```
**********************************************************
     624 Wed May 20 11:24:52 2015
new/usr/src/cmd/make/Makefile.com
make: unifdef SUN5_0 (defined)
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 # Copyright 2015, Richard Lowe.

  14 MAKE_INCLUDE= $(SRC)/cmd/make/include
  15 MAKE_DEFS= -DSYSV -DINTER -DSUNOS4_AND_AFTER
  15 MAKE_DEFS= -DSUN5_0 -DSYSV -DINTER -DSUNOS4_AND_AFTER
  16 $(RELEASE_BUILD)MAKE_DEFS += -DNDEBUG
  17 CFLAGS += $(CCVERBOSE)
  18 CPPFLAGS += -I$(MAKE_INCLUDE) $(MAKE_DEFS)
```

```
**********************************************************
   23360 Wed May 20 11:24:53 2015
new/usr/src/cmd/make/bin/ar.cc
make: unifdef SUN5_0 (defined)
**********************************************************
```

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

/*
 *      ar.c
 *
 *      Deal with the lib.a(member.o) and lib.a((entry-point)) notations
 *
 * Look inside archives for notations a(b) and a((b))
 *      a(b)    is file member   b  in archive a
 *      a((b))  is entry point   b  in object archive a
 *
 * For 6.0, create a make which can understand all archive
 * formats.  This is kind of tricky, and <ar.h> isnt any help.
 */

/*
 * Included files
 */
#include <avo/avo_alloca.h>             /* alloca() */
#include <ar.h>
#include <errno.h>              /* errno */
#include <fcntl.h>              /* open() */
#include <mk/defs.h>
#include <mksh/misc.h>          /* retmem_mb() */

#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
struct ranlib {
        union {
                off_t   ran_strx;       /* string table index of */
                char    *ran_name;      /* symbol defined by */
        }       ran_un;
        off_t   ran_off;                /* library member at this offset */
};
#else
#include <ranlib.h>
#endif

#if defined(linux)
```



```
#include <ctype.h>              /* isspace */
#else
#include <unistd.h>             /* close() */
#endif


/*
 * Defined macros
 */
#ifndef S5EMUL
#undef BITSPERBYTE
#define BITSPERBYTE     8
#endif

/*
 * Defines for all the different archive formats.  See next comment
 * block for justification for not using <ar.h>s versions.
 */
#define AR_5_MAGIC              "<ar>"          /* 5.0 format magic string */
#define AR_5_MAGIC_LENGTH       4               /* 5.0 format string length */

#define AR_PORT_MAGIC           "!<arch>\n"     /* Port. (6.0) magic string */
#define AR_PORT_MAGIC_LENGTH    8               /* Port. (6.0) string length */
#define AR_PORT_END_MAGIC       "'\n"           /* Port. (6.0) end of header */
#define AR_PORT_WORD            4               /* Port. (6.0) 'word' length */

/*
 * typedefs & structs
 */
/*
 * These are the archive file headers for the formats.  Note
 * that it really doesnt matter if these structures are defined
 * here.  They are correct as of the respective archive format
 * releases.  If the archive format is changed, then since backwards
 * compatability is the desired behavior, a new structure is added
 * to the list.
 */
typedef struct {        /* 5.0 ar header format: vax family; 3b family */
        char                    ar_magic[AR_5_MAGIC_LENGTH];    /* AR_5_MAGIC*/
        char                    ar_name[16];    /* Space terminated */
        char                    ar_date[AR_PORT_WORD];  /* sgetl() accessed */
        char                    ar_syms[AR_PORT_WORD];  /* sgetl() accessed */
} Arh_5;
```

_____*unchanged_portion_omitted_*

