

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

1

```
*****
86686 Wed May 20 12:22:44 2015
new/usr/src/cmd/make/bin/main.cc
make: undef for two bugfixes conditioned for unknown reasons (defined)
*****
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 #include <bsd/bsd.h>          /* bsd_signal() */

38 #include <locale.h>          /* setlocale() */
39 #include <libgen.h>
40 #include <mk/defs.h>
41 #include <mksh/macro.h>      /* getvar() */
42 #include <mksh/misc.h>      /* getmem(), setup_char_semantics() */

44 #include <pwd.h>              /* getpwnam() */
45 #include <setjmp.h>
46 #include <signal.h>
47 #include <stdlib.h>
48 #include <sys/errno.h>       /* ENOENT */
49 #include <sys/stat.h>        /* fstat() */
50 #include <fcntl.h>           /* open() */

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

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

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

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

2

```
62 // From parallel.cc
63 #define MAXJOBS_ADJUST_RFE4694000

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

68 /*
69  * Defined macros
70  */
71 #define LD_SUPPORT_ENV_VAR      "SGS_SUPPORT_32"
72 #define LD_SUPPORT_ENV_VAR_32  "SGS_SUPPORT_32"
73 #define LD_SUPPORT_ENV_VAR_64  "SGS_SUPPORT_64"
74 #define LD_SUPPORT_MAKE_LIB    "libmakestate.so.1"
75 #ifdef __i386
76 #define LD_SUPPORT_MAKE_ARCH   "i386"
77 #elif __sparc
78 #define LD_SUPPORT_MAKE_ARCH   "sparc"
79 #else
80 #error "Unsupported architecture"
81 #endif

83 /*
84  * typedefs & structs
85  */

87 /*
88  * Static variables
89  */
90 static char      *argv_zero_string;
91 static Boolean   build_failed_ever_seen;
92 static Boolean   continue_after_error_ever_seen; /* '-k' */
93 static Boolean   dmake_group_specified;         /* '-g' */
94 static Boolean   dmake_max_jobs_specified;      /* '-j' */
95 static Boolean   dmake_mode_specified;          /* '-m' */
96 static Boolean   dmake_add_mode_specified;      /* '-x' */
97 static Boolean   dmake_output_mode_specified;   /* '-x DMAKE_OUTPUT_MODE */
98 static Boolean   dmake_compat_mode_specified;   /* '-x SUN_MAKE_COMPAT_M */
99 static Boolean   dmake_odir_specified;          /* '-o' */
100 static Boolean   dmake_rcfile_specified;        /* '-c' */
101 static Boolean   env_wins;                       /* '-e' */
102 static Boolean   ignore_default_mk;             /* '-r' */
103 static Boolean   list_all_targets;              /* '-T' */
104 static int       mf_argc;
105 static char      **mf_argv;
106 static Dependency_rec not_auto_depen_struct;
107 static Dependency not_auto_depen = &not_auto_depen_struct;
108 static Boolean   pmake_cap_r_specified;         /* '-R' */
109 static Boolean   pmake_machinesfile_specified;  /* '-M' */
110 static Boolean   stop_after_error_ever_seen;    /* '-S' */
111 static Boolean   trace_status;                  /* '-p' */

113 #ifdef DMAKE_STATISTICS
114 static Boolean   getname_stat = false;
115 #endif

117 static time_t    start_time;
118 static int       g_argc;
119 static char      **g_argv;

121 /*
122  * File table of contents
123  */
124 extern "C" void  cleanup_after_exit(void);
```

```

126 extern "C" {
127     extern void          dmake_exit_callback(void);
128     extern void          dmake_message_callback(char *);
129 }
   unchanged portion omitted
636 #endif

