
1493         line->body.line.command_used = NULL;
1494     }
1495     if (continue_after_error ||
1496         fatal_in_progress ||
1497         !docheck) {
1498         warning(catgets(catd, 1, 256, "Command f
1499             rp->command ? line2->body.line.t
1500         build_failed_seen = true;
1501     } else {
1502         /*
1503          * XXX??? - DMake needs to exit(),

```

```

1304     * but shouldn't call fatal().
1305     */
1306 #ifdef PRINT_EXIT_STATUS
1307     warning(NOCATGETS("I'm in finish_childre
1308 #endif
1309
1310     fatal(catgets(catd, 1, 258, "Command fai
1311             rp->command ? line2->body.line.t
1312     }
1313     }
1314     if (!docheck) {
1315         delete_running_struct(rp);
1316         rp = *rp_prev;
1317         if (rp == NULL) {
1318             break;
1319         } else {
1320             goto bypass_for_loop_inc_4;
1321         }
1322     }
1323     update_target(get_prop(rp->target->prop, line_prop),
1324                 rp->state);
1325     finish_doname(rp);
1326     delete_running_struct(rp);
1327     rp = *rp_prev;
1328     if (rp == NULL) {
1329         break;
1330     } else {
1331         goto bypass_for_loop_inc_4;
1332     }
1333     } else {
1334         rp_prev = &rp->next;
1335     }
1336 }
1337 }
1338
1339 unchanged portion omitted
1340
1341 1497 /*
1342 1498 *
1343 1499 *     add_running(target, true_target, command, recursion_level, auto_count,
1344 1500 *     automatics, do_get, implicit)
1345 1501 *
1346 1502 *     Adds a record on the running list for this target, which
1347 1503 *     was just spawned and is running.
1348 1504 *
1349 1505 *     Parameters:
1350 1506 *         target           Target being built
1351 1507 *         true_target     True target for target
1352 1508 *         command        Running command.
1353 1509 *         recursion_level Debug indentation level
1354 1510 *         auto_count     Count of automatic dependencies
1355 1511 *         automatics     List of automatic dependencies
1356 1512 *         do_get         Sccs get flag
1357 1513 *         implicit       Implicit flag
1358 1514 *
1359 1515 *     Static variables used:
1360 1516 *         running_tail   Tail of running list
1361 1517 *         process_running PID of process
1362 1518 *
1363 1519 *     Global variables used:
1364 1520 *         current_line   Current line for target
1365 1521 *         current_target Current target being built
1366 1522 *         stderr_file    Temporary file for stdout
1367 1523 *         stdout_file    Temporary file for stdout
1368 1524 *         temp_file_name Temporary file for auto dependencies
1369 1525 void
1370 1526 add_running(Name target, Name true_target, Property command, int recursion_level

```

```

1527 {
1528     Running      rp;
1529     Name         *p;

1531     rp = new_running_struct();
1532     rp->state = build_running;
1533     rp->target = target;
1534     rp->true_target = true_target;
1535     rp->command = command;
1536     rp->recursion_level = recursion_level;
1537     rp->do_get = do_get;
1538     rp->implicit = implicit;
1539     rp->auto_count = auto_count;
1540     if (auto_count > 0) {
1541         rp->automatics = (Name *) getmem(auto_count * sizeof(Name));
1542         for (p = rp->automatics; auto_count > 0; auto_count--) {
1543             *p++ = *automatics++;
1544         }
1545     } else {
1546         rp->automatics = NULL;
1547     }
1548 #ifdef DISTRIBUTED
1549     if (dmake_mode_type == distributed_mode) {
1550         rp->make_refd = called_make;
1551         called_make = false;
1552     } else
1553 #endif
1554     {
1555         rp->pid = process_running;
1556         process_running = -1;
1557         childPid = -1;
1558     }
1559     rp->job_msg_id = job_msg_id;
1560     rp->stdout_file = stdout_file;
1561     rp->stderr_file = stderr_file;
1562     rp->temp_file = temp_file_name;
1563     rp->redo = false;
1564     rp->next = NULL;
1565     store_conditionals(rp);
1566     stdout_file = NULL;
1567     stderr_file = NULL;
1568     temp_file_name = NULL;
1569     current_target = NULL;
1570     current_line = NULL;
1571     *running_tail = rp;
1572     running_tail = &rp->next;
1573 }

```

unchanged_portion_omitted

```

2112 #ifdef DISTRIBUTED
2113 /*
2114  * Create and send an Avo_MToolJobResultMsg.
2115  */
2116 static void
2117 send_job_result_msg(Running rp)
2118 {
2119     Avo_MToolJobResultMsg *msg;
2120     RWCollectable *xdr_msg;

2122     msg = new Avo_MToolJobResultMsg();
2123     msg->setResult(rp->job_msg_id,
2124                 (rp->state == build_ok) ? 0 : 1,
2125                 DONE);
2126     append_job_result_msg(msg,
2127                          rp->stdout_file,
2128                          rp->stderr_file);

```

```

2130     xdr_msg = (RWCollectable *)msg;
2131     xdr(get_xdrs_ptr(), xdr_msg);
2132     (void) fflush(get_mtool_msgs_fp());

2134     delete msg;
2135 }

2137 /*
2138  * Append the stdout/err to Avo_MToolJobResultMsg.
2139  */
2140 static void
2141 append_job_result_msg(Avo_MToolJobResultMsg *msg, char *outFn, char *errFn)
2142 {
2143     FILE *fp;
2144     char line[MAXPATHLEN];

2146     fp = fopen(outFn, "r");
2147     if (fp == NULL) {
2148         /* Hmm... what should we do here? */
2149         return;
2150     }
2151     while (fgets(line, MAXPATHLEN, fp) != NULL) {
2152         if (line[strlen(line) - 1] == '\n') {
2153             line[strlen(line) - 1] = '\0';
2154         }
2155         msg->appendOutput(AVO_STRDUP(line));
2156     }
2157     (void) fclose(fp);
2158 }
2159 #endif

```

```

1907 static void
1908 delete_running_struct(Running rp)
1909 {
1910     if ((rp->conditional_cnt > 0) &&
1911         (rp->conditional_targets != NULL)) {
1912         retmem_mb((char *) rp->conditional_targets);
1913     }
1914 /**/
1915     if ((rp->auto_count > 0) &&
1916         (rp->automatics != NULL)) {
1917         retmem_mb((char *) rp->automatics);
1918     }
1919 /**/
1920     if(rp->sprodep_value) {
1921         free_name(rp->sprodep_value);
1922     }
1923     if(rp->sprodep_env) {
1924         retmem_mb(rp->sprodep_env);
1925     }
1926     retmem_mb((char *) rp);

```

unchanged_portion_omitted


```

*****
56788 Wed May 20 11:55:57 2015
new/usr/src/cmd/make/bin/read.cc
make: undef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  *      read.c
28  *
29  *      This file contains the makefile reader.
30  */

32 /*
33  * Included files
34  */
35 #include <alloca.h>          /* alloca() */
36 #include <errno.h>          /* errno */
37 #include <fcntl.h>          /* fcntl() */
38 #include <mk/defs.h>
39 #include <mksh/macro.h>     /* expand_value(), expand_macro() */
40 #include <mksh/misc.h>     /* getmem() */
41 #include <mksh/read.h>     /* get_next_block_fn() */
42 #include <sys/uio.h>        /* read() */
43 #include <unistd.h>         /* read(), unlink() */

46 /*
47  * typedefs & structs
48  */

50 /*
51  * Static variables
52  */

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

56 /*
57  * File table of contents
58  */
59 static void      parse_makefile(register Name true_makefile_name, register
60 static Source   push_macro_value(register Source bp, register wchar_t *b
61 extern void     enter_target_groups_and_dependencies(Name_vector target,

```

