

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

```
*****
4821 Wed May 20 11:37:20 2015
new/usr/src/cmd/make/bin/globals.cc
make: unifdef for SGE (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;

1

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

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 = txtl_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 **
124 * run any command */
126 Boolean no_parallel = false; /* TEAMWARE_MAKE_CMN */
127 #ifdef SGE_SUPPORT

2

```

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

```

```

189 /*
190  * File table of contents
191 */

```

```
*****
98530 Wed May 20 11:37:21 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for SGE (undefined)
*****
unchanged_portion_omitted_
156 #endif

158 extern Name normalize_name(register wchar_t *name_string, register i
160 extern int main(int, char * []);

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

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

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

189 jmp_buf jmpbuffer;
190 extern nl_catd catd;

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

```
216 main(int argc, char *argv[])
217 {
218     /*
219      * cp is a -> to the value of the MAKEFLAGS env var,
220      * which has to be regular chars.
221      */
222     register char *cp;
223     char make_state_dir[MAXPATHLEN];
224     Boolean parallel_flag = false;
225     char *programeptr;
226     char *slash_ptr;
227     mode_t um;
228     int i;
229 #ifdef TEAMWARE_MAKE_CMN
230     struct itimerval value;
231     char def_dmakerc_path[MAXPATHLEN];
232     Name dmake_name, dmake_name2;
233     Name dmake_value, dmake_value2;
234     Property prop, prop2;
235     struct stat statbuf;
236     int statval;
237 #endif
238
239 #ifndef PARALLEL
240     struct stat out_stat, err_stat;
241 #endif
242     hostid = gethostid();
243 #ifdef TEAMWARE_MAKE_CMN
244     avo_get_user(NULL, NULL); // Initialize user name
245 #endif
246     bsd_signals();
247
248     (void) setlocale(LC_ALL, "");

251 #ifdef DMAKE_STATISTICS
252     if (getenv(NOCATGETS("DMAKE_STATISTICS")) ) {
253         getname_stat = true;
254     }
255 #endif
256
257     /*
258      * avo_init() sets the umask to 0. Save it here and restore
259      * it after the avo_init() call.
260      */
261 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
262     um = umask(0);
263     avo_init(argv[0]);
264     umask(um);
265
266     cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
267 #endif
268
269 #if defined(TEAMWARE_MAKE_CMN)
270     catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
271     libcli_init();
272
273 #ifdef _CHECK_UPDATE_H
274     /* This is for dmake only (not for Solaris make).
275      * Check (in background) if there is an update (dmake patch)
276      * and inform user
277      */
278
279     {
280         Avo_err *err;
281         char *dir;
```

```

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

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

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

302 #ifndef TEAMWARE_MAKE_CMN
303     textdomain(NOCATGETS("SUNW_SPRO_MAKE"));
304#endif /* TEAMWARE_MAKE_CMN */

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

315 /*
316 * Set argv_zero_string to some form of argv[0] for
317 * recursive MAKE builds.
318 */
319
320 if (*argv[0] == (int) slash_char) {
321     /* argv[0] starts with a slash */
322     argv_zero_string = strdup(argv[0]);
323 } else if (strchr(argv[0], (int) slash_char) == NULL) {
324     /* argv[0] contains no slashes */
325     argv_zero_string = strdup(argv[0]);
326 } else {
327     /*
328      * argv[0] contains at least one slash,
329      * but doesn't start with a slash
330     */
331     char *tmp_current_path;
332     char *tmp_string;

334     tmp_current_path = get_current_path();
335     tmp_string = getmem(strlen(tmp_current_path) + 1 +
336                         strlen(argv[0]) + 1);
337     (void) sprintf(tmp_string,
338                   "%s/%s",
339                   tmp_current_path,
340                   argv[0]);
341     argv_zero_string = strdup(tmp_string);
342     retmem_mb(tmp_string);
343 }

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

```

```

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

357 /*
358 * look for last slash char in the path to look at the binary
359 * name. This is to resolve the hard link and invoke make
360 * in svr4 mode.
361 */

363 /* Sun OS make standard */
364 svr4 = false;
365 posix = false;
366 if (!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
367     svr4 = false;
368     posix = true;
369 } else {
370     programeptr = strrchr(argv[0], '/');
371     if (programeptr) {
372         programeptr++;
373     } else {
374         programeptr = argv[0];
375     }
376     if (!strcmp(programeptr, NOCATGETS("svr4.make"))) {
377         svr4 = true;
378         posix = false;
379     }
380 }
381 if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
382     svr4 = true;
383     posix = false;
384 }

386 /*
387 * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
388 */
389 char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"));
390 if (dmake_compat_mode_var != NULL) {
391     if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU")))
392         gnu_style = true;
393     //svr4 = false;
394     //posix = false;
395 }
396

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

```

```

414
415         }
416     }
417 }
418
419 #ifndef PARALLEL
420 /* find out if stdout and stderr point to the same place */
421 if (fstat(1, &out_stat) < 0) {
422     fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
423           "");
424 if (fstat(2, &err_stat) < 0) {
425     fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
426           ""));
427 if ((out_stat.st_dev == err_stat.st_dev) &&
428     (out_stat.st_ino == err_stat.st_ino)) {
429     stdout_stderr_same = true;
430 } else {
431     stdout_stderr_same = false;
432 }
433 #else
434     stdout_stderr_same = false;
435 #endif
436 /* Make the vroot package scan the path using shell semantics */
437 set_path_style(0);
438
439 setup_char_semantics();
440
441 setup_for_projectdir();
442
443 /*
444 * If running with .KEEP_STATE, curdir will be set with
445 * the connected directory.
446 */
447 (void) atexit(cleanup_after_exit);
448
449 load_cached_names();
450
451 /*
452 * Set command line flags
453 */
454 setup_makeflags_argv();
455 read_command_options(mf_argc, mf_argv);
456 read_command_options(argc, argv);
457 if (debug_level > 0) {
458     cp = getenv(makeflags->string_mb);
459     (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
460 }
461
462 setup_interrupt(handle_interrupt);
463
464 read_files_and_state(argc, argv);
465
466 #ifdef TEAMWARE_MAKE_CMN
467 /*
468 * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
469 */
470 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
471 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
472 prop2 = get_prop(dmake_name2->prop, macro_prop);
473 if (prop2 == NULL) {
474     /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
475     output_mode = txt1_mode;
476 } else {
477     dmake_value2 = prop2->body.macro.value;
478     if ((dmake_value2 == NULL) ||
479         (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {

```

```

480             output_mode = txt1_mode;
481         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
482             output_mode = txt2_mode;
483         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1")))
484             output_mode = html1_mode;
485         } else {
486             warning(catgets(catd, 1, 352, "Unsupported value '%s' fo
487                             dmake_value2->string_mb));
488         }
489     }
490     /*
491      * Find the dmake_mode: distributed, parallel, or serial.
492     */
493     if ((!pmake_cap_r_specified) &&
494         (!pmake_machinesfile_specified)) {
495         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
496         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
497         prop2 = get_prop(dmake_name2->prop, macro_prop);
498         if (prop2 == NULL) {
499             /* DMAKE_MODE not defined, default to distributed mode */
500             dmake_mode_type = distributed_mode;
501             no_parallel = false;
502         } else {
503             dmake_value2 = prop2->body.macro.value;
504             if ((dmake_value2 == NULL) ||
505                 (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed")))
506                 dmake_mode_type = distributed_mode;
507                 no_parallel = false;
508             } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel"))
509                 dmake_mode_type = parallel_mode;
510                 no_parallel = false;
511 #ifdef SGE_SUPPORT
512             grid = false;
513         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("grid")))
514             dmake_mode_type = parallel_mode;
515             no_parallel = false;
516             grid = true;
517         }
518         if ((!list_all_targets) &&
519             (report_dependencies_level == 0)) {
520             /*
521              * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
522              * They could be defined in the env, in the makefile, or on the
523              * command line.
524              * If neither is defined, and $(HOME)/.dmakerc does not exists,
525              * then print a message, and default to parallel mode.
526             */
527 #ifdef DISTRIBUTED
528             MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
529             dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
530             MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
531             dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
532             if (((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
533                 ((dmake_value = prop->body.macro.value) == NULL)) &&
534                 ((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
535                 ((dmake_value2 = prop2->body.macro.value) == NULL))) {
536                     Boolean empty_dmakerc = true;
537                     char *homedir = getenv(NOCATGETS("HOME"));
538

```

```

539
540     if ((homemedir != NULL) && (strlen(homedir) < (sizeof(def_
541         .sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
542         if (((statval = stat(def_dmakerc_path, &statbuf
543             ((statval == 0) && (statbuf.st_size == 0
544         } else {
545             Avo_dmakerc     *rcfile = new Avo_dmaker
546             Avo_err          *err = rcfile->read(def_
547             if (err) {
548                 fatal(err->str);
549             }
550             empty_dmakerc = rcfile->was_empty();
551             delete rcfile;
552         }
553         if (empty_dmakerc) {
554             if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
555                 putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
556                 (void) fprintf(stdout, catgets(catd, 1,
557                     (void) fprintf(stdout, catgets(catd, 1,
558
559                 dmake_mode_type = parallel_mode;
560                 no_parallel = false;
561             }
562         }
563 #else
564     if(dmake_mode_type == distributed_mode) {
565         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
566         (void) fprintf(stdout, NOCATGETS("      Defaulting to p
567         dmake_mode_type = parallel_mode;
568         no_parallel = false;
569     }
570 #endif /* DISTRIBUTED */
571 }
572 }
573 #endif

575 #ifdef TEAMWARE_MAKE_CMN
576     parallel_flag = true;
577     /* XXX - This is a major hack for DMake/Licensing. */
578     if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
579         if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
580             /*
581             * If the user can not get a TeamWare license,
582             * default to serial mode.
583             */
584             dmake_mode_type = serial_mode;
585             no_parallel = true;
586         } else {
587             putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
588         }
589         start_time = time(NULL);
590         /*
591             * XXX - Hack to disable SIGALRM's from licensing library's
592             *       setitimer().
593             */
594         value.it_interval.tv_sec = 0;
595         value.it_interval.tv_usec = 0;
596         value.it_value.tv_sec = 0;
597         value.it_value.tv_usec = 0;
598         (void) setitimer(ITIMER_REAL, &value, NULL);
599     }
600 /**
601 // If dmake is running with -t option, set dmake_mode_type to serial.
602 // This is done because doname() calls touch_command() that runs serially.
603 // If we do not do that, maketool will have problems.

```

```

605 //
606     if(touch) {
607         dmake_mode_type = serial_mode;
608         no_parallel = true;
609     }
610 #else
611     parallel_flag = false;
612 #endif

614 #if defined (TEAMWARE_MAKE_CMN) && defined(RDIRECT_ERR)
615     /*
616         * Check whether stdout and stderr are physically same.
617         * This is in order to decide whether we need to redirect
618         * stderr separately from stdout.
619         * This check is performed only if __DMAKE_SEPARATE_STDERR
620         * is not set. This variable may be used in order to preserve
621         * the 'old' behaviour.
622         */
623     out_err_same = true;
624     char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
625     if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
626         struct stat stdout_stat;
627         struct stat stderr_stat;
628         if( (fstat(1, &stdout_stat) == 0)
629             && (fstat(2, &stderr_stat) == 0)
630             {
631                 if( (stdout_stat.st_dev != stderr_stat.st_dev)
632                     || (stdout_stat.st_ino != stderr_stat.st_ino) )
633                     {
634                         out_err_same = false;
635                     }
636             }
637         }
638 #endif

640 #ifdef DISTRIBUTED
641     /*
642         * At this point, DMake should startup an rxm with any and all
643         * DMake command line options. Rxm will, among other things,
644         * read the rc file.
645         */
646     if ((!list_all_targets) &&
647         (report_dependencies_level == 0) &&
648         (dmake_mode_type == distributed_mode)) {
649         startup_rxm();
650     }
651 #endif
652
653 /*
654 * Enable interrupt handler for alarms
655 */
656 (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

658 /*
659 * Check if make should report
660 */
661 if (getenv(sunpro_dependencies->string_mb) != NULL) {
662     FILE *report_file;

664     report_dependency("");
665     report_file = get_report_file();
666     if ((report_file != NULL) && (report_file != (FILE*)-1)) {
667         (void) fprintf(report_file, "\n");
668     }
669 }


```

```

671 /*
672 *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
673 */
674 if (keep_state) {
675     maybe_append_prop(sunpro_dependencies, macro_prop)->
676         body.macro.exported = true;
677 } else {
678     maybe_append_prop(sunpro_dependencies, macro_prop)->
679         body.macro.exported = false;
680 }
681
682 working_on_targets = true;
683 if (trace_status) {
684     dump_make_state();
685     fclose(stdout);
686     fclose(stderr);
687     exit_status = 0;
688     exit(0);
689 }
690 if (list_all_targets) {
691     dump_target_list();
692     fclose(stdout);
693     fclose(stderr);
694     exit_status = 0;
695     exit(0);
696 }
697 trace_reader = false;
698
699 /*
700 * Set temp_file_directory to the directory the .make.state
701 * file is written to.
702 */
703 if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NULL)
704     temp_file_directory = strdup(get_current_path());
705 } else {
706     *slash_ptr = (int) nul_char;
707     (void) strcpy(make_state_dir, make_state->string_mb);
708     *slash_ptr = (int) slash_char;
709     /* when there is only one slash and it's the first
710      ** character, make_state_dir should point to '/'.
711      */
712     if(make_state_dir[0] == '\0') {
713         make_state_dir[0] = '/';
714         make_state_dir[1] = '\0';
715     }
716     if (make_state_dir[0] == (int) slash_char) {
717         temp_file_directory = strdup(make_state_dir);
718     } else {
719         char    tmp_current_path2[MAXPATHLEN];
720
721         (void) sprintf(tmp_current_path2,
722                         "%s/%s",
723                         get_current_path(),
724                         make_state_dir);
725         temp_file_directory = strdup(tmp_current_path2);
726     }
727 }
728
729 #ifdef DISTRIBUTED
730     building_serial = false;
731 #endif
732
733     report_dir_enter_leave(true);
734
735     make_targets(argc, argv, parallel_flag);