```
/*
 *      open_archive(filename, arp)
 *
 *      Return value:
 *                              Indicates if open failed or not
 *
 *      Parameters:
 *              filename        The name of the archive we need to read
 *              arp             Pointer to ar file description block
 *
 *      Global variables used:
 */
static Boolean
open_archive(char *filename, register Ar *arp)
{
        int                     fd;
        char                    mag_5[AR_5_MAGIC_LENGTH];
        char                    mag_port[AR_PORT_MAGIC_LENGTH];
        char                    buffer[4];

        arp->fd = NULL;
```

```
281         fd = open_vroot(filename, O_RDONLY, 0, NULL, VROOT_DEFAULT);
282         if ((fd < 0) || ((arp->fd = fdopen(fd, "r")) == NULL)) {
283                 return failed;
284         }
285         (void) fcntl(fileno(arp->fd), F_SETFD, 1);

291 #if !defined(SUN5_0) && !defined(linux) //XXX
292         /* Read enough of the archive to distinguish between the formats */
293         if (fread(mag_5, AR_5_MAGIC_LENGTH, 1, arp->fd) != 1) {
294                 return failed;
295         }
296         if (IS_EQUALN(mag_5, AR_5_MAGIC, AR_5_MAGIC_LENGTH)) {
297                 arp->type = AR_5;
298                 /* Must read in header to set necessary info */
299                 if (fseek(arp->fd, 0L, 0) != 0 ||
300                     fread((char *) &arp->arh_5, sizeof (Arh_5), 1, arp->fd) !=
301                                                                         1) {
302                         return failed;
303                 }
304                 arp->sym_begin = ftell(arp->fd);
305                 arp->num_symbols = sgetl(arp->arh_5.ar_syms);
306                 arp->first_ar_mem = arp->sym_begin +
307                                     sizeof (Ars_5) * arp->num_symbols;
308                 arp->sym_size = 0L;
309                 return succeeded;
310         }
311         if (fseek(arp->fd, 0L, 0) != 0) {
312                 return failed;
313         }
314 #endif
287         if (fread(mag_port, AR_PORT_MAGIC_LENGTH, 1, arp->fd) != 1) {
288                 return failed;
289         }
290         if (IS_EQUALN(mag_port, AR_PORT_MAGIC, AR_PORT_MAGIC_LENGTH)) {
291                 arp->type = AR_PORT;
292                 /*
293                  * Read in first member header to find out if there is
294                  * a symbol definition table.
295                  */

297                 int ret = read_member_header(&arp->ar_port, arp->fd, filename);
298                 if (ret == failed) {
299                         return failed;
300                 } else if(ret == -1) {
301                         /* There is no member header - empty archive */
302                         arp->sym_size = arp->num_symbols = arp->sym_begin = 0L;
303                         arp->first_ar_mem = ftell(arp->fd);
304                         return succeeded;
305                 }
306                 /*
307                  * The following values are the default if there is
308                  * no symbol directory and long member names.
309                  */
310                 arp->sym_size = arp->num_symbols = arp->sym_begin = 0L;
311                 arp->first_ar_mem = ftell(arp->fd) - (long) sizeof (Ar_port);

313                 /*
314                  * Do we have a symbol table? A symbol table is always
315                  * the first member in an archive. In 4.1.x it has the
316                  * name __.SYMDEF, in SVr4, it has the name "/         "
317                  */
318 /*
319                 MBSTOWCS(wcs_buffer, "/               ");
347 #ifdef SUN5_0
348                 MBSTOWCS(wcs_buffer, NOCATGETS("/               "));
349                 if (IS_WEQUALN(arp->ar_port.ar_name, wcs_buffer, 16)) {
```