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

645             MBSTOWCS(wcs_buffer, ".FAILED");
646             failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
647             if ((exit_status != 0) && (failed_name->prop != NULL)) {
648                 /*
649                  * [tolik] switch DMake to serial mode
650                  */
651                 dmake_mode_type = serial_mode;
652                 no_parallel = true;
653                 (void) doname(failed_name, false, true);
654             } else {
655                 if (!trace_status) {
656                     /*
657                      * Switch DMake to serial mode
658                      */
659                     dmake_mode_type = serial_mode;
660                     no_parallel = true;
661                     (void) doname(done, false, true);
662                 }
663             }
664         }
665     }
666     /*
667     * Remove the temp file utilities report dependencies thru if it
668     * is still around
669     */
670     if (temp_file_name != NULL) {
671         (void) unlink(temp_file_name->string_mb);
672     }
673     /*
674     * Do not save the current command in .make.state if make
675     * was interrupted.
676     */
677     if (current_line != NULL) {
678         command_changed = true;
679         current_line->body.line.command_used = NULL;
680     }
681     /*
682     * For each parallel build process running, remove the temp files
683     * and zap the command line so it won't be put in .make.state
684     */
685     for (rp = running_list; rp != NULL; rp = rp->next) {
686         if (rp->temp_file != NULL) {
687             (void) unlink(rp->temp_file->string_mb);
688         }
689         if (rp->stdout_file != NULL) {
690             (void) unlink(rp->stdout_file);
691             retmem_mb(rp->stdout_file);
692             rp->stdout_file = NULL;
693         }
694         if (rp->stderr_file != NULL) {
695             (void) unlink(rp->stderr_file);
696             retmem_mb(rp->stderr_file);

```

```

697         rp->stderr_file = NULL;
698     }
699     command_changed = true;
700     /*
701     line = get_prop(rp->target->prop, line_prop);
702     if (line != NULL) {
703         line->body.line.command_used = NULL;
704     }
705     */
706     /*
707     * Remove the statefile lock file if the file has been locked */
708     if ((make_state_lockfile != NULL) && (make_state_locked)) {
709         (void) unlink(make_state_lockfile);
710         make_state_lockfile = NULL;
711         make_state_locked = false;
712     }
713     /* Write .make.state */
714     write_state_file(1, (Boolean) 1);

718 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
719     job_adjust_fini();
720 #endif
721 }
   unchanged portion omitted

1131 /*
1132 * Convert the MAKEFLAGS string value into a vector of char *, similar
1133 * to argv.
1134 */
1135 static void
1136 setup_makeflags_argv()
1137 {
1138     char          *cp;
1139     char          *cp1;
1140     char          *cp2;
1141     char          *cp3;
1142     char          *cp_orig;
1143     Boolean       add_hyphen;
1144     int           i;
1145     char          tmp_char;

1147     mf_argc = 1;
1148     cp = getenv(makeflags->string_mb);
1149     cp_orig = cp;

1151     if (cp) {
1152         /*
1153          * If new MAKEFLAGS format, no need to add hyphen.
1154          * If old MAKEFLAGS format, add hyphen before flags.
1155          */

1157         if ((strchr(cp, (int) hyphen_char) != NULL) ||
1158             (strchr(cp, (int) equal_char) != NULL)) {

1160             /* New MAKEFLAGS format */

1162             add_hyphen = false;

1167 #ifdef ADDFIX5060758
1168             /* Check if MAKEFLAGS value begins with multiple
1169              * hyphen characters, and remove all duplicates.
1170              * Usually it happens when the next command is
1171              * used: $(MAKE) -$(MAKEFLAGS)
1172              *
1173              * This was a workaround for BugID 5060758, but
1174              * appears to have survived as a fix in make.

```

```

1172     * This is a workaround for BugID 5060758.
1173     */
1174     while (*cp) {
1175         if (*cp != (int) hyphen_char) {
1176             break;
1177         }
1178         cp++;
1179         if (*cp == (int) hyphen_char) {
1180             /* There are two hyphens. Skip one */
1181             cp_orig = cp;
1182             cp++;
1183         }
1184         if (!( *cp)) {
1185             /* There are hyphens only. Skip all */
1186             cp_orig = cp;
1187             break;
1188         }
1189     }
1190 #endif
1191 } else {
1192
1193     /* Old MAKEFLAGS format */
1194
1195     add_hyphen = true;
1196 }
1197
1198 /* Find the number of arguments in MAKEFLAGS */
1199 while (cp && *cp) {
1200     /* Skip white spaces */
1201     while (cp && *cp && isspace(*cp)) {
1202         cp++;
1203     }
1204     if (cp && *cp) {
1205         /* Increment arg count */
1206         mf_argc++;
1207         /* Go to next white space */
1208         while (cp && *cp && !isspace(*cp)) {
1209             if (*cp == (int) backslash_char) {
1210                 cp++;
1211             }
1212             cp++;
1213         }
1214     }
1215 }
1216 /* Allocate memory for the new MAKEFLAGS argv */
1217 mf_argv = (char **) malloc((mf_argc + 1) * sizeof(char *));
1218 mf_argv[0] = (char *)"MAKEFLAGS";
1219 /*
1220 * Convert the MAKEFLAGS string value into a vector of char *,
1221 * similar to argv.
1222 */
1223 cp = cp_orig;
1224 for (i = 1; i < mf_argc; i++) {
1225     /* Skip white spaces */
1226     while (cp && *cp && isspace(*cp)) {
1227         cp++;
1228     }
1229     if (cp && *cp) {
1230         cp_orig = cp;
1231         /* Go to next white space */
1232         while (cp && *cp && !isspace(*cp)) {
1233             if (*cp == (int) backslash_char) {
1234                 cp++;
1235             }
1236             cp++;
1237         }
1238     }
1239 }

```

