

```

new/usr/src/cmd/make/bin/main.cc
*****
98035 Wed May 20 11:44:21 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for _CHECK_UPDATE_H (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 rxmlPid;
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

```

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

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                  *progrnameptr;
226     char                  *slash_ptr;
227     mode_t                um;
228     int                   i;
229 #ifdef TEAMWARE_MAKE_CNN
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     struct stat           out_stat, err_stat;
240     hostid = gethostid();
241 #ifdef TEAMWARE_MAKE_CNN
242     avo_get_user(NULL, NULL); // Initialize user name
243 #endif
244     bsd_signals();
245
246     (void) setlocale(LC_ALL, "");
247
248 #ifdef DMAKE_STATISTICS
249     if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
250         getname_stat = true;
251     }
252 #endif
253
254
255     /*
256      * avo_init() sets the umask to 0. Save it here and restore
257      * it after the avo_init() call.
258      */
259 #if defined(TEAMWARE_MAKE_CNN) || defined(MAKETOOL)
260     um = umask(0);
261     avo_init(argv[0]);
262     umask(um);
263
264     cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
265 #endif
266
267 #if defined(TEAMWARE_MAKE_CNN)
268     catd = catopen(AVO_DOMAIN_DMAKE, NL_CAT_LOCALE);
269     libcli_init();
270
271 #ifdef _CHECK_UPDATE_H
272     /* This is for dmake only (not for Solaris make).
273      * Check (in background) if there is an update (dmake patch)
274      * and inform user
275      */
276     {
277         Avo_err            *err;
278         char              *dir;
279         err = avo_find_run_dir(&dir);
280         if (AVO_OK == err) {
281
282             /*
283              * If we have a directory, then we can
284              * do a check for updates.
285              */
286
287             if (avo_update_check(err, dir)) {
288                 /* Inform user */
289             }
290
291             /* Clean up */
292             avo_free(err);
293             avo_free(dir);
294
295             /* If we have an error, then return it */
296             if (err != AVO_OK) {
297                 return err;
298             }
299
300             /* If we have no errors, then return AVO_OK */
301             return AVO_OK;
302
303         }
304
305         /* If we have an error, then return it */
306         if (err != AVO_OK) {
307             return err;
308         }
309
310         /* If we have no errors, then return AVO_OK */
311         return AVO_OK;
312
313     }
314
315 #endif
316
317     /* If we have an error, then return it */
318     if (err != AVO_OK) {
319         return err;
320     }
321
322     /* If we have no errors, then return AVO_OK */
323     return AVO_OK;
324
325 #endif
326
327 #endif
```

```

282             }
283         }
284     } /* _CHECK_UPDATE_H */
285 #endif /* _CHECK_UPDATE_H */
271 #endif

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

276 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
277 /*
278  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
279  * from avo_util library.
280  */
281     libmksdmsi18n_init();
282 #endif

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

289 #ifdef TEAMWARE_MAKE_CMN
290     g_argc = argc;
291     g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
292     for (i = 0; i < argc; i++) {
293         g_argv[i] = argv[i];
294     }
295     g_argv[i] = NULL;
296 #endif /* TEAMWARE_MAKE_CMN */

298 /*
299  * Set argv_zero_string to some form of argv[0] for
300  * recursive MAKE builds.
301  */
303     if (*argv[0] == (int) slash_char) {
304         /* argv[0] starts with a slash */
305         argv_zero_string = strdup(argv[0]);
306     } else if (strchr(argv[0], (int) slash_char) == NULL) {
307         /* argv[0] contains no slashes */
308         argv_zero_string = strdup(argv[0]);
309     } else {
310         /*
311          * argv[0] contains at least one slash,
312          * but doesn't start with a slash
313         */
314         char    *tmp_current_path;
315         char    *tmp_string;

317         tmp_current_path = get_current_path();
318         tmp_string = getmem(strlen(tmp_current_path) + 1 +
319                             strlen(argv[0]) + 1);
320         (void) sprintf(tmp_string,
321                         "%s/%s",
322                         tmp_current_path,
323                         argv[0]);
324         argv_zero_string = strdup(tmp_string);
325         retmem_mb(tmp_string);
326     }

328 /*
329  * The following flags are reset if we don't have the
330  * (.nse_depinfo or .make.state) files locked and only set
331  * AFTER the file has been locked. This ensures that if the user
332  * interrupts the program while file_lock() is waiting to lock

```

```

333             * the file, the interrupt handler doesn't remove a lock
334             * that doesn't belong to us.
335             */
336         make_state_lockfile = NULL;
337         make_state_locked = false;

340 /*
341  * look for last slash char in the path to look at the binary
342  * name. This is to resolve the hard link and invoke make
343  * in svr4 mode.
344  */