```
350 #else
351                 MBSTOWCS(wcs_buffer, NOCATGETS("__.SYMDEF       "));
320                 if (IS_WEQUALN(arp->ar_port.ar_name, wcs_buffer, 16)) {
353 #endif
321  */
355 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
322                 if (IS_EQUALN(arp->ar_port.ar_name,
323                                 NOCATGETS("/               "),
324                                 16)) {
359 #else
360                 if (IS_EQUALN(arp->ar_port.ar_name,
361                                 NOCATGETS("__.SYMDEF       "),
362                                 16)) {
363 #endif
325                         if (sscanf(arp->ar_port.ar_size,
326                                    "%ld",
327                                    &arp->sym_size) != 1) {
328                                 return failed;
329                         }
330                         arp->sym_size += (arp->sym_size & 1); /* round up */
331                         if (fread(buffer, sizeof buffer, 1, arp->fd) != 1) {
332                                 return failed;
333                         }
334                         arp->num_symbols = sgetl(buffer);
335                         arp->sym_begin = ftell(arp->fd);
336                         arp->first_ar_mem = arp->sym_begin +
337                                             arp->sym_size - sizeof buffer;
338                 }
339                 return succeeded;
340         }
341         fatal(catgets(catd, 1, 3, "‘%s’ is not an archive"), filename);
342         /* NOTREACHED */
343         return failed;
344 }
```

_____***unchanged_portion_omitted_***

```
363 /*
364  *      read_archive_dir(arp, library, long_names_table)
365  *
366  *      Reads the directory of an archive and enters all
367  *      the members into the make symboltable in lib(member) format
368  *      with their dates.
369  *
370  *      Parameters:
371  *              arp             Pointer to ar file description block
372  *              library         Name of lib to enter members for.
373  *                              Used to form "lib(member)" string.
374  *              long_names_table table that contains list of members
375  *                              with names > 15 characters long
376  *
377  *      Global variables used:
378  */
379 static Boolean
419 #if defined(SUN5_0) || defined(linux) //XXX
380 read_archive_dir(register Ar *arp, Name library, char **long_names_table)
421 #else
422 read_archive_dir(register Ar *arp, Name library, char **)
423 #endif
381 {
382         wchar_t                 *name_string;
383         wchar_t                 *member_string;
384         register long           len;
385         register wchar_t        *p;
386         register char           *q;
387         register Name           name;
388         Property                member;
```

```
389         long                    ptr;
390         long                    date;

435 #if defined(SUN5_0) || defined(linux) //XXX
392         int                     offset;

394         /*
395          * If any of the members has a name > 15 chars,
396          * it will be found here.
397          */
398         if (process_long_names_member(arp, long_names_table, library->string_mb)
399                 return failed;
400         }
445 #endif
401         name_string = ALLOC_WC((int) (library->hash.length +
402                                       (int) ar_member_name_len * 2));
403         (void) mbstowcs(name_string, library->string_mb, (int) library->hash.len
404         member_string = name_string + library->hash.length;
405         *member_string++ = (int) parenleft_char;

407         if (fseek(arp->fd, arp->first_ar_mem, 0) != 0) {
408                 goto read_error;
409         }
410         /* Read the directory using the appropriate format */
411         switch (arp->type) {
412         case AR_5:
413             for (;;) {
414                 if (fread((char *) &arp->arf_5, sizeof arp->arf_5, 1, arp->fd)
415                     != 1) {
416                         if (feof(arp->fd)) {
417                                 return succeeded;
418                         }
419                         break;
420                 }
421                 len = sizeof arp->arf_5.arf_name;
422                 for (p = member_string, q = arp->arf_5.arf_name;
423                      (len > 0) && (*q != (int) nul_char) && !isspace(*q);
424                      ) {
425                         MBTOWC(p, q);
426                         p++;
427                         q++;
428                 }
429                 *p++ = (int) parenright_char;
430                 *p = (int) nul_char;
431                 name = GETNAME(name_string, FIND_LENGTH);
432                 /*
433                  * [tolik] Fix for dmake bug 1234018.
434                  * If name->stat.time is already set, then it should not
435                  * be changed. (D)make propogates time stamp for one
436                  * member, and when it calls exists() for another member,
437                  * the first one may be changed.
438                  */
439                 if(name->stat.time == file_no_time) {
440                         name->stat.time.tv_sec = sgetl(arp->arf_5.arf_date);
441                         name->stat.time.tv_nsec = LONG_MAX;
442                 }
443                 name->is_member = library->is_member;
444                 member = maybe_append_prop(name, member_prop);
445                 member->body.member.library = library;
446                 *--p = (int) nul_char;
447                 if (member->body.member.member == NULL) {
448                         member->body.member.member =
449                           GETNAME(member_string, FIND_LENGTH);
450                 }
451                 ptr = sgetl(arp->arf_5.arf_size);
452                 ptr += (ptr & 1);
```