```

62 extern Name      normalize_name(register wchar_t *name_string, register i
64 /*
65  *      read_simple_file(makefile_name, chase_path, doname_it,
66  *                      complain, must_exist, report_file, lock_makefile)
67  *
68  *      Make the makefile and setup to read it. Actually read it if it is stdio
69  *
70  *      Return value:
71  *                      false if the read failed
72  *
73  *      Parameters:
74  *      makefile_name   Name of the file to read
75  *      chase_path      Use the makefile path when opening file
76  *      doname_it       Call doname() to build the file first
77  *      complain        Print message if doname/open fails
78  *      must_exist      Generate fatal if file is missing
79  *      report_file     Report file when running -P
80  *      lock_makefile   Lock the makefile when reading
81  *
82  *      Static variables used:
83  *
84  *      Global variables used:
85  *      do_not_exec_rule Is -n on?
86  *      file_being_read  Set to the name of the new file
87  *      line_number      The number of the current makefile line
88  *      makefiles_used   A list of all makefiles used, appended to
89  */

92 Boolean
93 read_simple_file(register Name makefile_name, register Boolean chase_path, regis
94 {
95     static short      max_include_depth;
96     register Property makefile = maybe_append_prop(makefile_name,
97                                                     makefile_prop);
98     Boolean           forget_after_parse = false;
99     static pathpt    makefile_path;
100    register int      n;
101    char              *path;
102    register Source   source = ALLOC(Source);
103    Property          orig_makefile = makefile;
104    Dependency        *dpp;
105    Dependency        dp;
106    register int      length;
107    wchar_t           *previous_file_being_read = file_being_read;
108    int               previous_line_number = line_number;
109    wchar_t           previous_current_makefile[MAXPATHLEN];
110    Makefile_type     save_makefile_type;
111    Name              normalized_makefile_name;
112    register wchar_t  *string_start;
113    register wchar_t  *string_end;

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

119    if (max_include_depth++ >= 40) {
120        fatal(catgets(catd, 1, 66, "Too many nested include statements")
121    }
122    if (makefile->body.makefile.contents != NULL) {
123        retmem(makefile->body.makefile.contents);
124    }
125    source->inp_buf =
126        source->inp_buf_ptr =
127        source->inp_buf_end = NULL;

```

```

128 source->error_converting = false;
129 makefile->body.makefile.contents = NULL;
130 makefile->body.makefile.size = 0;
131 if ((makefile_name->hash.length != 1) ||
132     (wcb[0] != (int) hyphen_char)) {
133     if ((makefile->body.makefile.contents == NULL) &&
134         (doname_it)) {
135         if (makefile_path == NULL) {
136             add_dir_to_path(".",
137                             &makefile_path,
138                             -1);
139             add_dir_to_path(NOCATGETS("/usr/share/lib/make")
140                             &makefile_path,
141                             -1);
142             add_dir_to_path(NOCATGETS("/etc/default"),
143                             &makefile_path,
144                             -1);
145         }
146         save_makefile_type = makefile_type;
147         makefile_type = reading_nothing;
148         if (doname(makefile_name, true, false) == build_dont_kno
149             /* Try normalized filename */
150             string_start=get_wstring(makefile_name->string_m
151 for (string_end=string_start+1; *string_end != L
152 normalized_makefile_name=normalize_name(string_s
153 if ((strcmp(makefile_name->string_mb, normalized
154         (doname(normalized_makefile_name, true,
155             n = access_vroot(makefile_name->string_m
156                 4,
157                 chase_path ?
158                 makefile_path : NULL,
159                 VROOT_DEFAULT);
160         if (n == 0) {
161             get_vroot_path((char **) NULL,
162                             &path,
163                             (char **) NULL);
164             if ((path[0] == (int) period_cha
165                 (path[1] == (int) slash_char
166                     path += 2;
167             }
168             MBSTOWCS(wcs_buffer, path);
169             makefile_name = GETNAME(wcs_buff
170                                     FIND_LENGTH);
171         }
172     }
173     retmem(string_start);
174     /*
175     * Commented out: retmem_mb(normalized_makefile_
176     * We have to return this memory, but it seems t
177     * in dmake or in Sun C++ 5.7 compiler (it works
178     * is compiled using Sun C++ 5.6).
179     */
180     // retmem_mb(normalized_makefile_name->string_mb
181 }
182 makefile_type = save_makefile_type;
183 }
184 source->string.free_after_use = false;
185 source->previous = NULL;
186 source->already_expanded = false;
187 /* Lock the file for read, but not when -n. */
188 if (lock_makefile &&
189     !do_not_exec_rule) {
191     make_state_lockfile = getmem(strlen(make_state->string_
192     (void) sprintf(make_state_lockfile,
193     NOCATGETS("%s.lock"),

```

```

194     make_state->string_mb);
195     (void) file_lock(make_state->string_mb,
196     make_state_lockfile,
197     (int *) &make_state_locked,
198     0);
199     if(!make_state_locked) {
200         printf(NOCATGETS("-- NO LOCKING for read\n"));
201         retmem_mb(make_state_lockfile);
202         make_state_lockfile = 0;
203         return failed;
204     }
205 }
206 if (makefile->body.makefile.contents == NULL) {
207     save_makefile_type = makefile_type;
208     makefile_type = reading_nothing;
209     if ((doname_it) &&
210         (doname(makefile_name, true, false) == build_failed)
211         if (complain) {
212             (void) fprintf(stderr,
213 #ifdef DISTRIBUTED
214             catgets(catd, 1, 67, "dma
215 #else
216             catgets(catd, 1, 237, "ma
217 #endif
218             makefile_name->string_mb)
219         }
220         max_include_depth--;
221         makefile_type = save_makefile_type;
222         return failed;
223     }
224     makefile_type = save_makefile_type;
225     //
226     // Before calling exists() make sure that we have the ri
227     //
228     makefile_name->stat.time = file_no_time;
229     if (exists(makefile_name) == file_doesnt_exist) {
230         if (complain ||
231             (makefile_name->stat.stat_errno != ENOENT))
232             if (must_exist) {
233                 fatal(catgets(catd, 1, 68, "Can'
234                 makefile_name->string_mb,
235                 errmsg(makefile_name->
236                     stat.stat_errno));
237             } else {
238                 warning(catgets(catd, 1, 69, "Ca
239                 makefile_name->string_mb,
240                 errmsg(makefile_name->
241                     stat.stat_errno))
242             }
243         max_include_depth--;
244         if(make_state_locked && (make_state_lockfile !=
245             (void) unlink(make_state_lockfile);
246             retmem_mb(make_state_lockfile);
247             make_state_lockfile = NULL;
248             make_state_locked = false;
249         }
250         retmem(wcb);
251         retmem_mb((char *)source);
252         return failed;
253     }
254     /*
255     * These values are the size and bytes of
256     * the MULTI-BYTE makefile.
257     */

```

```

256     orig_makefile->body.makefile.size =
257     makefile->body.makefile.size =
258     source->bytes_left_in_file =
259     makefile_name->stat.size;
260     if (report_file) {
261         for (dpp = &makefiles_used;
262              *dpp != NULL;
263              dpp = &(*dpp)->next);
264         dp = ALLOC(Dependency);
265         dp->next = NULL;
266         dp->name = makefile_name;
267         dp->automatic = false;
268         dp->stale = false;
269         dp->built = false;
270         *dpp = dp;
271     }
272     source->fd = open_vroot(makefile_name->string_mb,
273                           O_RDONLY,
274                           0,
275                           NULL,
276                           VROOT_DEFAULT);
277     if (source->fd < 0) {
278         if (complain || (errno != ENOENT)) {
279             if (must_exist) {
280                 fatal(catgets(catd, 1, 70, "Can'
281 makefile_name->string_mb,
282 errmsg(errno));
283             } else {
284                 warning(catgets(catd, 1, 71, "Ca
285 makefile_name->string_mb
286 errmsg(errno));
287             }
288         }
289         max_include_depth--;
290         return failed;
291     }
292     (void) fcntl(source->fd, F_SETFD, 1);
293     orig_makefile->body.makefile.contents =
294     makefile->body.makefile.contents =
295     source->string.text.p =
296     source->string.buffer.start =
297     ALLOC_WC((int) (makefile_name->stat.size + 2));
298     if (makefile_type == reading_cpp_file) {
299         forget_after_parse = true;
300     }
301     source->string.text.end = source->string.text.p;
302     source->string.buffer.end =
303     source->string.text.p + makefile_name->stat.size;
304 } else {
305     /* Do we ever reach here? */
306     source->fd = -1;
307     source->string.text.p =
308     source->string.buffer.start =
309     makefile->body.makefile.contents;
310     source->string.text.end =
311     source->string.buffer.end =
312     source->string.text.p + makefile->body.makefile.size
313     source->bytes_left_in_file =
314     makefile->body.makefile.size;
315 }
316     file_being_read = wcb;
317 } else {
318     char *stdin_text_p;
319     char *stdin_text_end;
320     char *stdin_buffer_start;
321     char *stdin_buffer_end;

```