346 /* Sun OS make standard */
347 svr4 = false;
348 posix = false;
349 if (!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
350     svr4 = false;
351     posix = true;
352 } else {
353     programeptr = strrchr(argv[0], '/');
354     if (programeptr) {
355         programeptr++;
356     } else {
357         programeptr = argv[0];
358     }
359     if (!strcmp(programeptr, NOCATGETS("svr4.make"))) {
360         svr4 = true;
361         posix = false;
362     }
363 }
364 if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
365     svr4 = true;
366     posix = false;
367 }

369 /*
370  * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
371  */
372 char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"));
373 if (dmake_compat_mode_var != NULL) {
374     if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU")))
375         gnu_style = true;
376     //svr4 = false;
377     //posix = false;
378 }
379 }

381 /*
382  * Temporary directory set up.
383  */
384 char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
385 if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
386     strcpy(mbs_buffer, tmpdir_var);
387     for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
388          *tmpdir_var == '/' && tmpdir_var > mbs_buffer;
389          *tmpdir_var = '\0');
390     if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
391         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
392                         mbs_buffer, getpid());
393         int fd = mkstemp(mbs_buffer2);
394         if (fd >= 0) {
395             close(fd);
396             unlink(mbs_buffer2);
397             tmpdir = strdup(mbs_buffer);
398         }

```

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

399         }
400     }
401
402     /* find out if stdout and stderr point to the same place */
403     if (fstat(1, &out_stat) < 0) {
404         fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
405     }
406     if (fstat(2, &err_stat) < 0) {
407         fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"),
408     }
409     if ((out_stat.st_dev == err_stat.st_dev) &&
410         (out_stat.st_ino == err_stat.st_ino)) {
411         stdout_stderr_same = true;
412     } else {
413         stdout_stderr_same = false;
414     }
415     /* Make the vroot package scan the path using shell semantics */
416     set_path_style(0);

418     setup_char_semantics();

420     setup_for_projectdir();

422     /*
423      * If running with .KEEP_STATE, curdir will be set with
424      * the connected directory.
425      */
426     (void) atexit(cleanup_after_exit);

428     load_cached_names();

430     /*
431      * Set command line flags
432      */
433     setup_makeflags_argv();
434     read_command_options(mf_argc, mf_argv);
435     read_command_options(argc, argv);
436     if (debug_level > 0) {
437         cp = getenv(makeflags->string_mb);
438         (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp);
439     }

441     setup_interrupt(handle_interrupt);

443     read_files_and_state(argc, argv);

445 #ifdef TEAMWARE_MAKE_CMN
446     /*
447      * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
448      */
449     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
450     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
451     prop2 = get_prop(dmake_name2->prop, macro_prop);
452     if (prop2 == NULL) {
453         /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
454         output_mode = txt1_mode;
455     } else {
456         dmake_value2 = prop2->body.macro.value;
457         if ((dmake_value2 == NULL) ||
458             (!IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
459             output_mode = txt1_mode;
460         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2"))) {
461             output_mode = txt2_mode;
462         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))) {
463             output_mode = html1_mode;
464         } else {
```

```

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

465                                     warning(catgets(catd, 1, 352, "Unsupported value '%s' fo
466                                         dmake_value2->string_mb);
467                                     }
468                                 }
469                                 */
470                                 /* Find the dmake_mode: distributed, parallel, or serial.
471                                 */
472 if ((!pmake_cap_r_specified) &&
473     (!pmake_machinesfile_specified)) {
474     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
475     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
476     prop2 = get_prop(dmake_name2->prop, macro_prop);
477     if (prop2 == NULL) {
478         /* DMAKE_MODE not defined, default to distributed mode */
479         dmake_mode_type = distributed_mode;
480         no_parallel = false;
481     } else {
482         dmake_value2 = prop2->body.macro.value;
483         if ((dmake_value2 == NULL) ||
484             (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed")))
485             dmake_mode_type = distributed_mode;
486             no_parallel = false;
487         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel"))
488             dmake_mode_type = parallel_mode;
489             no_parallel = false;
490         } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial"))
491             dmake_mode_type = serial_mode;
492             no_parallel = true;
493         } else {
494             fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
495         }
496     }
497
498 if ((!list_all_targets) &&
499     (report_dependencies_level == 0)) {
500     /*
501     * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
502     * They could be defined in the env, in the makefile, or on the
503     * command line.
504     * If neither is defined, and ${HOME}/.dmakerc does not exists,
505     * then print a message, and default to parallel mode.
506     */
507 #ifdef DISTRIBUTED
508     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
509     dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
510     MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
511     dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
512     if (((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
513         ((dmake_value = prop->body.macro.value) == NULL)) &&
514         (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL) ||
515          ((dmake_value2 = prop2->body.macro.value) == NULL))) {
516         Boolean empty_dmakerc = true;
517         char *homedir = getenv(NOCATGETS("HOME"));
518         if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
519                                         sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
520                                         if (((statval = stat(def_dmakerc_path, &statbuf
521                                         ((statval == 0) && (statbuf.st_size == 0
522                                         } else {
523                                         Avo_dmakerc      *rcfile = new Avo_dmaker
524                                         Avo_err        *err = rcfile->read(def_
525                                         if (err) {
526                                             fatal(err->str);
527                                         }
528                                         empty_dmakerc = rcfile->was_empty();
529                                         delete rcfile;
530                                         }