```

```

737     report_dir_enter_leave(false);
738
739     if (build_failed_ever_seen) {
740         if (posix) {
741             exit_status = 1;
742         }
743         exit(1);
744     }
745     exit_status = 0;
746     exit(0);
747     /* NOTREACHED */
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
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2
```

```
new/usr/src/cmd/make/bin/main.cc
1002             kill (-getpid(), SIGTERM);
1003     }
1004 #ifdef TEAMWARE_MAKE_CMN
1005     /* Clean up all parallel/distributed children already finished */
1006     finish_children(false);
1007 #endif
1008
1009     /* Make sure the processes running under us terminate first */
1010
1011     while (wait((int *) NULL) != -1);
1012     /* Delete the current targets unless they are precious */
1013     if ((current_target != NULL) &&
1014         current_target->is_member &&
1015         ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
1016         current_target = member->body.member.library;
1017     }
1018     if (!do_not_exec_rule &&
1019         !touch &&
1020         !quest &&
1021         (current_target != NULL) &&
1022         !(current_target->stat.is_precious || all_precious)) {
1023
1024 /* BID_1030811 */
1025 /* azv 16 Oct 95 */
1026         current_target->stat.time = file_no_time;
1027
1028         if (exists(current_target) != file_doesnt_exist) {
1029             (void) fprintf(stderr,
1030                           "\n*** %s ",
1031                           current_target->string_mb);
1032             if (current_target->stat.is_dir) {
1033                 (void) fprintf(stderr,
1034                               catgets(catd, 1, 168, "not remove");
1035                               current_target->string_mb);
1036             } else if (unlink(current_target->string_mb) == 0) {
1037                 (void) fprintf(stderr,
1038                               catgets(catd, 1, 169, "removed.\n");
1039                               current_target->string_mb);
1040             } else {
1041                 (void) fprintf(stderr,
1042                               catgets(catd, 1, 170, "could not");
1043                               current_target->string_mb,
1044                               errmsg(errno));
1045             }
1046         }
1047     }
1048     for (rp = running_list; rp != NULL; rp = rp->next) {
1049         if (rp->state != build_running) {
1050             continue;
1051         }
1052         if (rp->target->is_member &&
1053             ((member = get_prop(rp->target->prop, member_prop)) != NULL)) {
1054             rp->target = member->body.member.library;
1055         }
1056         if (!do_not_exec_rule &&
1057             !touch &&
1058             !quest &&
1059             !(rp->target->stat.is_precious || all_precious)) {
1060
1061             rp->target->stat.time = file_no_time;
1062             if (exists(rp->target) != file_doesnt_exist) {
1063                 (void) fprintf(stderr,
1064                               "\n*** %s ",
1065                               rp->target->string_mb);
1066             }
1067         }
1068     }
1069 }
```

```
new/usr/src/cmd/make/bin/main.cc          (void) fprintf(stderr,
1068                                         catgets(catd, 1, 171, "no
1069                                         rp->target->string_mb);
1070 } else if (unlink(rp->target->string_mb) == 0) {
1071     (void) fprintf(stderr,
1072                     catgets(catd, 1, 172, "re
1073                     rp->target->string_mb);
1074     } else {
1075         (void) fprintf(stderr,
1076                         catgets(catd, 1, 173, "co
1077                         rp->target->string_mb,
1078                         errmsg(errno));
1079     }
1080 }
1081 }
1082 }
1083 }

1092 #ifdef SGE_SUPPORT
1093     /* Remove SGE script file */
1094     if (grid) {
1095         unlink(script_file);
1096     }
1097 #endif

1086     /* Have we locked .make.state or .nse_depinfo? */
1087     if ((make_state_lockfile != NULL) && (make_state_locked)) {
1088         unlink(make_state_lockfile);
1089         make_state_lockfile = NULL;
1090         make_state_locked = false;
1091     }
1092     /*
1093      * Re-read .make.state file (it might be changed by recursive make)
1094      */
1095     check_state(NULL);

1097     report_dir_enter_leave(false);

1099     exit_status = 2;
1100     exit(2);
1101 }



---

unchanged portion omitted
```

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

```
*****
```

```
53764 Wed May 20 11:37:22 2015
```

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

```
make: unifdef for SGE (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 #ifdef 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 <avo/util.h> /* avo_get_user(), avo_hostname() */
45 #include <mk/defs.h>
46 #include <mksh/dosys.h> /* redirect_io() */
47 #include <mksh/macro.h> /* expand_value() */
48 #include <mksh/misc.h>
49 #include <sys/signal.h>
50 #include <sys/stat.h>
51 #include <sys/types.h>
52 #include <sys/utsname.h>
53 #include <sys/wait.h>
54 #include <unistd.h>
```

```
56 #ifdef SGE_SUPPORT
57 #include <dmthread/Avo_PathNames.h>
58 #endif
```

```
58 /*
```

```
1
```

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

```
59 * Defined macros
60 */
61 #define MAXRULES 100
62
63 /*
64 * This const should be in avo_dms/include/AvoDmakeCommand.h
65 */
66 const int local_host_mask = 0x20;
```

```
69 /*
70 * typedefs & structs
71 */
```