```

322     char *p_mb;
323     int num_mb_chars;
324     size_t num_wc_chars;
325
326     MBSTOWCS(wcs_buffer, NOCATGETS("Standard in"));
327     makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
328     /*
329     * Memory to read standard in, then convert it
330     * to wide char strings.
331     */
332     stdin_buffer_start =
333     stdin_text_p = getmem(length = 1024);
334     stdin_buffer_end = stdin_text_p + length;
335     MBSTOWCS(wcs_buffer, NOCATGETS("standard input"));
336     file_being_read = (wchar_t *) wsdup(wcs_buffer);
337     line_number = 0;
338     while ((n = read(fileno(stdin),
339                    stdin_text_p,
340                    length)) > 0) {
341         length -= n;
342         stdin_text_p += n;
343         if (length == 0) {
344             p_mb = getmem(length = 1024 +
345                          (stdin_buffer_end -
346                           stdin_buffer_start));
347             (void) strncpy(p_mb,
348                          stdin_buffer_start,
349                          (stdin_buffer_end -
350                           stdin_buffer_start));
350             retmem_mb(stdin_buffer_start);
351             stdin_text_p = p_mb +
352             (stdin_buffer_end - stdin_buffer_start);
353             stdin_buffer_start = p_mb;
354             stdin_buffer_end =
355             stdin_buffer_start + length;
356             length = 1024;
357         }
358     }
359     if (n < 0) {
360         fatal(catgets(catd, 1, 72, "Error reading standard input
361 errmsg(errno));
362     }
363     stdin_text_p = stdin_buffer_start;
364     stdin_text_end = stdin_buffer_end - length;
365     num_mb_chars = stdin_text_end - stdin_text_p;
366
367     /*
368     * Now, convert the sequence of multibyte chars into
369     * a sequence of corresponding wide character codes.
370     */
371     source->string.free_after_use = false;
372     source->previous = NULL;
373     source->bytes_left_in_file = 0;
374     source->fd = -1;
375     source->already_expanded = false;
376     source->string.buffer.start =
377     source->string.text.p = ALLOC_WC(num_mb_chars + 1);
378     source->string.buffer.end =
379     source->string.text.p + num_mb_chars;
380     num_wc_chars = mbstowcs(source->string.text.p,
381                          stdin_text_p,
382                          num_mb_chars);
383     if (((int) num_wc_chars >= 0) {
384         source->string.text.end =
385         source->string.text.p + num_wc_chars;
386     }
387 }

```



```

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

1835 /*
1836 *      fatal_reader(format, args...)
1837 *
1838 *      Parameters:
1839 *          format      printf style format string
1840 *          args        arguments to match the format
1841 *
1842 *      Global variables used:
1843 *          file_being_read Name of the makefile being read
1844 *          line_number     Line that is being read
1845 *          report_pwd      Indicates whether current path should be shown
1846 *          temp_file_name  When reading tempfile we report that name
1847 */
1848 /*VARARGS*/
1849 void
1850 fatal_reader(char * pattern, ...)
1851 {
1852     va_list args;
1853     char message[1000];
1854
1855     va_start(args, pattern);
1856     if (file_being_read != NULL) {
1857         WCSTOMBS(mbs_buffer, file_being_read);
1858         if (line_number != 0) {
1859             (void) sprintf(message,
1860                 catgets(catd, 1, 112, "%s, line %d: %s"),
1861                 mbs_buffer,
1862                 line_number,
1863                 pattern);
1864         } else {
1865             (void) sprintf(message,
1866                 "%s: %s",
1867                 mbs_buffer,
1868                 pattern);
1869         }
1870         pattern = message;
1871     }
1872
1873     (void) fflush(stdout);
1874 #ifdef DISTRIBUTED
1875     (void) fprintf(stderr, catgets(catd, 1, 113, "dmake: Fatal error in read
1876 #else
1874     (void) fprintf(stderr, catgets(catd, 1, 238, "make: Fatal error in reade
1878 #endif
1875     (void) vfprintf(stderr, pattern, args);
1876     (void) fprintf(stderr, "\n");
1877     va_end(args);
1878
1879     if (temp_file_name != NULL) {
1880         (void) fprintf(stderr,
1885 #ifdef DISTRIBUTED
1886             catgets(catd, 1, 114, "dmake: Temp-file %s not re
1887 #else
1881             catgets(catd, 1, 239, "make: Temp-file %s not rem
1889 #endif
1882             temp_file_name->string_mb);
1883             temp_file_name = NULL;
1884     }

```

```

1886     if (report_pwd) {
1887         (void) fprintf(stderr,
1888             catgets(catd, 1, 115, "Current working directory
1889             get_current_path());
1890     }
1891     (void) fflush(stderr);
1892     exit_status = 1;
1893     exit(1);
1894 }
_____unchanged_portion_omitted_____

```

new/usr/src/cmd/make/include/avo/intl.h

1

1412 Wed May 20 11:55:59 2015

new/usr/src/cmd/make/include/avo/intl.h

make: unifdef for MAKETOOL and DISTRIBUTED (undefined)

```
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2001 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */
```

```
26 #ifndef _AVO_INTL_H
27 #define _AVO_INTL_H
```

```
30 /*
31  * For catgets
32 */
33 #include <nl_types.h>
```

```
36 /*
37  * NOCATGETS is a dummy macro that returns its argument.
38  * It is used to identify strings that we consciously do not
39  * want to apply catgets() to. We have tools that check the
40  * sources for strings that are not catgets'd and the tools
41  * ignore strings that are NOCATGETS'd.
42 */
43 #define NOCATGETS(str) (str)
```

```
45 /*
46  * Define the various text domains
47 */
48 #define AVO_DOMAIN_CODEMGR "codemgr"
49 #define AVO_DOMAIN_VERTOOL "vertool"
50 #define AVO_DOMAIN_FILEMERGE "filemerge"
51 #define AVO_DOMAIN_DMAKE "dmake"
52 #define AVO_DOMAIN_PMAKE "pmake"
53 #define AVO_DOMAIN_FREEZEPOINT "freezept"
54 #define AVO_DOMAIN_MAKETOOL "maketool"
```

```
50 #endif
```

new/usr/src/cmd/make/include/mk/defs.h

1

```
*****
14246 Wed May 20 11:55:59 2015
new/usr/src/cmd/make/include/mk/defs.h
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 #ifndef _MK_DEFS_H
2 #define _MK_DEFS_H
3 /*
4  * CDDL HEADER START
5  *
6  * The contents of this file are subject to the terms of the
7  * Common Development and Distribution License (the "License").
8  * You may not use this file except in compliance with the License.
9  *
10 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 * or http://www.opensolaris.org/os/licensing.
12 * See the License for the specific language governing permissions
13 * and limitations under the License.
14 *
15 * When distributing Covered Code, include this CDDL HEADER in each
16 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 * If applicable, add the following below this CDDL HEADER, with the
18 * fields enclosed by brackets "[]" replaced with your own identifying
19 * information: Portions Copyright [yyyy] [name of copyright owner]
20 *
21 * CDDL HEADER END
22 */
23 /*
24 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
25 * Use is subject to license terms.
26 */