```

1235     }
1236     tmp_char = *cp;
1237     *cp = (int) nul_char;
1238     if (add_hyphen) {
1239         mf_argv[i] = getmem(2 + strlen(cp_orig));
1240         mf_argv[i][0] = '\\0';
1241         (void) strcat(mf_argv[i], "-");
1242         // (void) strcat(mf_argv[i], cp_orig);
1243         unquote_str(cp_orig, mf_argv[i]+1);
1244     } else {
1245         mf_argv[i] = getmem(2 + strlen(cp_orig));
1246         //mf_argv[i] = strdup(cp_orig);
1247         unquote_str(cp_orig, mf_argv[i]);
1248     }
1249     *cp = tmp_char;
1250 }
1251 }
1252 mf_argv[i] = NULL;
1253 }

```

unchanged_portion_omitted

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

1

```
*****
19307 Wed May 20 12:22:45 2015
new/usr/src/cmd/make/bin/misc.cc
make: unifdef for two bugfixes conditioned for unknown reasons (defined)
*****
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 /*
27  *      misc.cc
28  *
29  *      This file contains various unclassified routines. Some main groups:
30  *      getname
31  *      Memory allocation
32  *      String handling
33  *      Property handling
34  *      Error message handling
35  *      Make internal state dumping
36  *      main routine support
37  */

39 /*
40  * Included files
41  */
42 #include <errno.h>
43 #include <mk/defs.h>
44 #include <mksh/macro.h>          /* SETVAR() */
45 #include <mksh/misc.h>          /* enable_interrupt() */
46 #include <stdarg.h>             /* va_list, va_start(), va_end() */
47 #include <vroot/report.h>       /* SUNPRO_DEPENDENCIES */
48 #include <libintl.h>

51 #define MAXJOBS_ADJUST_RFE4694000

53 #ifdef MAXJOBS_ADJUST_RFE4694000
50 extern void job_adjust_fini();
55 #endif /* MAXJOBS_ADJUST_RFE4694000 */

52 /*
53  * Defined macros
54  */
```

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

2

```
56 /*
57  * typedefs & structs
58  */

60 /*
61  * Static variables
62  */

64 /*
65  * File table of contents
66  */
67 static void      print_rule(register Name target);
68 static void      print_target_n_deps(register Name target);

70 /*****
71  *
72  *      getname
73  */

75 /*****
76  *
77  *      Memory allocation
78  */

80 /*
81  *      free_chain()
82  *
83  *      frees a chain of Name_vector's
84  *
85  *      Parameters:
86  *          ptr          Pointer to the first element in the chain
87  *                      to be freed.
88  *
89  *      Global variables used:
90  */
91 void
92 free_chain(Name_vector ptr)
93 {
94     if (ptr != NULL) {
95         if (ptr->next != NULL) {
96             free_chain(ptr->next);
97         }
98         free((char *) ptr);
99     }
100 }

102 /*****
103  *
104  *      String manipulation
105  */

107 /*****
108  *
109  *      Nameblock property handling
110  */

112 /*****
113  *
114  *      Error message handling
115  */

117 /*
118  *      fatal(format, args...)
119  *
120  *      Print a message and die
121  */
```