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

```
89 /*
90 * File table of contents
91 */
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 Dname 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_dname(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);
```

```
111 /*
112 * execute_parallel(line, waitflg)
113 *
114 * DMake 2.x:
115 * parallel mode: spawns a parallel process to execute the command group.
116 * distributed mode: sends the command group down the pipe to rxm.
117 *
118 * Return value:
119 * The result of the execution
120 *
121 * Parameters:
122 * line The command group to execute
123 */
124 Dname
```

```
2
```

```

125 execute_parallel(Property line, Boolean waitflg, Boolean local)
126 {
127     int             argcnt;
128     int             cmd_options = 0;
129     char            *commands[MAXRULES + 5];
130     char            *cp;
131 #ifdef DISTRIBUTED
132     Avo_DoJobMsg  *dmake_job_msg = NULL;
133 #endif
134     Name            dmake_name;
135     Name            dmake_value;
136     int             ignore;
137     Name            make_machines_name;
138     char            **p;
139     Property        prop;
140     Name            result = build_ok;
141     Cmd_line        rule;
142     Boolean         silent_flag;
143     Name            target = line->body.line.target;
144     Boolean         wrote_state_file = false;

145     if ((pmake_max_jobs == 0) &
146         (dmake_mode_type == parallel_mode)) {
147         if (user_name[0] == '\0') {
148             avo_get_user(user_name, NULL);
149         }
150         if (local_host[0] == '\0') {
151             strcpy(local_host, avo_hostname());
152         }
153         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
154         dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
155         if ((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
156             ((dmake_value = prop->body.macro.value) != NULL)) {
157             pmake_max_jobs = atoi(dmake_value->string_mb);
158             if (pmake_max_jobs <= 0) {
159                 warning(catgets(catd, 1, 308, "DMAKE_MAX_JOBS ca
160                 warning(catgets(catd, 1, 309, "setting DMAKE_MAX
161                 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
162             }
163         } else {
164             /*
165             * For backwards compatibility w/ PMake 1.x, when
166             * DMake 2.x is being run in parallel mode, DMake
167             * should parse the PMake startup file
168             * $($HOME)/.make.machines to get the pmake_max_jobs.
169             */
170             MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
171             dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
172             if (((prop = get_prop(dmake_name->prop, macro_prop)) != NULL) &&
173                 ((dmake_value = prop->body.macro.value) != NULL)) {
174                 make_machines_name = dmake_value;
175             } else {
176                 make_machines_name = NULL;
177             }
178             if ((pmake_max_jobs = read_make_machines(make_machines_n
179                 pmake_max_jobs = PMAKE_DEF_MAX_JOBS;
180             }
181         }
182     }
183 #ifdef DISTRIBUTED
184     if (send_mtool_msgs) {
185         send_rsrc_info_msg(pmake_max_jobs, local_host, user_name
186     }
187 #endif
188 }
189 if ((dmake_mode_type == serial_mode) ||

```

```

190
191         ((dmake_mode_type == parallel_mode) && (waitflg))) {
192             return (execute_serial(line));
193         }
194 #ifdef DISTRIBUTED
195     if (dmake_mode_type == distributed_mode) {
196         if (local) {
197             return (execute_serial(line));
198         }
199         waitflg = true;
200     }
201     dmake_job_msg = new Avo_DoJobMsg();
202     dmake_job_msg->setJobId(++job_msg_id);
203     dmake_job_msg->setTarget(target->string_mb);
204     dmake_job_msg->setImmediateOutput(0);
205     called_make = false;
206     } else
207 #endif
208     {
209         p = commands;
210     }

211     argcnt = 0;
212     for (rule = line->body.line.command_used;
213         rule != NULL;
214         rule = rule->next) {
215         if (posix && (touch || quest) && !rule->always_exec) {
216             continue;
217         }
218         if (vpath_defined) {
219             rule->command_line =
220                 vpath_transformation(rule->command_line);
221         }
222         if (dmake_mode_type == distributed_mode) {
223             cmd_options = 0;
224             if (local) {
225                 cmd_options |= local_host_mask;
226             }
227         } else {
228             silent_flag = false;
229             ignore = 0;
230         }
231         if (rule->command_line->hash.length > 0) {
232             if (++argcnt == MAXRULES) {
233                 if (dmake_mode_type == distributed_mode) {
234                     /* XXX - tell rxm to execute on local ho
235                     /* I WAS HERE!!! */
236                 } else {
237                     /* Too many rules, run serially instead.
238                     return build_serial;
239                 }
240             }
241         }
242 #ifdef DISTRIBUTED
243         if (dmake_mode_type == distributed_mode) {
244             /*
245             * XXX - set assign_mask to tell rxm
246             * to do the following.
247             */
248             /* From execute_serial():
249             */
250             if (rule->assign) {
251                 result = build_ok;
252                 do_assign(rule->command_line, target);
253             }
254             if (0) {
255             } else if (report_dependencies_level == 0) {
256                 if (rule->ignore_error) {
257                     cmd_options |= ignore_mask;
258                 }
259             }
260         }
261     }
262 }

```

```

257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287 #endif
288 {
289     if (rule->silent && !silent) {
290         silent_flag = true;
291     }
292     if (rule->ignore_error) {
293         ignore++;
294     }
295     /* XXX - need to add support for + prefix */
296     if (silent_flag || ignore) {
297         *p = getmem((silent_flag ? 1 : 0) +
298                     ignore +
299                     (strlen(rule->
300                             command_line->
301                             string_mb)) +
302                     1);
303     cp = *p++;
304     if (silent_flag) {
305         *cp++ = (int) at_char;
306     }
307     if (ignore) {
308         *cp++ = (int) hyphen_char;
309     }
310     (void) strcpy(cp, rule->command_line->st
311 ) else {
312         *p++ = rule->command_line->string_mb;
313     }
314 }
315
316     if ((argcnt == 0) ||
317         (report_dependencies_level > 0)) {
318 #ifdef DISTRIBUTED
319     if (dmake_job_msg) {
320         delete dmake_job_msg;
321     }

```

```

322 #endif
323             return build_ok;
324         }
325     }
326 #ifdef DISTRIBUTED
327     if (dmake_mode_type == distributed_mode) {
328         // Send a DoJob message to the rxm process.
329         distribute_rxm(dmake_job_msg);
330
331         // Wait for an acknowledgement.
332         Avo_AcknowledgeMsg *ackMsg = getAcknowledgeMsg();
333         if (ackMsg) {
334             delete ackMsg;
335         }
336
337         if (waitflg) {
338             // Wait for, and process a job result.
339             result = await_dist(waitflg);
340             if (called_make) {
341                 maybe_reread_make_state();
342             }
343             check_state(temp_file_name);
344             if (result == build_failed) {
345                 if (!continue_after_error) {
346
347 #ifdef PRINT_EXIT_STATUS
348             warning(NOCATGETS("I'm in execute_parallel
349 #endif
350
351             fatal(catgets(catd, 1, 252, "Command fai
352                         target->string_mb));
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386 #endif
387
388         { *p = NULL;

```

```

390     Doname res = distribute_process(commands, line);
391     if (res == build_running) {
392         parallel_process_cnt++;
393     }
394     /*
395      * Return only those memory that were specially allocated
396      * for part of commands.
397     */
398     for (int i = 0; commands[i] != NULL; i++) {
399         if ((commands[i][0] == (int) at_char) ||
400             (commands[i][0] == (int) hyphen_char)) {
401             retmem_mb(commands[i]);
402         }
403     }
404     return res;
405 }
406 }
407 }



---



unchanged_portion_omitted_


722 #endif /* MAXJOBS_ADJUST_RFE4694000 */
723 #endif /* TEAMWARE_MAKE_CMN */

725 /*
726  * distribute_process(char **commands, Property line)
727  *
728  * Parameters:
729  *     commands      argv vector of commands to execute
730  *
731  * Return value:
732  *     The result of the execution
733  *
734  * Static variables used:
735  *     process_running Set to the pid of the process set running
736  * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
737  *     job_adjust_mode Current job adjust mode
738  * #endif
739 */
740 static Doname
741 distribute_process(char **commands, Property line)
742 {
743     static unsigned file_number = 0;
744     wchar_t        string[MAXPATHLEN];
745     char          mbstring[MAXPATHLEN];
746     int           filed;
747     int           res;
748     int           tmp_index;
749     char          *tmp_index_str_ptr;

751 #if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
752     while (parallel_process_cnt >= pmake_max_jobs) {
753         await_parallel(false);
754         finish_children(true);
755     }
756 #else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
757     /* initialize adjust mode, if not initialized */
758     if (job_adjust_mode == ADJUST_UNKNOWN) {
759         job_adjust_init();
760     }
761     /* actions depend on adjust mode */
762     switch (job_adjust_mode) {
763     case ADJUST_M1:
764         while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
765                         await_parallel(false));