28 /*
29 * Included files
30 */
31 #ifdef DISTRIBUTED
32 #   include <dm/Avo_AcknowledgeMsg.h>
33 #   include <dm/Avo_DoJobMsg.h>
34 #   include <dm/Avo_JobResultMsg.h>
35 #endif

32 #include <mksh/defs.h>

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

36 /*
37  * Defined macros
38 */

40 #define SKIPSPACE(x)   while (*x &&
41                          ((*x == (int) space_char) ||
42                           (*x == (int) tab_char) ||
43                           (*x == (int) comma_char))) {
44                          x++;
45                          }

47 #define SKIPWORD(x)   while (*x &&
48                          (*x != (int) space_char) &&
49                          (*x != (int) tab_char) &&
50                          (*x != (int) newline_char) &&
51                          (*x != (int) comma_char) &&
52                          (*x != (int) equal_char)) {
53                          x++;

```

new/usr/src/cmd/make/include/mk/defs.h

2

```
54 }
56 #define SKIPTOEND(x)   while (*x &&
57                          (*x != (int) newline_char)) {
58                          x++;
59                          }

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

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

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

69 #define SETVAR(name, value, append) \
70      setvar_daemon(name, value, append, no_daemon, \
71                    true, debug_level)
72 #define MAX(a,b)          (((a)>(b))?a):(b)
73 /*
74  * New feature added to SUN5_0 make, invoke the vanilla svr4 make when
75  * the USE_SVR4_MAKE environment variable is set.
76 */
77 #define SVR4_MAKE          "/usr/ccs/lib/svr4.make"
78 #define USE_SVR4_MAKE     "USE_SVR4_MAKE"
79 /*
80  * The standard MAXHOSTNAMELEN is 64. We want 32.
81 */
82 #define MAX_HOSTNAMELEN   32

85 /*
86  * typedefs & structs
87 */
88 typedef enum {
89     no_state,
90     scan_name_state,
91     scan_command_state,
92     enter_dependencies_state,
93     enter_conditional_state,
94     enter_equal_state,
95     illegal_bytes_state,
96     illegal_eoln_state,
97     poorly_formed_macro_state,
98     exit_state
99 } Reader_state;
_____ unchanged_portion_omitted_____

169 /*
170  * Typedefs for all structs
171 */
172 typedef struct _Cmd_line      *Cmd_line, Cmd_line_rec;
173 typedef struct _Dependency    *Dependency, Dependency_rec;
174 typedef struct _Macro         *Macro, Macro_rec;
175 typedef struct _Name_vector   *Name_vector, Name_vector_rec;
176 typedef struct _Percent       *Percent, Percent_rec;
177 typedef struct _Dyntarget     *Dyntarget;
178 typedef struct _Recursive_make *Recursive_make, Recursive_make_rec;
179 typedef struct _Running       *Running, Running_rec;

182 /*
183  * extern declarations for all global variables.
184  * The actual declarations are in globals.cc
185 */

```

```

186 extern Boolean    allrules_read;
187 extern Name       posix_name;
188 extern Name       svr4_name;
189 extern Boolean    sdot_target;
190 extern Boolean    all_parallel;
191 extern Boolean    assign_done;
192 extern Boolean    build_failed_seen;
201 #ifdef DISTRIBUTED
202 extern Boolean    building_serial;
203 #endif
193 extern Name       built_last_make_run;
194 extern Name       c_at;
206 #ifdef DISTRIBUTED
207 extern Boolean    called_make;
208 #endif
195 extern Boolean    command_changed;
196 extern Boolean    commands_done;
197 extern Chain     conditional_targets;
198 extern Name       conditionals;
199 extern Boolean    continue_after_error;
200 extern Property   current_line;
201 extern Name       current_make_version;
202 extern Name       current_target;
203 extern short      debug_level;
204 extern Cmd_line   default_rule;
205 extern Name       default_rule_name;
206 extern Name       default_target_to_build;
207 extern Boolean    depinfo_already_read;
208 extern Name       dmake_group;
209 extern Name       dmake_max_jobs;
210 extern Name       dmake_mode;
211 extern DMake_mode dmake_mode_type;
212 extern Name       dmake_output_mode;
213 extern DMake_output_mode output_mode;
214 extern Name       dmake_odir;
215 extern Name       dmake_rcfile;
216 extern Name       done;
217 extern Name       dot;
218 extern Name       dot_keep_state;
219 extern Name       dot_keep_state_file;
220 extern Name       empty_name;
221 extern Boolean    fatal_in_progress;
222 extern int        file_number;
223 extern Name       force;
224 extern Name       ignore_name;
225 extern Boolean    ignore_errors;
226 extern Boolean    ignore_errors_all;
227 extern Name       init;
228 extern int        job_msg_id;
229 extern Boolean    keep_state;
230 extern Name       make_state;
231 #ifdef TEAMWARE_MAKE_CMN
232 extern timestruc_t make_state_before;
233 #endif
234 extern Boolean    make_state_locked;
235 extern Dependency makefiles_used;
236 extern Name       makeflags;
237 extern Name       make_version;
238 extern char       mbs_buffer2[];
239 extern char       *mbs_ptr;
240 extern char       *mbs_ptr2;
241 extern Boolean    no_action_was_taken;
242 extern int        mtool_msgs_fd;
243 extern Boolean    no_parallel;
244 extern Name       no_parallel_name;
245 extern Name       not_auto;

```

```

246 extern Boolean    only_parallel;
247 extern Boolean    parallel;
248 extern Name       parallel_name;
249 extern Name       localhost_name;
250 extern int        parallel_process_cnt;
251 extern Percent    percent_list;
252 extern Dymtarget   dyntarget_list;
253 extern Name       plus;
254 extern Name       pmake_machinesfile;
255 extern Name       precious;
256 extern Name       primary_makefile;
257 extern Boolean    quest;
258 extern short      read_trace_level;
259 extern Boolean    reading_dependencies;
260 extern int        recursion_level;
261 extern Name       recursive_name;
262 extern short      report_dependencies_level;
263 extern Boolean    report_pwd;
264 extern Boolean    rewrite_statefile;
265 extern Running    running_list;
266 extern char       *sccs_dir_path;
267 extern Name       sccs_get_name;
268 extern Name       sccs_get_posix_name;
269 extern Cmd_line   sccs_get_rule;
270 extern Cmd_line   sccs_get_org_rule;
271 extern Cmd_line   sccs_get_posix_rule;
272 extern Name       get_name;
273 extern Name       get_posix_name;
274 extern Cmd_line   get_rule;
275 extern Cmd_line   get_posix_rule;
276 extern Boolean    send_mtool_msgs;
277 extern Boolean    all_precious;
278 extern Boolean    report_cwd;
279 extern Boolean    silent_all;
280 extern Boolean    silent;
281 extern Name       silent_name;
282 extern char       *stderr_file;
283 extern char       *stdout_file;
284 extern Boolean    stdout_stderr_same;
285 extern Dependency suffixes;
286 extern Name       suffixes_name;
287 extern Name       sunpro_dependencies;
288 extern Boolean    target_variants;
289 extern const char *tmpdir;
290 extern const char *temp_file_directory;
291 extern Name       temp_file_name;
292 extern short      temp_file_number;
293 extern wchar_t    *top_level_target;
294 extern Boolean    touch;
295 extern Boolean    trace_reader;
296 extern Boolean    build_unconditional;
297 extern pathpt     vroot_path;
298 extern Name       wait_name;
299 extern wchar_t    wcs_buffer2[];
300 extern wchar_t    *wcs_ptr;
301 extern wchar_t    *wcs_ptr2;
302 extern nl_catd    catd;
303 extern long int   hostid;

305 /*
306  * Declarations of system defined variables
307  */
308 /* On linux this variable is defined in 'signal.h' */
309 extern char       *sys_siglist[];