```

122 *      Parameters:
123 *          format      printf type format string
124 *          args        Arguments to match the format
125 *
126 *      Global variables used:
127 *          fatal_in_progress Indicates if this is a recursive call
128 *          parallel_process_cnt Do we need to wait for anything?
129 *          report_pwd      Should we report the current path?
130 */
131 /*VARARGS*/
132 void
133 fatal(const char *message, ...)
134 {
135     va_list args;
136
137     va_start(args, message);
138     (void) fflush(stdout);
139     (void) fprintf(stderr, gettext("make: Fatal error: "));
140     (void) vfprintf(stderr, message, args);
141     (void) fprintf(stderr, "\n");
142     va_end(args);
143     if (report_pwd) {
144         (void) fprintf(stderr,
145             gettext("Current working directory %s\n"),
146             get_current_path());
147     }
148     (void) fflush(stderr);
149     if (fatal_in_progress) {
150         exit_status = 1;
151         exit(1);
152     }
153     fatal_in_progress = true;
154     /* Let all parallel children finish */
155     if ((dmake_mode_type == parallel_mode) &&
156         (parallel_process_cnt > 0)) {
157         (void) fprintf(stderr,
158             gettext("Waiting for %d %s to finish\n"),
159             parallel_process_cnt,
160             parallel_process_cnt == 1 ?
161             gettext("job") : gettext("jobs"));
162         (void) fflush(stderr);
163     }
164
165     while (parallel_process_cnt > 0) {
166         await_parallel(true);
167         finish_children(false);
168     }
169
170     #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
171     job_adjust_fini();
172     #endif
173
174     exit_status = 1;
175     exit(1);
176 }

```

unchanged portion omitted

```

*****
45066 Wed May 20 12:22:46 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef for two bugfixes conditioned for unknown reasons (defined)
*****
_____unchanged_portion_omitted_____

250 #define MAXJOBS_ADJUST_RFE4694000

252 #ifdef MAXJOBS_ADJUST_RFE4694000

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

257 /*
258 *      adjust_pmake_max_jobs (int pmake_max_jobs)
259 *
260 *      Parameters:
261 *          pmake_max_jobs - max jobs limit set by user
262 *
263 *      External functions used:
264 *          sysconf()
265 *          getloadavg()
266 */
267 static int
268 adjust_pmake_max_jobs (int pmake_max_jobs)
269 {
270     static int      ncpu = 0;
271     double          loadavg[3];
272     int             adjustment;
273     int             adjusted_max_jobs;

275     if (ncpu <= 0) {
276         if ((ncpu = sysconf(_SC_NPROCESSORS_ONLN)) <= 0) {
277             ncpu = 1;
278         }
279     }
280     if (getloadavg(loadavg, 3) != 3) return(pmake_max_jobs);
281     adjustment = ((int)loadavg[LOADAVG_LMIN]);
282     if (adjustment < 2) return(pmake_max_jobs);
283     if (ncpu > 1) {
284         adjustment = adjustment / ncpu;
285     }
286     adjusted_max_jobs = pmake_max_jobs - adjustment;
287     if (adjusted_max_jobs < 1) adjusted_max_jobs = 1;
288     return(adjusted_max_jobs);
289 }
_____unchanged_portion_omitted_____

538 #endif /* MAXJOBS_ADJUST_RFE4694000 */

536 /*
537 *      distribute_process(char **commands, Property line)
538 *
539 *      Parameters:
540 *          commands      argv vector of commands to execute
541 *
542 *      Return value:
543 *          The result of the execution
544 *
545 *      Static variables used:
546 *          process_running Set to the pid of the process set running

```

```

551 * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
552 *      job_adjust_mode Current job adjust mode
553 * #endif
554 */
555 static Doname
556 distribute_process(char **commands, Property line)
557 {
558     static unsigned file_number = 0;
559     wchar_t         string[MAXPATHLEN];
560     char            mbstring[MAXPATHLEN];
561     int             filed;
562     int             res;
563     int             tmp_index;
564     char            *tmp_index_str_ptr;

566 #if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
567     while (parallel_process_cnt >= pmake_max_jobs) {
568         await_parallel(false);
569         finish_children(true);
570     }
571 #else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
572 /* initialize adjust mode, if not initialized */
573 if (job_adjust_mode == ADJUST_UNKNOWN) {
574     job_adjust_init();
575 }