```

```

767                     finish_children(true);
768                 }
769                 break;
770             case ADJUST_M2:
771                 if ((res = m2_acquire_job()) == 0) {
772                     if (parallel_process_cnt > 0) {
773                         await_parallel(false);
774                         finish_children(true);
775                     }
776                     if ((res = m2_acquire_job()) == 0) {
777                         return build_serial;
778                     }
779                 } else {
780                     return build_serial;
781                 }
782             }
783             if (res < 0) {
784                 /* job adjustment error */
785                 job_adjust_error();
786             }
787             /* no adjustment */
788             while (parallel_process_cnt >= pmake_max_jobs) {
789                 await_parallel(false);
790                 finish_children(true);
791             }
792         }
793         break;
794     default:
795         while (parallel_process_cnt >= pmake_max_jobs) {
796             await_parallel(false);
797             finish_children(true);
798         }
799     }
800 #endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
801 #ifdef DISTRIBUTED
802     if (send_mtool_msgs) {
803         send_job_start_msg(line);
804     }
805 #endif
806 #ifdef DISTRIBUTED
807     setvar_envvar((Avo_DoJobMsg *)NULL);
808 #else
809     setvar_envvar();
810 #endif
811     /*
812      * Tell the user what DMake is doing.
813      */
814     if (!silent && output_mode != txt2_mode) {
815         /*
816          * Print local_host --> x job(s).
817          */
818         (void) fprintf(stdout,
819                         catgets(catd, 1, 325, "%s --> %d %s\n"),
820                         local_host,
821                         parallel_process_cnt + 1,
822                         (parallel_process_cnt == 0) ? catgets(catd, 1, 12
823
824             /* Print command line(s). */
825             tmp_index = 0;
826             while (commands[tmp_index] != NULL) {
827                 /* No @ char. */
828                 /* XXX - need to add [2] when + prefix is added */
829                 if ((commands[tmp_index][0] != (int) at_char) &&
830                     (commands[tmp_index][1] != (int) at_char)) {
831                     tmp_index_str_ptr = commands[tmp_index];
832                     if (*tmp_index_str_ptr == (int) hyphen_char) {

```

```

833         }
834         tmp_index_str_ptr++;
835         (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
836     }
837     tmp_index++;
838 }
839 (void) fflush(stdout);
840 }

842 (void) sprintf(mbstring,
843                 NOCATGETS("%s/dmake.stdout.%d.%d.XXXXXX"),
844                 tmpdir,
845                 getpid(),
846                 file_number++);
847
848 mktemp(mbstring);

849 stdout_file = strdup(mbstring);
850 stderr_file = NULL;
851 #if defined(TEAMWARE_MAKE_CMN) && defined(RDIRECT_ERR)
852 if (!out_err_same) {
853     (void) sprintf(mbstring,
854                 NOCATGETS("%s/dmake.stderr.%d.%d.XXXXXX"),
855                 tmpdir,
856                 getpid(),
857                 file_number++);
858
859 mktemp(mbstring);
860
861 stderr_file = strdup(mbstring);
862 }
863 #endif

864 #ifdef SGE_SUPPORT
865     if (grid) {
866         static char *dir4gridscripts = NULL;
867         static char *hostName = NULL;
868         if (dir4gridscripts == NULL) {
869             Name dmakeOdir_name, dmakeOdir_value;
870             Property prop;
871             MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
872             dmakeOdir_name = GETNAME(wcs_buffer, FIND_LENGTH);
873             if ((prop = get_prop(dmakeOdir_name->prop, macro_prop))
874                 (&(dmakeOdir_value = prop->body.macro.value)) != NULL)
875                 dir4gridscripts = dmakeOdir_value->string_mb;
876         }
877         dir4gridscripts = Avo_PathNames::pathname_output_directory;
878         hostName = Avo_PathNames::pathname_local_host();
879     }
880     (void) sprintf(script_file,
881                 NOCATGETS("%s/dmake.script.%s.%d.%d.XXXXXX"),
882                 dir4gridscripts,
883                 hostName,
884                 getpid(),
885                 file_number++);
886 }
887 #endif /* SGE_SUPPORT */
888 process_running = run_rule_commands(local_host, commands);
889
890 return build_running;
891 }
892 unchanged_portion_omitted_
893
894 /*
895 * This function replaces the makesh binary.
896 */

```

```

2025
2026 #ifdef SGE_SUPPORT
2027 #define DO_CHECK(f)      if (f <= 0) { \
2028                             fprintf(stderr, \
2029                                 catgets(catd, 1, 347, "Could not write to t \
2030                                         script_file, errmsg(errno)); \
2031                             _exit(1); \
2032                         }
2033 #endif /* SGE_SUPPORT */

2034 static pid_t
2035 run_rule_commands(char *host, char **commands)
2036 {
2037     Boolean always_exec;
2038     Name command;
2039     Boolean ignore;
2040     int length;
2041     Dename result;
2042     Boolean silent_flag;
2043
2044 #ifdef SGE_SUPPORT
2045     wchar_t *wcmd, *tmp_wcs_buffer = NULL;
2046     char *cmd, *tmp_mbs_buffer = NULL;
2047     FILE *scrpf;
2048     Name shell = getvar(shell_name);
2049     wchar_t *tmp_wcs_buffer;
2050 #endif /* SGE_SUPPORT */

2051     childPid = fork();
2052     switch (childPid) {
2053     case -1: /* Error */
2054         fatal(catgets(catd, 1, 337, "Could not fork child process for dm \
2055                                         errmsg(errno)); \
2056         break;
2057     case 0: /* Child */
2058         /* To control the processed targets list is not the child's busi \
2059         running_list = NULL;
2060
2061 #if defined(RDIRECT_ERR)
2062         if(out_err_same) {
2063             redirect_io(stdout_file, (char*)NULL);
2064         } else {
2065             redirect_io(stdout_file, stderr_file);
2066         }
2067 #endif
2068 #ifdef SGE_SUPPORT
2069         if (grid) {
2070             int fdes = mkstemp(script_file);
2071             if ((fdes < 0) || (scrpf = fdopen(fdes, "w")) == NULL) {
2072                 fprintf(stderr,
2073                     catgets(catd, 1, 341, "Could not create \
2074                                         script_file, errmsg(errno)); \
2075                     _exit(1);
2076             }
2077             if (IS_EQUAL(shell->string_mb, ""))
2078                 shell = shell_name;
2079         }
2080 #endif /* SGE_SUPPORT */
2081         for (commands = commands;
2082              (*commands != (char *)NULL);
2083              commands++) {
2084             silent_flag = silent;
2085             ignore = false;
2086             always_exec = false;
2087         }
2088     }
2089 }