311 /*

```



```

312 * Declarations of system supplied functions
313 */
314 extern int          file_lock(char *, char *, int *, int);

316 /*
317 * Declarations of functions declared and used by make
318 */
319 extern void          add_pending(Name target, int recursion_level, Boolean do
320 extern void          add_running(Name target, Name true_target, Property comm
321 extern void          add_serial(Name target, int recursion_level, Boolean do_
322 extern void          add_subtree(Name target, int recursion_level, Boolean do
323 extern void          append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar
324 #ifdef DISTRIBUTED
325 extern void          await_dist(Boolean waitflg);
326 #endif
327 #ifdef TEAMWARE_MAKE_CMN
328 extern void          await_parallel(Boolean waitflg);
329 #endif
330 extern void          build_suffix_list(Name target_suffix);
331 extern Boolean       check_auto_dependencies(Name target, int auto_count, Nam
332 extern void          check_state(Name temp_file_name);
333 extern void          cond_macros_into_string(Name np, String_rec *buffer);
334 extern void          construct_target_string();
335 extern void          create_xdrs_ptr(void);
336 extern void          depvar_add_to_list (Name name, Boolean cmdline);
337 #ifdef DISTRIBUTED
338 extern void          distribute_rxm(Avo_DoJobMsg *dmake_job_msg);
339 extern int           getRxmMessage(void);
340 extern Avo_JobResultMsg* getJobResultMsg(void);
341 extern Avo_AcknowledgeMsg* getAcknowledgeMsg(void);
342 #endif
343 extern Doname        doname(register Name target, register Boolean do_get, re
344 extern Doname        doname_check(register Name target, register Boolean do_g
345 extern Doname        doname_parallel(Name target, Boolean do_get, Boolean imp
346 extern Doname        dosys(register Name command, register Boolean ignore_err
347 extern void          dump_make_state(void);
348 extern void          dump_target_list(void);
349 extern void          enter_conditional(register Name target, Name name, Name
350 extern void          enter_dependencies(register Name target, Chain target_gr
351 extern void          enter_dependency(Property line, register Name depe, Bool
352 extern void          enter_equal(Name name, Name value, register Boolean appe
353 extern Percent       enter_percent(register Name target, Chain target_group,
354 extern Dyntarget     enter_dyntarget(register Name target);
355 extern Name_vector   enter_name(String string, Boolean tail_present, register
356 extern Boolean       exec_vp(register char *name, register char **argv, char
357 extern Doname        execute_parallel(Property line, Boolean waitflg, Boolean
358 extern Doname        execute_serial(Property line);
359 extern timestruc_t& exists(register Name target);
360 extern void          fatal(const char *, ...);
361 extern void          fatal_reader(char *, ...);
362 extern Doname        find_ar_suffix_rule(register Name target, Name true_targ
363 extern Doname        find_double_suffix_rule(register Name target, Property *
364 extern Doname        find_percent_rule(register Name target, Property *comman
365 extern int           find_run_directory (char *cmd, char *cwd, char *dir, cha
366 extern Doname        find_suffix_rule(Name target, Name target_body, Name tar
367 extern Chain         find_target_groups(register Name_vector target_list, reg
368 extern void          finish_children(Boolean docheck);
369 extern void          finish_running(void);
370 extern void          free_chain(Name_vector ptr);
371 extern void          gather_recursive_deps(void);
372 extern char          *get_current_path(void);
373 extern int           get_job_msg_id(void);
374 extern FILE          *get_mtool_msgs_fp(void);
375 #ifdef DISTRIBUTED
376 extern Boolean       get_dmake_group_specified(void);
377 extern Boolean       get_dmake_max_jobs_specified(void);

```

```

378 extern Boolean       get_dmake_mode_specified(void);
379 extern Boolean       get_dmake_odir_specified(void);
380 extern Boolean       get_dmake_rcfile_specified(void);
381 extern Boolean       get_pmake_machinesfile_specified(void);
382 #endif
383 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
384 extern XDR           *get_xdrs_ptr(void);
385 #endif
386 extern wchar_t      *getmem_wc(register int size);
387 /* On linux getwd(char *) is defined in 'unistd.h' */
388 #ifdef __cplusplus
389 extern "C" {
390 #endif
391 extern char          *getwd(char *);
392 #ifdef __cplusplus
393 }
394 #endif
395 extern void          handle_interrupt(int);
396 extern Boolean       is_running(Name target);
397 extern void          load_cached_names(void);
398 extern Boolean       parallel_ok(Name target, Boolean line_prop_must_exists);
399 extern void          print_dependencies(register Name target, register Proper
400 extern void          send_job_start_msg(Property line);
401 extern void          send_rsrc_info_msg(int max_jobs, char *hostname, char *u
402 extern void          print_value(register Name value, Daemon daemon);
403 extern timestruc_t& read_archive(register Name target);
404 extern int           read_dir(Name dir, wchar_t *pattern, Property line, wcha
405 extern void          read_directory_of_file(register Name file);
406 extern int           read_make_machines(Name make_machines_name);
407 extern Boolean       read_simple_file(register Name makefile_name, register B
408 extern void          remove_recursive_dep(Name target);
409 extern void          report_recursive_dep(Name target, char *line);
410 extern void          report_recursive_done(void);
411 extern void          report_recursive_init(void);
412 extern Recursive_make find_recursive_target(Name target);
413 extern void          reset_locals(register Name target, register Property old
414 extern void          set_locals(register Name target, register Property old_l
415 setvar_append(register Name name, register Name value);
416 #ifdef DISTRIBUTED
417 extern void          setvar_envvar(Avo_DoJobMsg *dmake_job_msg);
418 #else
419 extern void          setvar_envvar(void);
420 #endif
421 extern void          special_reader(Name target, register Name_vector depes,
422 startup_rxm();
423 extern Doname        target_can_be_built(register Name target);
424 extern char          *time_to_string(const timestruc_t &time);
425 extern void          update_target(Property line, Doname result);
426 extern void          warning(char *, ...);
427 extern void          write_state_file(int report_recursive, Boolean exiting);
428 extern Name          vpath_translation(register Name cmd);
429 #define DEPINFO_FMT_VERSION "VERS2$"
430 #define VER_LEN strlen(DEPINFO_FMT_VERSION)
431 #endif

```

new/usr/src/cmd/make/include/mksh/dosys.h

1

```
*****
1947 Wed May 20 11:55:59 2015
new/usr/src/cmd/make/include/mksh/dosys.h
make: unifdef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 #ifndef _MKSH_DOSYS_H
2 #define _MKSH_DOSYS_H
3 /*
4  * CDDL HEADER START
5  *
6  * The contents of this file are subject to the terms of the
7  * Common Development and Distribution License (the "License").
8  * You may not use this file except in compliance with the License.
9  *
10 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 * or http://www.opensolaris.org/os/licensing.
12 * See the License for the specific language governing permissions
13 * and limitations under the License.
14 *
15 * When distributing Covered Code, include this CDDL HEADER in each
16 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 * If applicable, add the following below this CDDL HEADER, with the
18 * fields enclosed by brackets "[]" replaced with your own identifying
19 * information: Portions Copyright [yyyy] [name of copyright owner]
20 *
21 * CDDL HEADER END
22 */
23 /*
24 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
25 * Use is subject to license terms.
26 */

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

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

38 #endif
```

new/usr/src/cmd/make/include/mksh/mksh.h

1

1047 Wed May 20 11:56:00 2015

new/usr/src/cmd/make/include/mksh/mksh.h

make: undef for MAKE TOOL and DISTRIBUTED (undefined)