576 /* actions depend on adjust mode */
577 switch (job_adjust_mode) {
578 case ADJUST_M1:
579     while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
580         await_parallel(false);
581         finish_children(true);
582     }
583     break;
584 case ADJUST_M2:
585     if ((res = m2_acquire_job()) == 0) {
586         if (parallel_process_cnt > 0) {
587             await_parallel(false);
588             finish_children(true);
589         }
590         if ((res = m2_acquire_job()) == 0) {
591             return build_serial;
592         }
593     } else {
594         return build_serial;
595     }
596 }
597 if (res < 0) {
598     /* job adjustment error */
599     job_adjust_error();
600 }

601 /* no adjustment */
602 while (parallel_process_cnt >= pmake_max_jobs) {
603     await_parallel(false);
604     finish_children(true);
605 }
606 }

615 #endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */

```

```

604     setvar_envvvar();
605     /*
606     * Tell the user what DMake is doing.
607     */
608     if (!silent && output_mode != txt2_mode) {
609         /*
610         * Print local_host --> x job(s).
611         */
612         (void) fprintf(stdout,
613             gettext("%s --> %d %s\n"),
614             local_host,
615             parallel_process_cnt + 1,
616             (parallel_process_cnt == 0) ? gettext("job") : ge

618         /* Print command line(s). */
619         tmp_index = 0;
620         while (commands[tmp_index] != NULL) {
621             /* No @ char. */
622             /* XXX - need to add [2] when + prefix is added */
623             if ((commands[tmp_index][0] != (int) at_char) &&
624                 (commands[tmp_index][1] != (int) at_char)) {
625                 tmp_index_str_ptr = commands[tmp_index];
626                 if (*tmp_index_str_ptr == (int) hyphen_char) {
627                     tmp_index_str_ptr++;
628                 }
629                 (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
630             }
631             tmp_index++;
632         }
633         (void) fflush(stdout);
634     }

636     (void) sprintf(mbstring,
637         "%s/dmake.stdout.%d.%d.XXXXXX",
638         tmpdir,
639         getpid(),
640         file_number++);

642     mktemp(mbstring);

644     stdout_file = strdup(mbstring);
645     stderr_file = NULL;

647     if (!out_err_same) {
648         (void) sprintf(mbstring,
649             "%s/dmake.stderr.%d.%d.XXXXXX",
650             tmpdir,
651             getpid(),
652             file_number++);

654         mktemp(mbstring);

656         stderr_file = strdup(mbstring);
657     }

659     process_running = run_rule_commands(local_host, commands);

661     return build_running;
662 }

```

unchanged portion omitted

```

1089 /*
1090 *     await_parallel(waitflg)
1091 *
1092 *     Waits for parallel children to exit and finishes their processing.
1093 *     If waitflg is false, the function returns after update_delay.

```

```

1094 *
1095 *     Parameters:
1096 *         waitflg           dwight
1097 */
1098 void
1099 await_parallel(Boolean waitflg)
1100 {
1101     Boolean        nohang;
1102     pid_t          pid;
1103     int            status;
1104     Running        rp;
1105     int            waiterr;

1107     nohang = false;
1108     for ( ; ; ) {
1109         if (!nohang) {
1110             (void) alarm((int) update_delay);
1111         }
1112         pid = waitpid((pid_t)-1,
1113             &status,
1114             nohang ? WNOHANG : 0);
1115         waiterr = errno;
1116         if (!nohang) {
1117             (void) alarm(0);
1118         }
1119         if (pid <= 0) {
1120             if (waiterr == EINTR) {
1121                 if (waitflg) {
1122                     continue;
1123                 } else {
1124                     return;
1125                 }
1126             } else {
1127                 return;
1128             }
1129         }
1130         for (rp = running_list;
1131             (rp != NULL) && (rp->pid != pid);
1132             rp = rp->next) {
1133             ;
1134         }
1135         if (rp == NULL) {
1136             fatal(gettext("Internal error: returned child pid not in
1137             } else {
1138                 rp->state = (WIFEXITED(status) && WEXITSTATUS(status) ==
1139             }
1140             nohang = true;
1141             parallel_process_cnt--;

1155 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
1156     if (job_adjust_mode == ADJUST_M2) {
1157         if (m2_release_job()) {
1158             job_adjust_error();
1159         }
1160     }
1161 #endif
1162 }

```

unchanged portion omitted