```

```

2062
2063     while (((**commands == (int) at_char) ||
2064             (**commands == (int) hyphen_char) ||
2065             (**commands == (int) plus_char)) {
2066         if (**commands == (int) at_char) {
2067             silent_flag = true;
2068         }
2069         if (**commands == (int) hyphen_char) {
2070             ignore = true;
2071         }
2072         if (**commands == (int) plus_char) {
2073             always_exec = true;
2074         }
2075         (*commands)++;
2076     }
2132 #ifdef SGE_SUPPORT
2133     if (grid) {
2134         if ((length = strlen(*commands)) >= MAXPATHLEN /
2135             wcmd = tmp_wcs_buffer = ALLOC_WC(length
2136             (void) mbstowcs(tmp_wcs_buffer, *command
2137             ) else {
2138                 MBSTOWCS(wcs_buffer, *commands);
2139                 wcs_buffer;
2140                 cmd = mbs_buffer;
2141             }
2142             wchar_t *from = wcmb + wslen(wcmb);
2143             wchar_t *to = from + (from - wcmb);
2144             *to = (int) nul_char;
2145             while (from > wcmb) {
2146                 --to = --from;
2147                 if (*from == (int) newline_char) { // ne
2148                     *to = *--from;
2149                 } else if (wschr(char_semantics_char, *f
2150                     *to = (int) backslash_char;
2151                 }
2152             if (length >= MAXPATHLEN*MB_LEN_MAX/2) { // size
2153                 cmd = tmp_mbs_buffer = getmem((length *
2154                     (void) wcstombs(tmp_mbs_buffer, to, (len
2155             ) else {
2156                 WCSTOMB(mbs_buffer, to);
2157                 cmd = mbs_buffer;
2158             }
2159             char *mbst, *mbend;
2160             if ((length > 0) &&
2161                 !silent_flag) {
2162                 for (mbst = cmd; (mbend = strstr(mbst, "
2163                     *mbend = '\0';
2164                     DO_CHECK(fprintf(scrfp, NOCATGET
2165                     *mbend = '\\';
2166                 }
2167                 DO_CHECK(fprintf(scrfp, NOCATGETS("/usr/
2168             )
2169             if (!do_not_exec_rule ||
2170                 !working_on_targets ||
2171                 always_exec) {
2172                 DO_CHECK(fprintf(scrfp, NOCATGETS("%s -c
2173                 DO_CHECK(fputs(NOCATGETS("__DMAKECMDEXIT
2174                 if (ignore) {
2175                     DO_CHECK(fprintf(scrfp, NOCATGET
2176                         catgets(catd, 1, 343, "\\
2177                         catgets(catd, 1, 344, "("
2178             ) else {
2179                 DO_CHECK(fprintf(scrfp, NOCATGET
2180                     catgets(catd, 1, 342, "\\
2181             )
2182             if (silent_flag) {

```

```

2184     DO_CHECK(fprintf(scrfp, NOCATGET
2185         catgets(catd, 1, 345, "T
2186         for (mbst = cmd; (mbend = strstr
2187             *mbend = '\0';
2188             DO_CHECK(fprintf(scrfp,
2189                 *mbend = '\\';
2190             )
2191             DO_CHECK(fprintf(scrfp, NOCATGET
2192             )
2193             if (!ignore) {
2194                 DO_CHECK(fputs(NOCATGETS("\texit
2195             )
2196             DO_CHECK(fputs(NOCATGETS("fi\n"), scrfp)
2197             )
2198             if (tmp_wcs_buffer) {
2199                 retmem_mb(tmp_mbs_buffer);
2200                 tmp_mbs_buffer = NULL;
2201             }
2202             if (tmp_wcs_buffer) {
2203                 retmem(tmp_wcs_buffer);
2204                 tmp_wcs_buffer = NULL;
2205             }
2206             continue;
2207         )
2208     #endif /* SGE_SUPPORT */
2209     if ((length = strlen(*commands)) >= MAXPATHLEN) {
2210         tmp_wcs_buffer = ALLOC_WC(length + 1);
2211         (void) mbstowcs(tmp_wcs_buffer, *commands, leng
2212         command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2213         retmem(tmp_wcs_buffer);
2214     } else {
2215         MBSTOWCS(wcs_buffer, *commands);
2216         command = GETNAME(wcs_buffer, FIND_LENGTH);
2217     }
2218     if ((command->hash.length > 0) &&
2219         !silent_flag) {
2220         (void) printf("%s\n", command->string_mb);
2221     }
2222     result = dosys(command,
2223         ignore,
2224         false,
2225         false, /* bugs #4085164 & #4990057 */
2226         /* BOOLEAN(silent_flag && ignore), */
2227         always_exec,
2228         (Name) NULL,
2229         false);
2230     if (result == build_failed) {
2231         if (silent_flag) {
2232             (void) printf(catgets(catd, 1, 152, "The
2233         }
2234         if (!ignore) {
2235             _exit(1);
2236         }
2237     }
2238 }
2239 #ifndef SGE_SUPPORT
2240     _exit(0);
2241 #else
2242     if (!grid) {
2243         _exit(0);
2244     }
2245     DO_CHECK(fputs(NOCATGETS("exit 0\n"), scrfp));
2246     if (fclose(scrfp) != 0) {
2247         fprintf(stderr,
2248             catgets(catd, 1, 346, "Could not close file: %s:
2249             script_file, errmsg(errno));