```
1 #ifndef _MKSH_MKSH_H
2 #define _MKSH_MKSH_H
3 /*
4  * CDDL HEADER START
5  *
6  * The contents of this file are subject to the terms of the
7  * Common Development and Distribution License (the "License").
8  * You may not use this file except in compliance with the License.
9  *
10 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 * or http://www.opensolaris.org/os/licensing.
12 * See the License for the specific language governing permissions
13 * and limitations under the License.
14 *
15 * When distributing Covered Code, include this CDDL HEADER in each
16 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 * If applicable, add the following below this CDDL HEADER, with the
18 * fields enclosed by brackets "[]" replaced with your own identifying
19 * information: Portions Copyright [yyyy] [name of copyright owner]
20 *
21 * CDDL HEADER END
22 */
23 /*
24 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
25 * Use is subject to license terms.
26 */

29 /*
30 * Included files
31 */
32 #if defined(DISTRIBUTED) || defined(MAKE TOOL) /* tolik */
33 #   include <dm/Avo_DmakeCommand.h>
34 #endif

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

39 #if defined(DISTRIBUTED) || defined(MAKE TOOL) /* tolik */

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

43 #endif /* TEAMWARE_MAKE_CMN */

37 #endif
```

```

*****
20098 Wed May 20 11:56:00 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: undef for MAKETOOL and DISTRIBUTED (undefined)
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2005 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

27 /*
28 *      dosys.cc
29 *
30 *      Execute one commandline
31 */

33 /*
34 * Included files
35 */
36 #include <sys/wait.h>          /* WIFEXITED(status) */
37 #include <alloca.h>           /* alloca() */

39 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL) /* tolik */
40 #if defined(DISTRIBUTED)
41 #   include <dm/Avo_CmdOutput.h>
42 #   include <rw/xdrstrea.h>
43 #endif
44 #endif

39 #include <stdio.h>             /* errno */
40 #include <errno.h>             /* errno */
41 #include <fcntl.h>             /* open() */
42 #include <mksh/dosys.h>
43 #include <mksh/macro.h>        /* getvar() */
44 #include <mksh/misc.h>         /* getmem(), fatal_mksh(), errmsg() */
45 #include <mkstdmsil8n/mkstdmsil8n.h> /* libmkstdmsil8n_init() */
46 #include <sys/signal.h>        /* SIG_DFL */
47 #include <sys/stat.h>          /* open() */
48 #include <sys/wait.h>          /* wait() */
49 #include <ulimit.h>            /* ulimit() */
50 #include <unistd.h>            /* close(), dup2() */

```

54 /*

```

55 * Defined macros
56 */
64 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
65 #define SEND_MTOOL_MSG(cmds) \
66     if (send_mtool_msgs) { \
67         cmds \
68     }
69 #else
70 #define SEND_MTOOL_MSG(cmds)
71 #endif

59 /*
60 * typedefs & structs
61 */

63 /*
64 * Static variables
65 */

67 /*
68 * File table of contents
69 */
70 static Boolean  exec_vp(register char *name, register char **argv, char **envp,

72 /*
73 * Workaround for NFS bug. Sometimes, when running 'open' on a remote
74 * dmake server, it fails with "Stale NFS file handle" error.
75 * The second attempt seems to work.
76 */
77 int
78 my_open(const char *path, int oflag, mode_t mode) {
79     int res = open(path, oflag, mode);
80     if (res < 0 && (errno == ESTALE || errno == EAGAIN)) {
81         /* Stale NFS file handle. Try again */
82         res = open(path, oflag, mode);
83     }
84     return res;
85 }

_____ unchanged_portion_omitted

538 /*
539 *      await(ignore_error, silent_error, target, command, running_pid)
540 *
541 *      Wait for one child process and analyzes
542 *      the returned status when the child process terminates.
543 *
544 *      Return value:
545 *
546 *
547 *      Parameters:
548 *      ignore_error    Should we abort on error?
549 *      silent_error    Should error messages be suppressed for dmake?
550 *      target          The target we are building, for error msgs
551 *      command         The command we ran, for error msgs
552 *      running_pid     The pid of the process we are waiting for
553 *
554 *      Static variables used:
555 *      filter_file     The fd for the filter file
556 *      filter_file_name The name of the filter file
557 *
558 *      Global variables used:
559 *      filter_stderr   Set if -X is on
560 */

575 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
576 Boolean
577 await(register Boolean ignore_error, register Boolean silent_error, Name target,

```

```

578 #else
561 Boolean
562 await(register Boolean ignore_error, register Boolean silent_error, Name target,
581 #endif
563 {
564     int          status;
565     char         *buffer;
566     int          core_dumped;
567     int          exit_status;
587 #if defined(DISTRIBUTED) || defined(MAKE TOOL) /* tolik */
588     Avo_CmdOutput *make_output_msg;
589 #endif
568     FILE         *outfp;
569     register pid_t pid;
570     struct stat  stat_buff;
571     int          termination_signal;
572     char         tmp_buf[MAXPATHLEN];
595 #if defined(DISTRIBUTED) || defined(MAKE TOOL) /* tolik */
596     RWCollectable *xdr_msg;
597 #endif

574     while ((pid = wait(&status)) != running_pid) {
575         if (pid == -1) {
576             fatal_mksh(catgets(libmkdsmsil8n_catd, 1, 98, "wait() fa
577             }
578         }
579         (void) fflush(stdout);
580         (void) fflush(stderr);

582         if (status == 0) {

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

588             return succeeded;
589         }

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

596         exit_status = WEXITSTATUS(status);

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

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

605         /*
606          * If the child returned an error, we now try to print a
607          * nice message about it.
608          */
609         SEND_MTOOL_MSG(
610             make_output_msg = new Avo_CmdOutput();
611             (void) sprintf(tmp_buf, "%d", job_msg_id);
612             make_output_msg->appendOutput(strdup(tmp_buf));
613         );

615         tmp_buf[0] = (int) nul_char;
616         if (!silent_error) {
617             if (exit_status != 0) {
618                 (void) fprintf(stdout,

```

```

619             catgets(libmkdsmsil8n_catd, 1, 103, "****
620             exit_status);
621             SEND_MTOOL_MSG(
622                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
623                 catgets(libmkdsmsil8n_catd, 1, 10
624                 exit_status);
625             );
626         } else {
627             (void) fprintf(stdout,
628                 catgets(libmkdsmsil8n_catd, 1, 10
629                 termination_signal);
630             SEND_MTOOL_MSG(
631                 (void) sprintf(&tmp_buf[strlen(tmp_buf)]
632                 catgets(libmkdsmsil8n_cat
633                 termination_signal);
634             );
635             if (core_dumped) {
636                 (void) fprintf(stdout,
637                 catgets(libmkdsmsil8n_catd, 1, 10
638                 SEND_MTOOL_MSG(
639                 (void) sprintf(&tmp_buf[strlen(tmp_buf)]
640                 catgets(libmkdsmsil8n_cat
641                 );
642             }
643         }
644         if (ignore_error) {
645             (void) fprintf(stdout,
646                 catgets(libmkdsmsil8n_catd, 1, 109, " (ig
647             SEND_MTOOL_MSG(
648                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
649                 catgets(libmkdsmsil8n_catd, 1, 11
650             );
651         }
652         (void) fprintf(stdout, "\n");
653         (void) fflush(stdout);
654         SEND_MTOOL_MSG(
655             make_output_msg->appendOutput(strdup(tmp_buf));
656         );
657     }
658     SEND_MTOOL_MSG(
659         xdr_msg = (RWCollectable*) make_output_msg;
660         xdr(xdrs_p, xdr_msg);
661         delete make_output_msg;
662     );

664 #ifdef PRINT_EXIT_STATUS
665     warning_mksh(NOCATGETS("I'm in await(), returning failed.));
666 #endif

668     return failed;
669 }

```

unchanged portion omitted

```

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

1045 /*
1046 * We use a permanent buffer to reset SUNPRO_DEPENDENCIES value.
1047 */
1048 char *sunpro_dependencies_buf = NULL;
1049 char *sunpro_dependencies_oldbuf = NULL;
1050 int sunpro_dependencies_buf_size = 0;