```
453                 if (fseek(arp->fd, ptr, 1) != 0) {
454                         goto read_error;
455                 }
456             }
457             break;
458         case AR_PORT:
459             for (;;) {
460                 if ((fread((char *) &arp->ar_port,
461                            sizeof arp->ar_port,
462                            1,
463                            arp->fd) != 1) ||
464                     !IS_EQUALN(arp->ar_port.ar_fmag,
465                                AR_PORT_END_MAGIC,
466                                sizeof arp->ar_port.ar_fmag)) {
467                     if (feof(arp->fd)) {
468                             return succeeded;
469                     }
470                     fatal(
471                         catgets(catd, 1, 28, "Read error in archive '%s'
472                         library->string_mb,
473                         ftell(arp->fd)
474                     );
475                 }
521 #if defined(SUN5_0) || defined(linux) //XXX
476                 /* If it's a long name, retrieve it from long name table */
477                 if (arp->ar_port.ar_name[0] == '/') {
478                         /*
479                          * "len" is used for hashing the string.
480                          * We're using "ar_member_name_len" instead of
481                          * the actual name length since it's the longest
482                          * string the "ar" command can handle at this
483                          * point.
484                          */
485                         len = ar_member_name_len;
486                         sscanf(arp->ar_port.ar_name + 1,
487                                "%ld",
488                                &offset);
489                         q = *long_names_table + offset;
490                 } else {
491                         q = arp->ar_port.ar_name;
492                         len = sizeof arp->ar_port.ar_name;
493                 }
540 #else
541                 q = arp->ar_port.ar_name;
542                 len = sizeof arp->ar_port.ar_name;
543 #endif
494
495                 for (p = member_string;
496                      (len > 0) &&
497                      (*q != (int) nul_char) &&
498                      !isspace(*q) &&
499                      (*q != (int) slash_char);
500                      ) {
501                         MBTOWC(p, q);
502                         p++;
503                         q++;
504                 }
505                 *p++ = (int) parenright_char;
506                 *p = (int) nul_char;
507                 name = GETNAME(name_string, FIND_LENGTH);
508                 name->is_member = library->is_member;
509                 member = maybe_append_prop(name, member_prop);
510                 member->body.member.library = library;
511                 *--p = (int) nul_char;
512                 if (member->body.member.member == NULL) {
513                         member->body.member.member =
```

```
514                             GETNAME(member_string, FIND_LENGTH);
515                     }
516                     if (sscanf(arp->ar_port.ar_date, "%ld", &date) != 1) {
517                             WCSTOMBS(mbs_buffer, name_string);
518                             fatal(catgets(catd, 1, 4, "Bad date field for member
519                                   mbs_buffer,
520                                   library->string_mb);
521                     }
522                     /*
523                      * [tolik] Fix for dmake bug 1234018.
524                      */
525                     if(name->stat.time == file_no_time) {
526                         name->stat.time.tv_sec = date;
527                         name->stat.time.tv_nsec = LONG_MAX;
528                     }
529                     if (sscanf(arp->ar_port.ar_size, "%ld", &ptr) != 1) {
530                             WCSTOMBS(mbs_buffer, name_string);
531                             fatal(catgets(catd, 1, 5, "Bad size field for member
532                                   mbs_buffer,
533                                   library->string_mb);
534                     }
535                     ptr += (ptr & 1);
536                     if (fseek(arp->fd, ptr, 1) != 0) {
537                             goto read_error;
538                     }
539             }
540             break;
541         }