```

```

531         }
532         if (empty_dmakerc) {
533             if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NULL)
534                 putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE"))
535             (void) fprintf(stdout, catgets(catd, 1,
536                                         "DMAKE_DEF_PRINTED\n"))
537             (void) fprintf(stdout, catgets(catd, 1,
538                                         "Defaulting to p\n"))
539             dmake_mode_type = parallel_mode;
540             no_parallel = false;
541         }
542     }
543     if(dmake_mode_type == distributed_mode) {
544         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mode\n"))
545         (void) fprintf(stdout, NOCATGETS("Defaulting to p\n"))
546         dmake_mode_type = parallel_mode;
547         no_parallel = false;
548     }
549 #endif /* DISTRIBUTED */
550 }
551 }
552 #endif

554 #ifdef TEAMWARE_MAKE_CMN
555     parallel_flag = true;
556     /* XXX - This is a major hack for DMake/Licensing. */
557     if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
558         if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
559             /* If the user can not get a TeamWare license,
560             * default to serial mode.
561             */
562             dmake_mode_type = serial_mode;
563             no_parallel = true;
564     } else {
565         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
566     }
567     start_time = time(NULL);
568     /*
569     * XXX - Hack to disable SIGALRM's from licensing library's
570     *       setitimer().
571     */
572     value.it_interval.tv_sec = 0;
573     value.it_interval.tv_usec = 0;
574     value.it_value.tv_sec = 0;
575     value.it_value.tv_usec = 0;
576     (void) setitimer(ITIMER_REAL, &value, NULL);
577 }
578 }

580 // If dmake is running with -t option, set dmake_mode_type to serial.
581 // This is done because doname() calls touch_command() that runs serially.
582 // If we do not do that, maketool will have problems.
583 //
584 if(touch) {
585     dmake_mode_type = serial_mode;
586     no_parallel = true;
587 }
588 }
589 #else
590     parallel_flag = false;
591 #endif

593 #if defined(TEAMWARE_MAKE_CMN) && defined(RDIRECT_ERR)
594     /*
595     * Check whether stdout and stderr are physically same.
596     * This is in order to decide whether we need to redirect

```

```

597         * stderr separately from stdout.
598         * This check is performed only if __DMAKE_SEPARATE_STDERR
599         * is not set. This variable may be used in order to preserve
600         * the 'old' behaviour.
601         */
602         out_err_same = true;
603         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
604         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS(
605             struct stat stdout_stat;
606             struct stat stderr_stat;
607             if( (fstat(1, &stdout_stat) == 0)
608                 && (fstat(2, &stderr_stat) == 0) )
609                 {
610                     if( (stdout_stat.st_dev != stderr_stat.st_dev)
611                         || (stdout_stat.st_ino != stderr_stat.st_ino) )
612                         {
613                             out_err_same = false;
614                         }
615                 }
616             } #endif

619 #ifdef DISTRIBUTED
620     /*
621     * At this point, DMake should startup an r xm with any and all
622     * DMake command line options. R xm will, among other things,
623     * read the rc file.
624     */
625     if ((!list_all_targets) &&
626         (report_dependencies_level == 0) &&
627         (dmake_mode_type == distributed_mode)) {
628         startup_rxm();
629     }
630 #endif
631 /*
632 */
633 /*
634 */
635 (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

636 /*
637 */
638 /*
639 */
640 if (getenv(sunpro_dependencies->string_mb) != NULL) {
641     FILE *report_file;

642     report_dependency("");
643     report_file = get_report_file();
644     if ((report_file != NULL) && (report_file != (FILE*)-1)) {
645         (void) fprintf(report_file, "\n");
646     }
647 }

648 */

649 /*
650 */
651 /*
652 */
653 if (keep_state) {
654     maybe_append_prop(sunpro_dependencies, macro_prop)->
655         body.macro.exported = true;
656 } else {
657     maybe_append_prop(sunpro_dependencies, macro_prop)->
658         body.macro.exported = false;
659 }

660 working_on_targets = true;
661 if (trace_status) {