```

```

2250         _exit(1);
2251     }
2252 }
2253
2254 #define DEFAULT_QRSH_TRIES_NUMBER      1
2255 #define DEFAULT_QRSH_TIMEOUT          0
2256
2257     static char    *sge_env_var = NULL;
2258     static int     qrsh_tries_number = DEFAULT_QRSH_TRIES_NUMBER;
2259     static int     qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2260
2261 #define SGE_DEBUG
2262 #ifdef SGE_DEBUG
2263     static Boolean do_not_remove = false;
2264     if (sge_env_var == NULL) {
2265         sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TRIES"))
2266         if (sge_env_var != NULL) {
2267             qrsh_tries_number = atoi(sge_env_var);
2268             if (qrsh_tries_number < 1 || qrsh_tries_number >
2269                 qrsh_tries_number = DEFAULT_QRSH_TRIES_N
2270             }
2271
2272         sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TIMEOUT")
2273         if (sge_env_var != NULL) {
2274             qrsh_timeout = atoi(sge_env_var);
2275             if (qrsh_timeout <= 0) {
2276                 qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2277             }
2278         } else {
2279             sge_env_var = "";
2280         }
2281 #ifdef SGE_DEBUG
2282     sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_DEBUG"))
2283     if (sge_env_var == NULL) {
2284         sge_env_var = "";
2285     }
2286     if (strstr(sge_env_var, NOCATGETS("noqrsh")) != NULL)
2287         qrsh_tries_number = 0;
2288     if (strstr(sge_env_var, NOCATGETS("donotremove")) != NUL
2289         do_not_remove = true;
2290 #endif /* SGE_DEBUG */
2291     }
2292     for (int i = qrsh_tries_number; , i--)
2293     if ((childPid = fork()) < 0) {
2294         fatal(catgets(catd, 1, 348, "Could not fork child proces
2295         errmsg(errno));
2296         _exit(1);
2297     } else if (childPid == 0) {
2298         enable_interrupt((void (*) (int))SIG_DFL);
2299         if (i > 0) {
2300             static char qrsh_cmd[50+MAXPATHLEN] = NOCATGETS(
2301             static char *fname_ptr = NULL;
2302             static char *argv[] = { NOCATGETS("sh"),
2303                                     NOCATGETS("-fce"),
2304                                     qrsh_cmd,
2305                                     NULL};
2306             if (fname_ptr == NULL) {
2307                 fname_ptr = qrsh_cmd + strlen(qrsh_cmd);
2308             }
2309             strcpy(fname_ptr, script_file);
2310             (void) execve(NOCATGETS("/bin/sh"), argv, enviro
2311         } else {
2312             static char *argv[] = { NOCATGETS("sh"),
2313                                     script_file,
2314                                     NULL};
2315             (void) execve(NOCATGETS("/bin/sh"), argv, enviro

```

```

2316         }
2317         fprintf(stderr,
2318             catgets(catd, 1, 349, "Could not load 'qrsh': %s
2319             errmsg(errno));
2320         _exit(1);
2321     } else {
2322         int             status;
2323         pid_t          pid;
2324         while ((pid = wait(&status)) != childPid) {
2325             if (pid == -1) {
2326                 fprintf(stderr,
2327                     catgets(catd, 1, 350, "wait() fa
2328                     errmsg(errno));
2329                 _exit(1);
2330             }
2331             if (status != 0 && i > 0) {
2332                 if (i > 1) {
2333                     sleep(qrsh_timeout);
2334                 }
2335                 continue;
2336             }
2337 #ifdef SGE_DEBUG
2338         if (do_not_remove) {
2339             if (status) {
2340                 fprintf(stderr,
2341                     NOCATGETS("SGE script failed: %s
2342                     script_file);
2343             }
2344             _exit(status ? 1 : 0);
2345         }
2346 #endif /* SGE_DEBUG */
2347         (void) unlink(script_file);
2348         _exit(status ? 1 : 0);
2349     }
2350 }
2351
2352 #endif /* SGE_SUPPORT */
2353 break;
2354 default:
2355     break;
2356 }
2357 return childPid;
2358 }
2359
2360 unchanged_portion_omitted

```

```
*****
15375 Wed May 20 11:37:23 2015
new/usr/src/cmd/include/mk/defs.h
make: unifdef for SGE (undefined)
*****
_____ unchanged_portion_omitted_



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

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

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

```

301 #ifdef SGE_SUPPORT
302 extern char           script_file[];
303 #endif
298 extern Boolean        stdout_stderr_same;
299 extern Dependency     suffixes;
300 extern Name           suffixes_name;
301 extern Name           sunpro_dependencies;
302 extern Boolean        target_variants;
303 extern const char    *tmpdir;
304 extern const char    *temp_file_directory;
305 extern Name           temp_file_name;
306 extern short          temp_file_number;
307 extern wchar_t        *top_level_target;
308 extern Boolean        touch;
309 extern Boolean        trace_reader;
310 extern Boolean        build_unconditional;
311 extern pathpt         vroot_path;
312 extern Name           wait_name;
313 extern wchar_t        wcs_buf[2];
314 extern wchar_t        *wcs_ptr;
315 extern wchar_t        *wcs_ptr2;
316 extern nl_catd        catd;
317 extern long int       hostid;

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

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

330 /*
331  * Declarations of functions declared and used by make
332 */
333 extern void            add_pending(Name target, int recursion_level, Boolean do_
334 extern void            add_running(Name target, Name true_target, Property comm_
335 extern void            add_serial(Name target, int recursion_level, Boolean do_
336 extern void            add_subtree(Name target, int recursion_level, Boolean do_
337 extern void            append_or_replace_macro_in_dyn_array(ASCII_Dyn_Array *Ar_
338 #ifdef DISTRIBUTED
339 extern Doname         await_dist(Boolean waitflg);
340 #endif
341 #ifdef TEAMWARE_MAKE_CMN
342 extern void            await_parallel(Boolean waitflg);
343 #endif
344 extern void            build_suffix_list(Name target_suffix);
345 extern Boolean         check_auto_dependencies(Name target, int auto_count, Nam_
346 extern void            check_state(Name temp_file_name);
347 extern void            cond_macros_into_string(Name np, String_rec *buffer);
348 extern void            construct_target_string();
349 extern void            create_xdrs_ptr(void);
350 extern void            depvar_add_to_list (Name name, Boolean cmdline);
351 #ifdef DISTRIBUTED
352 extern void            distribute_rxm(Avo_DoJobMsg *dmake_job_msg);
353 extern int             getRxmMessage(void);
354 extern Avo_JobResultMsg* getJobResultMsg(void);
355 extern Avo_AcknowledgeMsg* getAcknowledgeMsg(void);
356 #endif
357 extern Doname          done(name(register Name target, register Boolean do_get, re_
358 extern Doname          done_name_check(register Name target, register Boolean do_g_
359 extern Doname          done_parallel(Name target, Boolean do_get, Boolean imp_
360 extern Doname          dosys(register Name command, register Boolean ignore_err_

```

```

361 extern void            dump_make_state(void);
362 extern void            dump_target_list(void);
363 extern void            enter_conditional(register Name target, Name name, Name_
364 extern void            enter_dependencies(register Name target, Chain target_gr_
365 extern void            enter_dependency(Property line, register Name depe, Bool_
366 extern void            enter_equal(Name name, Name value, register Boolean appe_
367 extern Percent         enter_percent(register Name target, Chain target_group,_
368 extern Dyntarget      enter_dyntarget(register Name target);
369 extern Name_vector     enter_name(String string, Boolean tail_present, register_
370 extern Boolean         exec_vp(register char *name, register char **argv, char_
371 extern Doname          execute_parallel(Property line, Boolean waitflg, Boolean_
372 extern Doname          execute_serial(Property line));
373 extern timestruc_t&   exists(register Name target);
374 extern void            fatal(char *, ...);
375 extern void            fatal_reader(char *, ...);
376 extern Doname          find_ar_suffix_rule(register Name target, Name true_targ_
377 extern Doname          find_double_suffix_rule(register Name target, Property *
378 extern Doname          find_percent_rule(register Name target, Property *comm_
379 extern int              find_run_directory (char *cmd, char *cwd, char *dir, cha_
380 extern Doname          find_suffix_rule(Name target, Name target_body, Name tar_
381 extern Chain           find_target_groups(register Name_vector target_list, reg_
382 extern void            finish_children(Boolean docheck);
383 extern void            finish_running(void);
384 extern void            free_chain(Name_vector ptr);
385 extern void            gather_recursive_deps(void);
386 extern char            *get_current_path(void);
387 extern int             get_job_msg_id(void);
388 extern FILE            *get_mtool_msgs_fp(void);