543         /* Only here if fread() [or IS_EQUALN()] failed and not at EOF */
544 read_error:
545         fatal(catgets(catd, 1, 6, "Read error in archive `%s': %s"),
546               library->string_mb,
547               errmsg(errno));
548             /* NOTREACHED */
549 }
```

*unchanged_portion_omitted_*

```
607 /*
608  *      translate_entry(arp, target, member)
609  *
610  *      Finds the member for one lib.a((entry))
611  *
612  *      Parameters:
613  *              arp             Pointer to ar file description block
614  *              target          Target to find member name for
615  *              member          Property to fill in with info
616  *
617  *      Global variables used:
618  */
619 static void
620 translate_entry(register Ar *arp, Name target, register Property member, char **
621 {
622         register int            len;
623         register int            i;
624         wchar_t                 *member_string;
625         ar_port_word            *offs;
626         int                     strtablen;
627         char                    *syms;          /* string table */
628         char                    *csym;          /* string table */
629         ar_port_word            *offend;        /* end of offsets table */
630         int                     date;
631         register wchar_t        *ap;
632         register char           *hp;
633         int                     maxs;
634         int                     offset;
```



```
635         char            buffer[4];

637         if (arp->sym_begin == 0L || arp->num_symbols == 0L) {
638                 fatal(catgets(catd, 1, 8, "Cannot find symbol `%s' in archive `%
639                       member->body.member.entry->string_mb,
640                       member->body.member.library->string_mb);
641         }

643         if (fseek(arp->fd, arp->sym_begin, 0) != 0) {
644                 goto read_error;
645         }
646         member_string = ALLOC_WC((int) ((int) ar_member_name_len * 2));

648         switch (arp->type) {
649         case AR_5:
650                 if ((len = member->body.member.entry->hash.length) > 8) {
651                         len = 8;
652                 }
653                 for (i = 0; i < arp->num_symbols; i++) {
654                         if (fread((char *) &arp->ars_5,
655                                   sizeof arp->ars_5,
656                                   1,
657                                   arp->fd) != 1) {
658                                 goto read_error;
659                         }
660                         if (IS_EQUALN(arp->ars_5.sym_name,
661                                       member->body.member.entry->string_mb,
662                                       len)) {
663                                 if ((fseek(arp->fd,
664                                           sgetl(arp->ars_5.sym_ptr),
665                                           0) != 0) ||
666                                     (fread((char *) &arp->arf_5,
667                                           sizeof arp->arf_5,
668                                           1,
669                                           arp->fd) != 1)) {
670                                         goto read_error;
671                                 }
672                                 MBSTOWCS(wcs_buffer, arp->arf_5.arf_name);
673                                 (void) wsncpy(member_string,
674                                               wcs_buffer,
675                                               wslen(wcs_buffer));
676                                 member_string[sizeof(arp->arf_5.arf_name)] =
677                                                                     (int) nul_char;
678                                 member->body.member.member =
679                                       GETNAME(member_string, FIND_LENGTH);
680                                 target->stat.time.tv_sec = sgetl(arp->arf_5.arf_
681                                 target->stat.time.tv_nsec = LONG_MAX;
682                                 return;
683                         }
684                 }
685                 break;
686         case AR_PORT:
687                 offs = (ar_port_word *) alloca((int) (arp->num_symbols * AR_PORT
688                 if (fread((char *) offs,
689                           AR_PORT_WORD,
690                           (int) arp->num_symbols,
691                           arp->fd) != arp->num_symbols) {
692                         goto read_error;
693                 }

695                 for(i=0;i<arp->num_symbols;i++) {
696                         *((int*)buffer)=offs[i];
697                         offs[i]=(ar_port_word)sgetl(buffer);
698                 }