1052 /*
1053 * setvar_daemon(name, value, append, daemon, strip_trailing_spaces)
1054 *
1055 * Set a macro value, possibly supplying a daemon to be used
1056 * when referencing the value.
1057 *
1058 * Return value:
1059 *           The property block with the new value
1060 *
1061 * Parameters:
1062 *   name       Name of the macro to set
1063 *   value      The value to set
1064 *   append     Should we reset or append to the current value?
1065 *   daemon     Special treatment when reading the value
1066 *   strip_trailing_spaces from the end of value->string
1067 *   debug_level Indicates how much tracing we should do
1068 *
1069 * Global variables used:
1070 *   makefile_type Used to check if we should enforce read only
1071 *   path_name      The Name "PATH", compared against
1072 *   virtual_root   The Name "VIRTUAL_ROOT", compared against
1073 *   vpath_defined  Set if the macro VPATH is set
1074 *   vpath_name     The Name "VPATH", compared against
1075 *   envvar        A list of environment vars with $ in value
1076 */
1077 Property
1078 setvar_daemon(register Name name, register Name value, Boolean append, Daemon da
1079 {
1080     register Property macro = maybe_append_prop(name, macro_prop);
1081     register Property macro_apx = get_prop(name->prop, macro_append_pr
1082     int length = 0;
1083     String_rec destination;
1084     wchar_t buffer[STRING_BUFFER_LENGTH];
1085     register Chain chain;
1086     Name val;
1087     wchar_t *val_string = (wchar_t*)NULL;
1088     Wstring wcb;

1091     if ((makefile_type != reading_nothing) &&
1092         macro->body.macro.read_only) {
1093         return macro;
1094     }
1095     /* Strip spaces from the end of the value */
1096     if (daemon == no_daemon) {
1097         if (value != NULL) {
1098             wcb.init(value);
1099             length = wcb.length();
1100             val_string = wcb.get_string();
1101         }
1102         if ((length > 0) && iswspace(val_string[length-1])) {
1103             INIT_STRING_FROM_STACK(destination, buffer);

```

```

1104         buffer[0] = 0;
1105         append_string(val_string, &destination, length);
1106         if (strip_trailing_spaces) {
1107             while ((length > 0) &&
1108                 iswspace(destination.buffer.start[length-
1109                     destination.buffer.start[--length] = 0;
1110             }
1111         }
1112         value = GETNAME(destination.buffer.start, FIND_LENGTH);
1113     }
1114 }
1115
1116 if (macro_apx != NULL) {
1117     val = macro_apx->body.macro_appendix.value;
1118 } else {
1119     val = macro->body.macro.value;
1120 }

1122 if (append) {
1123     /*
1124     * If we are appending, we just tack the new value after
1125     * the old one with a space in between.
1126     */
1127     INIT_STRING_FROM_STACK(destination, buffer);
1128     buffer[0] = 0;
1129     if ((macro != NULL) && (val != NULL)) {
1130         APPEND_NAME(val,
1131             &destination,
1132             (int) val->hash.length);
1133         if (value != NULL) {
1134             wcb.init(value);
1135             if (wcb.length() > 0) {
1136                 MBTOWC(wcs_buffer, " ");
1137                 append_char(wcs_buffer[0], &destination)
1138             }
1139         }
1140     }
1141     if (value != NULL) {
1142         APPEND_NAME(value,
1143             &destination,
1144             (int) value->hash.length);
1145     }
1146     value = GETNAME(destination.buffer.start, FIND_LENGTH);
1147     wcb.init(value);
1148     if (destination.free_after_use) {
1149         retmem(destination.buffer.start);
1150     }
1151 }

1153 /* Debugging trace */
1154 if (debug_level > 1) {
1155     if (value != NULL) {
1156         switch (daemon) {
1157             case chain_daemon:
1158                 (void) printf("%s =", name->string_mb);
1159                 for (chain = (Chain) value;
1160                     chain != NULL;
1161                     chain = chain->next) {
1162                     (void) printf(" %s", chain->name->string
1163                 }
1164                 (void) printf("\n");
1165                 break;
1166             case no_daemon:
1167                 (void) printf("%s= %s\n",
1168                     name->string_mb,
1169                     value->string_mb);

```

```

1170         break;
1171     } else {
1172         (void) printf("%s =\n", name->string_mb);
1173     }
1174 }
1175 }
1176 /* Set the new values in the macro property block */
1177 /**/
1178 if (macro_apx != NULL) {
1179     macro_apx->body.macro_appendix.value = value;
1180     INIT_STRING_FROM_STACK(destination, buffer);
1181     buffer[0] = 0;
1182     if (value != NULL) {
1183         APPEND_NAME(value,
1184                     &destination,
1185                     (int) value->hash.length);
1186         if (macro_apx->body.macro_appendix.value_to_append != NU
1187             MBTOWC(wcs_buffer, " ");
1188             append_char(wcs_buffer[0], &destination);
1189         }
1190     }
1191     if (macro_apx->body.macro_appendix.value_to_append != NULL) {
1192         APPEND_NAME(macro_apx->body.macro_appendix.value_to_appen
1193                     &destination,
1194                     (int) macro_apx->body.macro_appendix.value
1195                 }
1196     value = GETNAME(destination.buffer.start, FIND_LENGTH);
1197     if (destination.free_after_use) {
1198         retmem(destination.buffer.start);
1199     }
1200 }
1201 /**/
1202 macro->body.macro.value = value;
1203 macro->body.macro.daemon = daemon;
1204 /*
1205  * If the user changes the VIRTUAL_ROOT, we need to flush
1206  * the root package cache.
1207  */
1208 if (name == path_name) {
1209     flush_path_cache();
1210 }
1211 if (name == virtual_root) {
1212     flush_vroot_cache();
1213 }
1214 /* If this sets the VPATH we remember that */
1215 if ((name == vpath_name) &&
1216     (value != NULL) &&
1217     (value->hash.length > 0)) {
1218     vpath_defined = true;
1219 }
1220 /*
1221  * For environment variables we also set the
1222  * environment value each time.
1223  */
1224 if (macro->body.macro.exported) {
1225     static char *env;
1226 }
1227 #ifdef DISTRIBUTED
1228     if (!reading_environment && (value != NULL)) {
1229 #else
1230     if (!reading_environment && (value != NULL) && value->dollar) {
1231 #endif
1232         Envvar p;
1233
1234         for (p = envvar; p != NULL; p = p->next) {
1235             if (p->name == name) {

```

```

1232         p->value = value;
1233         p->already_put = false;
1234         goto found_it;
1235     }
1236 }
1237 p = ALLOC(Envvar);
1238 p->name = name;
1239 p->value = value;
1240 p->next = envvar;
1241 p->env_string = NULL;
1242 p->already_put = false;
1243 envvar = p;
1244 found_it:;
1245 #ifdef DISTRIBUTED
1246     if (reading_environment || (value == NULL) || !value->dollar) {
1247 #else
1248     } else {
1249 #endif
1250         length = 2 + strlen(name->string_mb);
1251         if (value != NULL) {
1252             length += strlen(value->string_mb);
1253         }
1254         Property env_prop = maybe_append_prop(name, env_mem_prop
1255         /*
1256          * We use a permanent buffer to reset SUNPRO_DEPENDENCIE
1257          */
1258         if (!strncmp(name->string_mb, NOCATGETS("SUNPRO_DEPENDEN
1259             if (length >= sunpro_dependencies_buf_size) {
1260                 sunpro_dependencies_buf_size=length*2;
1261                 if (sunpro_dependencies_buf_size < 4096)
1262                     sunpro_dependencies_buf_size = 4
1263                 if (sunpro_dependencies_buf)
1264                     sunpro_dependencies_oldbuf = sun
1265                     sunpro_dependencies_buf=getmem(sunpro_de
1266             }
1267             env = sunpro_dependencies_buf;
1268         } else {
1269             env = getmem(length);
1270         }
1271         env_alloc_num++;
1272         env_alloc_bytes += length;
1273         (void) sprintf(env,
1274                        "%s=%s",
1275                        name->string_mb,
1276                        value == NULL ?
1277                        "" : value->string_mb);
1278         (void) putenv(env);
1279         env_prop->body.env_mem.value = env;
1280         if (sunpro_dependencies_oldbuf) {
1281             /* Return old buffer */
1282             retmem_mb(sunpro_dependencies_oldbuf);
1283             sunpro_dependencies_oldbuf = NULL;
1284         }
1285     }
1286 }
1287 if (name == target_arch) {
1288     Name ha = getvar(host_arch);
1289     Name ta = getvar(target_arch);
1290     Name vr = getvar(virtual_root);
1291     int length;
1292     wchar_t *new_value;
1293     wchar_t *old_vr;
1294     Boolean new_value_allocated = false;
1295 }
1296 Wstring ha_str(ha);

```