389 #ifdef DISTRIBUTED
390 extern Boolean         get_dmake_group_specified(void);
391 extern Boolean         get_dmake_max_jobs_specified(void);
392 extern Boolean         get_dmake_mode_specified(void);
393 extern Boolean         get_dmake_odir_specified(void);
394 extern Boolean         get_dmake_rcfile_specified(void);
395 extern Boolean         get_pmake_machinesfile_specified(void);
396 #endif
397 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolk */
398 extern XDR             *get_xdrs_ptr(void);
399 #endif
400 extern wchar_t          *getmem_wc(register int size);
401 /* On linux getwd(char *) is defined in 'unistd.h' */
402 #ifdef __cplusplus
403 extern "C" {
404 #endif
405 extern char             *getwd(char *);
406 #ifdef __cplusplus
407 }

unchanged_portion_omitted_

```

```
*****
 22848 Wed May 20 11:37:23 2015
new/usr/src/cmd/include/mksh/defs.h
make: unifdef for SGE (undefined)
*****
_____unchanged_portion_omitted_____
116 /*
117 * For make i18n. Codeset independent.
118 * Setup character semantics by identifying all the special characters
119 * of make, and assigning each an entry in the char_semantics[] vector.
120 */
121 enum {
122     ampersand_char_entry = 0,      /* 0 */
123     asterisk_char_entry,        /* 1 */
124     at_char_entry,              /* 2 */
125     backquote_char_entry,       /* 3 */
126     backslash_char_entry,       /* 4 */
127     bar_char_entry,              /* 5 */
128     bracketleft_char_entry,     /* 6 */
129     bracketright_char_entry,    /* 7 */
130     colon_char_entry,           /* 8 */
131     dollar_char_entry,          /* 9 */
132     doublequote_char_entry,     /* 10 */
133     equal_char_entry,            /* 11 */
134     exclam_char_entry,          /* 12 */
135     greater_char_entry,         /* 13 */
136     hat_char_entry,              /* 14 */
137     hyphen_char_entry,           /* 15 */
138     less_char_entry,             /* 16 */
139     newline_char_entry,          /* 17 */
140     numbersign_char_entry,       /* 18 */
141     parenleft_char_entry,        /* 19 */
142     parenright_char_entry,       /* 20 */
143     percent_char_entry,          /* 21 */
144     plus_char_entry,              /* 22 */
145     question_char_entry,         /* 23 */
146     quote_char_entry,             /* 24 */
147     semicolon_char_entry,        /* 25 */
148 #ifdef SGE_SUPPORT
149     space_char_entry,            /* 26 */
150     tab_char_entry,              /* 27 */
151     no_semantics_entry,          /* 28 */
152 #else
153     no_semantics_entry,          /* 26 */
154 #endif /* SGE_SUPPORT */
149 };

151 /*
152 * CHAR_SEMATICS_ENTRIES should be the number of entries above.
153 * The last entry in char_semantics[] should be blank.
154 */
161 #ifdef SGE_SUPPORT
162 #define CHAR_SEMATICS_ENTRIES 29
163 /*
164 #define CHAR_SEMATICS_STRING "&*@\`\\/[ ]:$!=>-\\n#()%+?;^<' \\t"
165 */
166 #else
155 #define CHAR_SEMATICS_ENTRIES 27
156 /*
157 #define CHAR_SEMATICS_STRING "&*@\`\\|[ ]:$!=>-\\n#()%+?;^<'\""
158 */
171 #endif /* SGE_SUPPORT */

160 /*
161 * Some utility macros

```

```

162 */
163 #define ALLOC(x)
164 #define ALLOC_WC(x)
165 #define FIND_LENGTH
166 #define GETNAME(a,b)
167 #define IS_EQUAL(a,b)
168 #define IS_EQUALN(a,b,n)
169 #define IS_WEQUAL(a,b)
170 #define IS_WEQUALN(a,b,n)
171 #define MBLEN(a)
172 #define MBSTOWCS(a,b)
173 #define MBTOWC(a,b)
174 #define SIZEOFWCHAR_T
175 #define VSIZEOF(v)
176 #define WCSTOMBS(a,b)
177 #define WCTOMB(a,b)
178 #define HASH(v, c)

180 extern void mbstowcs_with_check(wchar_t *pwcs, const char *s, size_t n);

182 /*
183 * Bits stored in funny vector to classify chars
184 */
185 enum {
186     dollar_sem = 0001,
187     meta_sem = 0002,
188     percent_sem = 0004,
189     wildcard_sem = 0010,
190     command_prefix_sem = 0020,
191     special_macro_sem = 0040,
192     colon_sem = 0100,
193     parenleft_sem = 0200
194 };
_____unchanged_portion_omitted_____

```

new/usr/src/cmd/make/lib/mksh/globals.cc

```
*****
3035 Wed May 20 11:37:24 2015
new/usr/src/cmd/make/lib/mksh/globals.cc
make: unifdef for SGE (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 */

27 /*
28 *      globals.cc
29 *
30 *      This declares all global variables
31 */

33 /*
34 *      Included files
35 */
36 #include <mksh/globals.h>

38 /*
39 *      Defined macros
40 */

42 /*
43 *      typedefs & structs
44 */

46 /*
47 *      Global variables
48 */
49 char          char_semantics[CHAR_SEMANTICS_ENTRIES];
50 wchar_t       char_semantics_char[] = {
51     ampersand_char,
52     asterisk_char,
53     at_char,
54     backquote_char,
55     backslash_char,
56     bar_char,
57     bracketleft_char,
58     bracketright_char,
59     colon_char,
60     dollar_char,
61     doublequote_char,
```

1

new/usr/src/cmd/make/lib/mksh/globals.cc

```
62     equal_char,
63     exclam_char,
64     greater_char,
65     hat_char,
66     hyphen_char,
67     less_char,
68     newline_char,
69     numbersign_char,
70     parenleft_char,
71     parenright_char,
72     percent_char,
73     plus_char,
74     question_char,
75     quote_char,
76     semicolon_char,
77 #ifdef SGE_SUPPORT
78     space_char,
79     tab_char,
80#endif
81     nul_char
82 };

_____unchanged_portion_omitted_____
```

2