```
**********************************************************
   98813 Wed May 20 11:35:09 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for USE_DMS_CCR (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/intl.h>
  37 #       include <avo/libcli.h>          /* libcli_init() */
  38 #       include <avo/cli_license.h>     /* avo_cli_get_license() */
  39 #       include <avo/find_dir.h>        /* avo_find_run_dir() */
  40 #       include <avo/version_string.h>
  41 #       include <avo/util.h>            /* avo_init() */
  42 #ifdef USE_DMS_CCR
  43 #       include <avo/usage_tracking.h>
  44 #else
  42 #       include <avo/cleanup.h>
  43 #endif
  47 #endif

  45 #if defined(TEAMWARE_MAKE_CMN)
  46 /* This is for dmake only (not for Solaris make).
  47  * Include code to check updates (dmake patches)
  48  */
  49 #ifdef _CHECK_UPDATE_H
  50 #include <libAU.h>
  51 #endif
  52 #endif

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

  56 #ifdef DISTRIBUTED
  57 #       include <dm/Avo_AcknowledgeMsg.h>
```



```
  58 #       include <rw/xdrstrea.h>
  59 #       include <dmrc/dmrc.h> /* dmakerc file processing */
  60 #endif

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

  68 #if defined(TEAMWARE_MAKE_CMN)
  73 #ifdef USE_DMS_CCR
  74 #       include <pubdmsi18n/pubdmsi18n.h>       /* libpubdmsi18n_init() */
  75 #endif
  69 #endif

  71 #include <pwd.h>                /* getpwnam() */
  72 #include <setjmp.h>
  73 #include <signal.h>
  74 #include <stdlib.h>
  75 #include <sys/errno.h>          /* ENOENT */
  76 #include <sys/stat.h>           /* fstat() */
  77 #include <fcntl.h>              /* open() */

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

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

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

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

  93 #ifdef MAXJOBS_ADJUST_RFE4694000
  94 extern void job_adjust_fini();
  95 #endif /* MAXJOBS_ADJUST_RFE4694000 */
  96 #endif /* TEAMWARE_MAKE_CMN */


  99 /*
 100  * Defined macros
 101  */
 102 #define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT")
 103 #define LD_SUPPORT_MAKE_LIB     NOCATGETS("libmakestate.so.1")

 105 /*
 106  * typedefs & structs
 107  */

 109 /*
 110  * Static variables
 111  */
 112 static  char            *argv_zero_string;
 113 static  Boolean         build_failed_ever_seen;
 114 static  Boolean         continue_after_error_ever_seen; /* '-k' */
 115 static  Boolean         dmake_group_specified;          /* '-g' */
 116 static  Boolean         dmake_max_jobs_specified;       /* '-j' */
 117 static  Boolean         dmake_mode_specified;           /* '-m' */
 118 static  Boolean         dmake_add_mode_specified;       /* '-x' */
 119 static  Boolean         dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE
 120 static  Boolean         dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M
```

```
121 static  Boolean         dmake_odir_specified;           /* '-o' */
122 static  Boolean         dmake_rcfile_specified;         /* '-c' */
123 static  Boolean         env_wins;                       /* '-e' */
124 static  Boolean         ignore_default_mk;              /* '-r' */
125 static  Boolean         list_all_targets;               /* '-T' */
126 static  int             mf_argc;
127 static  char            **mf_argv;
128 static  Dependency_rec  not_auto_depen_struct;
129 static  Dependency      not_auto_depen = &not_auto_depen_struct;
130 static  Boolean         pmake_cap_r_specified;          /* '-R' */
131 static  Boolean         pmake_machinesfile_specified;   /* '-M' */
132 static  Boolean         stop_after_error_ever_seen;     /* '-S' */
133 static  Boolean         trace_status;                   /* '-p' */

135 #ifdef DMAKE_STATISTICS
136 static  Boolean         getname_stat = false;
137 #endif

139 #if defined(TEAMWARE_MAKE_CMN)
140         static  time_t          start_time;
141         static  int             g_argc;
142         static  char            **g_argv;
150 #ifdef USE_DMS_CCR
151         static  Avo_usage_tracking *usageTracking = NULL;
152 #else
143         static  Avo_cleanup     *cleanup = NULL;
144 #endif
155 #endif

146 /*
147  * File table of contents
148  */
149         extern "C" void         cleanup_after_exit(void);

151 #ifdef TEAMWARE_MAKE_CMN
152 extern "C" {
153         extern  void            dmake_exit_callback(void);
154         extern  void            dmake_message_callback(char *);
155 }
156 #endif

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

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

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

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

179 #ifdef DISTRIBUTED
180         extern  int             dmake_ofd;
181         extern  FILE*           dmake_ofp;
182         extern  int             rxmPid;
```



```
183         extern  XDR             xdrs_out;
184 #endif
185 #ifdef TEAMWARE_MAKE_CMN
186         extern  char            verstring[];
187 #endif

189 jmp_buf jmpbuffer;
190 extern nl_catd catd;

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

239 #ifndef PARALLEL
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();

248         (void) setlocale(LC_ALL, "");
```