```
1293     Wstring      ta_str(ta);
1294     Wstring      vr_str(vr);

1296     wchar_t * wcb_ha = ha_str.get_string();
1297     wchar_t * wcb_ta = ta_str.get_string();
1298     wchar_t * wcb_vr = vr_str.get_string();

1300     length = 32 +
1301             wslen(wcb_ha) +
1302             wslen(wcb_ta) +
1303             wslen(wcb_vr);
1304     old_vr = wcb_vr;
1305     MBSTOWCS(wcs_buffer, NOCATGETS("/usr/arch/"));
1306     if (IS_WEQUALN(old_vr,
1307                  wcs_buffer,
1308                  wslen(wcs_buffer))) {
1309         old_vr = (wchar_t *) wschr(old_vr, (int) colon_char) + 1
1310     }
1311     if ( (ha == ta) || (wslen(wcb_ta) == 0) ) {
1312         new_value = old_vr;
1313     } else {
1314         new_value = ALLOC_WC(length);
1315         new_value_allocated = true;
1316         WCSTOMBS(mbs_buffer, old_vr);
1317         (void) wsprintf(new_value,
1318                        NOCATGETS("/usr/arch/%s/%s:%s"),
1319                        ha->string_mb + 1,
1320                        ta->string_mb + 1,
1321                        mbs_buffer);
1322     }
1323     if (new_value[0] != 0) {
1324         (void) setvar_daemon(virtual_root,
1325                             GETNAME(new_value, FIND_LENGTH),
1326                             false,
1327                             no_daemon,
1328                             true,
1329                             debug_level);
1330     }
1331     if (new_value_allocated) {
1332         retmem(new_value);
1333     }
1334 }
1335 return macro;
1336 }
```

unchanged_portion_omitted


```

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

60 /*
61 * File table of contents
62 */
63 static void      change_sunpro_dependencies_value(char *oldpath, char *newpath);
64 static void      init_mksh_globals(char *shell);
65 static void      set_env_vars(char *env_list[]);

67 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
68 /*
69 *      Execute the command(s) of one Make or DMake rule
70 */
71 int
72 do_job(Avo_DmakeCommand *cmd_list[], char *env_list[], char *stdout_file, char *
73 {
74     Boolean          always_exec_flag;
75     char             *cmd;
76     Avo_DmakeCommand *cmd_list_p;
77     Name             command;
78     Boolean          do_not_exec_flag;
79     Boolean          ignore_flag;
80     int              length;
81     Boolean          make_refd_flag;
82     Boolean          meta_flag;
83     char             pathname[MAXPATHLEN];
84     Doname           result;
85     Boolean          silent_flag;
86     wchar_t         *tmp_wcs_buffer;

88     if ((childPid = fork()) < 0) { /* error */
89         ;
90     } else if (childPid > 0) { /* parent */
91         ;
92     } else { /* child, mksh */
93         (void) sigset(SIGCHLD, SIG_DFL);
94         enable_interrupt(handle_interrupt_mksh);
95         /* set environment variables */
96         set_env_vars(env_list);
97         /* redirect stdout and stderr to temp files */
98         dup2(1, 2); // Because fatal_mksh() prints error messages in
99                 // stderr but dmake uses stderr for XDR communic
100                // and stdout for errors messages.
101         redirect_io(stdout_file, stderr_file);
102         /* try cd'ing to cwd */
103         if (my_chdir(cwd) != 0) {
104             /* try the netpath machine:pathname */
105             if (!avo_netpath_to_path(cwd, pathname)) {
106                 fatal_mksh(catgets(libmkstdmsi18n_catd, 1, 137, "
107             } else if (my_chdir(pathname) != 0) {
108                 fatal_mksh(catgets(libmkstdmsi18n_catd, 1, 138, "
109             }
110             /*
111             * change the value of SUNPRO_DEPENDENCIES
112             * to the new path.
113             */
114             change_sunpro_dependencies_value(cwd, pathname);
115         }
116         init_mksh_globals(shell);
117         for (cmd_list_p = cmd_list;

```

```

118     *cmd_list_p != (Avo_DmakeCommand *) NULL;
119     cmd_list_p++) {
120         if ((*cmd_list_p)->ignore()) {
121             ignore_flag = true;
122         } else {
123             ignore_flag = false;
124         }
125         if ((*cmd_list_p)->silent()) {
126             silent_flag = true;
127         } else {
128             silent_flag = false;
129         }
130     /*
131     if ((*cmd_list_p)->always_exec()) {
132         always_exec_flag = true;
133     } else {
134         always_exec_flag = false;
135     }
136     */
137     always_exec_flag = false;
138     if ((*cmd_list_p)->meta()) {
139         meta_flag = true;
140     } else {
141         meta_flag = false;
142     }
143     if ((*cmd_list_p)->make_refd()) {
144         make_refd_flag = true;
145     } else {
146         make_refd_flag = false;
147     }
148     if ((*cmd_list_p)->do_not_exec()) {
149         do_not_exec_flag = true;
150     } else {
151         do_not_exec_flag = false;
152     }
153     do_not_exec_rule = do_not_exec_flag;
154     cmd = (*cmd_list_p)->getCmd();
155     if ((length = strlen(cmd)) >= MAXPATHLEN) {
156         tmp_wcs_buffer = ALLOC_WC(length + 1);
157         (void) mbstowcs(tmp_wcs_buffer, cmd, length + 1);
158         command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
159         retmem(tmp_wcs_buffer);
160     } else {
161         MBSTOWCS(wcs_buffer, cmd);
162         command = GETNAME(wcs_buffer, FIND_LENGTH);
163     }
164     if ((command->hash.length > 0) &&
165         (!silent_flag || do_not_exec_flag)) {
166         (void) printf("%s\n", command->string_mb);
167     }
168     result = dosys_mksh(command,
169         ignore_flag,
170         make_refd_flag,
171         false, /* bugs #4085164 & #4990057 */
172         /* BOOLEAN(silent_flag && ignore_flg
173         always_exec_flag,
174         (Name) NULL,
175         false,
176         NULL,
177         NULL,
178         vroot_path,
179         nice_prio);
180     if (result == build_failed) {

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

```

```
184 #endif
186         if (silent_flag) {
187             (void) printf(catgets(libmksdmsi18n_catd
188                 command->string_mb);
189         }
190         if (!ignore_flag && !ignore) {
192 #ifdef PRINT_EXIT_STATUS
193             warning_mksh(NOCATGETS("I'm in do_job()),
194 #endif
196             exit(1);
197         }
198     }
199 }
201 #ifdef PRINT_EXIT_STATUS
202     warning_mksh(NOCATGETS("I'm in do_job(), exiting 0."));
203 #endif
205     exit(0);
206 }
207 return childPid;
208 }
209 #endif /* TEAMWARE_MAKE_CMN */

68 static void
69 set_env_vars(char *env_list[])
70 {
71     char **env_list_p;
73     for (env_list_p = env_list;
74         *env_list_p != (char *) NULL;
75         env_list_p++) {
76         putenv(*env_list_p);
77     }
78 }
_____unchanged_portion_omitted_
```