```

```
663         dump_make_state();
664         fclose(stdout);
665         fclose(stderr);
666         exit_status = 0;
667         exit(0);
668     }
669     if (list_all_targets) {
670         dump_target_list();
671         fclose(stdout);
672         fclose(stderr);
673         exit_status = 0;
674         exit(0);
675     }
676     trace_reader = false;

677     /*
678      * Set temp_file_directory to the directory the .make.state
679      * file is written to.
680      */
681     if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NULL)
682         temp_file_directory = strdup(get_current_path());
683     else {
684         *slash_ptr = (int) nul_char;
685         (void) strcpy(make_state_dir, make_state->string_mb);
686         *slash_ptr = (int) slash_char;
687         /* when there is only one slash and it's the first
688          ** character, make_state_dir should point to '/'.
689          */
690         if(make_state_dir[0] == '\0') {
691             make_state_dir[0] = '/';
692             make_state_dir[1] = '\0';
693         }
694         if (make_state_dir[0] == (int) slash_char) {
695             temp_file_directory = strdup(make_state_dir);
696         } else {
697             char    tmp_current_path2[MAXPATHLEN];
698
699             (void) sprintf(tmp_current_path2,
700                           "%s/%s",
701                           get_current_path(),
702                           make_state_dir);
703             temp_file_directory = strdup(tmp_current_path2);
704         }
705     }
706 }

707 #ifdef DISTRIBUTED
708     building_serial = false;
709 #endif

710 report_dir_enter_leave(true);

711 make_targets(argc, argv, parallel_flag);

712 report_dir_enter_leave(false);

713 if (build_failed_ever_seen) {
714     if (posix) {
715         exit_status = 1;
716     }
717     exit(1);
718 }
719 exit_status = 0;
720 exit(0);
721 /* NOTREACHED */
722 }



---


723 unchanged_portion_omitted_
```

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

```
*****
53472 Wed May 20 11:44:22 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef for _CHECK_UPDATE_H (undefined)
*****
_____unchanged_portion_omitted_____
1313 /*
1314 *      await_parallel(waitflg)
1315 *
1316 *      Waits for parallel children to exit and finishes their processing.
1317 *      If waitflg is false, the function returns after update_delay.
1318 *
1319 *      Parameters:
1320 *          waitflg          dwight
1321 */
1322 void
1323 await_parallel(Boolean waitflg)
1324 {
1325 #ifdef _CHECK_UPDATE_H
1326     static int number_of_unknown_children = 0;
1327 #endif /* _CHECK_UPDATE_H */
1328
1329     Boolean      nohang;
1330     pid_t        pid;
1331     int          status;
1332     Running    rp;
1333     int          waitererr;
1334
1335     nohang = false;
1336     for ( ; ; ) {
1337         if (!nohang) {
1338             if ((void) alarm((int) update_delay);
1339             pid = waitpid((pid_t)-1,
1340                             &status,
1341                             nohang ? WNOHANG : 0);
1342             waitererr = errno;
1343             if (!nohang) {
1344                 if ((void) alarm(0);
1345                 if (pid <= 0) {
1346                     if (waitererr == EINTR) {
1347                         if (waitflg) {
1348                             continue;
1349                         } else {
1350                             return;
1351                         }
1352                     }
1353                 }
1354                 for (rp = running_list;
1355                     (rp != NULL) && (rp->pid != pid);
1356                     rp = rp->next) {
1357                     ;
1358                 }
1359                 if (rp == NULL) {
1360 #ifdef _CHECK_UPDATE_H
1361                     /* Ignore first child - it is check_update */
1362                     if (number_of_unknown_children <= 0) {
1363                         number_of_unknown_children = 1;
1364                         return;
1365                     }
1366 #endif /* _CHECK_UPDATE_H */
1367                     if (send_mtool_msgs) {
1368                         continue;
1369                     }
1370                 }
1371             }
1372         }
1373     }
1374 }
```

1

```
new/usr/src/cmd/make/bin/parallel.cc
*****
1362         } else {
1363             fatal(catgets(catd, 1, 128, "Internal error: ret
1364             }
1365         } else {
1366             rp->state = (WIFEXITED(status) && WEXITSTATUS(status) ==
1367             }
1368             nohang = true;
1369             parallel_process_cnt--;
1370
1371 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
1372     if (job_adjust_mode == ADJUST_M2) {
1373         if (m2_release_job()) {
1374             job_adjust_error();
1375         }
1376     }
1377 #endif
1378 }
1379 }
_____unchanged_portion_omitted_____
2
```