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



        /*
         * avo_init() sets the umask to 0.  Save it here and restore
         * it after the avo_init() call.
         */
#if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
        um = umask(0);
        avo_init(argv[0]);
        umask(um);

#ifdef USE_DMS_CCR
        usageTracking = new Avo_usage_tracking(NOCATGETS("dmake"), argc, argv);
#else
        cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
#endif
#endif

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

#ifdef _CHECK_UPDATE_H
        /* This is for dmake only (not for Solaris make).
         * Check (in background) if there is an update (dmake patch)
         * and inform user
         */
        {
                Avo_err         *err;
                char            *dir;
                err = avo_find_run_dir(&dir);
                if (AVO_OK == err) {
                        AU_check_update_service(NOCATGETS("Dmake"), dir);
                }
        }
#endif /* _CHECK_UPDATE_H */
#endif

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


#if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
/*
 * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
 * from avo_util library.
 */
        libmksdmsi18n_init();
#ifdef USE_DMS_CCR
        libpubdmsi18n_init();
#endif
#endif



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

#ifdef TEAMWARE_MAKE_CMN
        g_argc = argc;
```



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

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

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

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

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


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

        /* Sun OS make standart */
        svr4 = false;
        posix = false;
        if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
                svr4 = false;
                posix = true;
        } else {
                prognameptr = strrchr(argv[0], '/');
                if(prognameptr) {
                        prognameptr++;
                } else {
```

```
374                         prognameptr = argv[0];
375                 }
376                 if(!strcmp(prognameptr, 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                 }
394                 //svr4 = false;
395                 //posix = false;
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                                 tmpdir = strdup(mbs_buffer);
415                         }
416                 }
417         }

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);

439         setup_char_semantics();
```



```
441         setup_for_projectdir();

443         /*
444          * If running with .KEEP_STATE, curdir will be set with
445          * the connected directory.
446          */
447         (void) atexit(cleanup_after_exit);

449         load_cached_names();

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         }

462         setup_interrupt(handle_interrupt);

464         read_files_and_state(argc, argv);

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 #endif
518                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
519                         dmake_mode_type = serial_mode;
520                         no_parallel = true;
521                 } else {
522                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
523                 }
524         }

526         if ((!list_all_targets) &&
527             (report_dependencies_level == 0)) {
528                 /*
529                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
530                  * They could be defined in the env, in the makefile, or on the
531                  * command line.
532                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
533                  * then print a message, and default to parallel mode.
534                  */
535 #ifdef DISTRIBUTED
536                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
537                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
538                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
539                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
540                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
541                      ((dmake_value = prop->body.macro.value) == NULL)) &&
542                     (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
543                      ((dmake_value2 = prop2->body.macro.value) == NULL))) {
544                         Boolean empty_dmakerc = true;
545                         char *homedir = getenv(NOCATGETS("HOME"));
546                         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
547                                 sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
548                                 if ((((statval = stat(def_dmakerc_path, &statbuf
549                                         ((statval == 0) && (statbuf.st_size == 0
550                                 } else {
551                                         Avo_dmakerc     *rcfile = new Avo_dmaker
552                                         Avo_err         *err = rcfile->read(def_
553                                         if (err) {
554                                                 fatal(err->str);
555                                         }
556                                         empty_dmakerc = rcfile->was_empty();
557                                         delete rcfile;
558                                 }
559                         }
560                         if (empty_dmakerc) {
561                                 if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
562                                         putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
563                                         (void) fprintf(stdout, catgets(catd, 1,
564                                         (void) fprintf(stdout, catgets(catd, 1,
565                                 }
566                                 dmake_mode_type = parallel_mode;
567                                 no_parallel = false;
568                         }
569                 }
570 #else
571                 if(dmake_mode_type == distributed_mode) {
```



```
572                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
573                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
574                         dmake_mode_type = parallel_mode;
575                         no_parallel = false;
576                 }
577 #endif  /* DISTRIBUTED */
578         }
579     }
580 #endif

582 #ifdef TEAMWARE_MAKE_CMN
583         parallel_flag = true;
584         /* XXX - This is a major hack for DMake/Licensing. */
585         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
586                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
587                         /*
588                          * If the user can not get a TeamWare license,
589                          * default to serial mode.
590                          */
591                         dmake_mode_type = serial_mode;
592                         no_parallel = true;
593                 } else {
594                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
595                 }
596                 start_time = time(NULL);
597                 /*
598                  * XXX - Hack to disable SIGALRM's from licensing library's
599                  *       setitimer().
600                  */
601                 value.it_interval.tv_sec = 0;
602                 value.it_interval.tv_usec = 0;
603                 value.it_value.tv_sec = 0;
604                 value.it_value.tv_usec = 0;
605                 (void) setitimer(ITIMER_REAL, &value, NULL);
606         }

608 //
609 // If dmake is running with -t option, set dmake_mode_type to serial.
610 // This is done because doname() calls touch_command() that runs serially.
611 // If we do not do that, maketool will have problems.
612 //
613         if(touch) {
614                 dmake_mode_type = serial_mode;
615                 no_parallel = true;
616         }
617 #else
618         parallel_flag = false;
619 #endif

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

```
638                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
639                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
640                         {
641                                 out_err_same = false;
642                         }
643                 }
644         }
645 #endif

647 #ifdef DISTRIBUTED
648         /*
649          * At this point, DMake should startup an rxm with any and all
650          * DMake command line options. Rxm will, among other things,
651          * read the rc file.
652          */
653         if ((!list_all_targets) &&
654             (report_dependencies_level == 0) &&
655             (dmake_mode_type == distributed_mode)) {
656                 startup_rxm();
657         }
658 #endif
659
660 /*
661  *      Enable interrupt handler for alarms
662  */
663         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

665 /*
666  *      Check if make should report
667  */
668         if (getenv(sunpro_dependencies->string_mb) != NULL) {
669                 FILE    *report_file;

671                 report_dependency("");
672                 report_file = get_report_file();
673                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
674                         (void) fprintf(report_file, "\n");
675                 }
676         }

678 /*
679  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly.
680  */
681         if (keep_state) {
682                 maybe_append_prop(sunpro_dependencies, macro_prop)->
683                   body.macro.exported = true;
684         } else {
685                 maybe_append_prop(sunpro_dependencies, macro_prop)->
686                   body.macro.exported = false;
687         }

689         working_on_targets = true;
690         if (trace_status) {
691                 dump_make_state();
692                 fclose(stdout);
693                 fclose(stderr);
694                 exit_status = 0;
695                 exit(0);
696         }
697         if (list_all_targets) {
698                 dump_target_list();
699                 fclose(stdout);
700                 fclose(stderr);
701                 exit_status = 0;
702                 exit(0);
703         }
```



```
704         trace_reader = false;

706         /*
707          * Set temp_file_directory to the directory the .make.state
708          * file is written to.
709          */
710         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
711                 temp_file_directory = strdup(get_current_path());
712         } else {
713                 *slash_ptr = (int) nul_char;
714                 (void) strcpy(make_state_dir, make_state->string_mb);
715                 *slash_ptr = (int) slash_char;
716                    /* when there is only one slash and it's the first
717                    ** character, make_state_dir should point to '/'.
718                    */
719                 if(make_state_dir[0] == '\0') {
720                    make_state_dir[0] = '/';
721                    make_state_dir[1] = '\0';
722                 }
723                 if (make_state_dir[0] == (int) slash_char) {
724                         temp_file_directory = strdup(make_state_dir);
725                 } else {
726                         char    tmp_current_path2[MAXPATHLEN];
727
728                         (void) sprintf(tmp_current_path2,
729                                         "%s/%s",
730                                         get_current_path(),
731                                         make_state_dir);
732                         temp_file_directory = strdup(tmp_current_path2);
733                 }
734         }

736 #ifdef DISTRIBUTED
737         building_serial = false;
738 #endif

740         report_dir_enter_leave(true);

742         make_targets(argc, argv, parallel_flag);

744         report_dir_enter_leave(false);

746         if (build_failed_ever_seen) {
747                 if (posix) {
748                         exit_status = 1;
749                 }
750                 exit(1);
751         }
752         exit_status = 0;
753         exit(0);
754         /* NOTREACHED */
755 }

757 /*
758  *      cleanup_after_exit()
759  *
760  *      Called from exit(), performs cleanup actions.
761  *
762  *      Parameters:
763  *              status          The argument exit() was called with
764  *              arg             Address of an argument vector to
765  *                              cleanup_after_exit()
766  *
767  *      Global variables used:
768  *              command_changed Set if we think .make.state should be rewritten
769  *              current_line    Is set we set commands_changed
```

```
 770  *              do_not_exec_rule
 771  *                              True if -n flag on
 772  *              done            The Name ".DONE", rule we run
 773  *              keep_state      Set if .KEEP_STATE seen
 774  *              parallel        True if building in parallel
 775  *              quest           If -q is on we do not run .DONE
 776  *              report_dependencies
 777  *                              True if -P flag on
 778  *              running_list    List of parallel running processes
 779  *              temp_file_name  The temp file is removed, if any
 780  *              catd    the message catalog file
 799  *              usage_tracking  Should have been constructed in main()
 800  *                              should destroyed just before exiting
 781  */
 782 extern "C" void
 783 cleanup_after_exit(void)
 784 {
 785         Running         rp;

 787 extern long     getname_bytes_count;
 788 extern long     getname_names_count;
 789 extern long     getname_struct_count;
 790 extern long     freename_bytes_count;
 791 extern long     freename_names_count;
 792 extern long     freename_struct_count;
 793 extern long     other_alloc;

 795 extern long     env_alloc_num;
 796 extern long     env_alloc_bytes;


 799 #ifdef DMAKE_STATISTICS
 800 if(getname_stat) {
 801         printf(NOCATGETS(">>> Getname statistics:\n"));
 802         printf(NOCATGETS("  Allocated:\n"));
 803         printf(NOCATGETS("        Names: %ld\n"), getname_names_count);
 804         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), getname_bytes_c
 805         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), getname_struct_
 806         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), getname_struct_

 808         printf(NOCATGETS("\n  Unallocated: %ld\n"), freename_names_count);
 809         printf(NOCATGETS("        Names: %ld\n"), freename_names_count);
 810         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), freename_bytes_
 811         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), freename_struct
 812         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), freename_struct

 814         printf(NOCATGETS("\n  Total used: %ld Kb (%ld bytes)\n"), (getname_struc

 816         printf(NOCATGETS("\n>>> Other:\n"));
 817         printf(
 818                 NOCATGETS("       Env (%ld): %ld Kb (%ld bytes)\n"),
 819                 env_alloc_num,
 820                 env_alloc_bytes/1000,
 821                 env_alloc_bytes
 822         );

 824 }
 825 #endif

 827 /*
 828 #ifdef DISTRIBUTED
 829     if (get_parent() == TRUE) {
 830 #endif
 831  */

 833         parallel = false;
```



```
 834         /* If we used the SVR4_MAKE, don't build .DONE or .FAILED */
 835         if (!getenv(USE_SVR4_MAKE)){
 836             /* Build the target .DONE or .FAILED if we caught an error */
 837             if (!quest && !list_all_targets) {
 838                 Name            failed_name;

 840                 MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
 841                 failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
 842                 if ((exit_status != 0) && (failed_name->prop != NULL)) {
 843 #ifdef TEAMWARE_MAKE_CMN
 844                         /*
 845                          * [tolik] switch DMake to serial mode
 846                          */
 847                         dmake_mode_type = serial_mode;
 848                         no_parallel = true;
 849 #endif
 850                         (void) doname(failed_name, false, true);
 851                 } else {
 852                     if (!trace_status) {
 853 #ifdef TEAMWARE_MAKE_CMN
 854                         /*
 855                          * Switch DMake to serial mode
 856                          */
 857                         dmake_mode_type = serial_mode;
 858                         no_parallel = true;
 859 #endif
 860                         (void) doname(done, false, true);
 861                     }
 862                 }
 863             }
 864         }
 865         /*
 866          * Remove the temp file utilities report dependencies thru if it
 867          * is still around
 868          */
 869         if (temp_file_name != NULL) {
 870                 (void) unlink(temp_file_name->string_mb);
 871         }
 872         /*
 873          * Do not save the current command in .make.state if make
 874          * was interrupted.
 875          */
 876         if (current_line != NULL) {
 877                 command_changed = true;
 878                 current_line->body.line.command_used = NULL;
 879         }
 880         /*
 881          * For each parallel build process running, remove the temp files
 882          * and zap the command line so it won't be put in .make.state
 883          */
 884         for (rp = running_list; rp != NULL; rp = rp->next) {
 885                 if (rp->temp_file != NULL) {
 886                         (void) unlink(rp->temp_file->string_mb);
 887                 }
 888                 if (rp->stdout_file != NULL) {
 889                         (void) unlink(rp->stdout_file);
 890                         retmem_mb(rp->stdout_file);
 891                         rp->stdout_file = NULL;
 892                 }
 893                 if (rp->stderr_file != NULL) {
 894                         (void) unlink(rp->stderr_file);
 895                         retmem_mb(rp->stderr_file);
 896                         rp->stderr_file = NULL;
 897                 }
 898                 command_changed = true;
 899 /*
```

```
900                 line = get_prop(rp->target->prop, line_prop);
901                 if (line != NULL) {
902                         line->body.line.command_used = NULL;
903                 }
904  */
905         }
906         /* Remove the statefile lock file if the file has been locked */
907         if ((make_state_lockfile != NULL) && (make_state_locked)) {
908                 (void) unlink(make_state_lockfile);
909                 make_state_lockfile = NULL;
910                 make_state_locked = false;
911         }
912         /* Write .make.state */
913         write_state_file(1, (Boolean) 1);

915 #ifdef TEAMWARE_MAKE_CMN
916         // Deleting the usage tracking object sends the usage mail
937 #ifdef USE_DMS_CCR
938         //usageTracking->setExitStatus(exit_status, NULL);
939         //delete usageTracking;
940 #else
917         cleanup->set_exit_status(exit_status);
918         delete cleanup;
919 #endif
944 #endif

921 /*
922 #ifdef DISTRIBUTED
923     }
924 #endif
925  */

927 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
928         job_adjust_fini();
929 #endif

931 #ifdef TEAMWARE_MAKE_CMN
932         catclose(catd);
933 #endif
934 #ifdef DISTRIBUTED
935         if (rxmPid > 0) {
936                 // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
937                 Avo_AcknowledgeMsg acknowledgeMsg;
938                 RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

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

942                 if (xdrResult) {
943                         fflush(dmake_ofp);
944                 } else {
945 /*
946                         fatal(catgets(catd, 1, 266, "couldn't tell rxm to exit")
947  */
948                         kill(rxmPid, SIGTERM);
949                 }

951                 waitpid(rxmPid, NULL, 0);
952                 rxmPid = 0;
953         }
954 #endif
955 }
```

_____***unchanged_portion_omitted_***