700                 strtablen=arp->sym_size-4-(int) (arp->num_symbols * AR_PORT_WORD
```

```
701                 syms = (char *) alloca(strtablen);
702                 if (fread(syms,
703                           sizeof (char),
704                           strtablen,
705                           arp->fd) != strtablen) {
706                         goto read_error;
707                 }
708                 offend = &offs[arp->num_symbols];
709                 while (offs < offend) {
710                         maxs = strlen(member->body.member.entry->string_mb);
711                         if(strlen(syms) > maxs)
712                                 maxs = strlen(syms);
713                         if (IS_EQUALN(syms,
714                                       member->body.member.entry->string_mb,
715                                       maxs)) {
716                                 if (fseek(arp->fd,
717                                           (long) *offs,
718                                           0) != 0) {
719                                         goto read_error;
720                                 }
721                                 if ((fread((char *) &arp->ar_port,
722                                            sizeof arp->ar_port,
723                                            1,
724                                            arp->fd) != 1) ||
725                                     !IS_EQUALN(arp->ar_port.ar_fmag,
726                                                AR_PORT_END_MAGIC,
727                                                sizeof arp->ar_port.ar_fmag)) {
728                                         goto read_error;
729                                 }
730                                 if (sscanf(arp->ar_port.ar_date,
731                                            "%ld",
732                                            &date) != 1) {
733                                         fatal(catgets(catd, 1, 9, "Bad date fiel
734                                               arp->ar_port.ar_name,
735                                               target->string_mb);
736                                 }
737 #if defined(SUN5_0) || defined(linux) //XXX
738                                 /* If it's a long name, retrieve it from long name table */
739                                 if (arp->ar_port.ar_name[0] == '/') {
740                                         sscanf(arp->ar_port.ar_name + 1,
741                                                "%ld",
742                                                &offset);
743                                         len = ar_member_name_len;
744                                         hp = *long_names_table + offset;
745                                 } else {
746                                         len = sizeof arp->ar_port.ar_name;
747                                         hp = arp->ar_port.ar_name;
748                                 }
749 #else
750                                 hp = arp->ar_port.ar_name;
751 #endif
752                                 ap = member_string;
753                                 while (*hp &&
754                                        (*hp != (int) slash_char) &&
755                                        (ap < &member_string[len])) {
756                                         MBTOWC(ap, hp);
757                                         ap++;
758                                         hp++;
759                                 }
760                                 *ap = (int) nul_char;
761                                 member->body.member.member =
762                                         GETNAME(member_string, FIND_LENGTH);
```



```
763                                 target->stat.time.tv_sec = date;
764                                 target->stat.time.tv_nsec = LONG_MAX;
765                                 return;
766                         }
767                 }
768                 fatal(catgets(catd, 1, 10, "Cannot find symbol `%s' in archive `%s'"),
769                       member->body.member.entry->string_mb,
770                       member->body.member.library->string_mb);
771                 /*NOTREACHED*/
772
773 read_error:
774                 if (ferror(arp->fd)) {
775                         fatal(catgets(catd, 1, 11, "Read error in archive `%s': %s"),
776                               member->body.member.library->string_mb,
777                               errmsg(errno));
778                 } else {
779                         fatal(catgets(catd, 1, 12, "Read error in archive `%s': Prematur
780                               member->body.member.library->string_mb);
781                 }
782 }
```

*unchanged_portion_omitted_*

```
**********************************************************
  105329 Wed May 20 11:24:54 2015
new/usr/src/cmd/make/bin/doname.cc
make: unifdef SUN5_0 (defined)
**********************************************************
______unchanged_portion_omitted_

/*
 * DONE.
 *
 *      run_command(line)
 *
 *      Takes one Cmd_line and runs the commands from it.
 *
 *      Return value:
 *                              Indicates if the command failed or not
 *
 *      Parameters:
 *              line            The command line to run
 *
 *      Global variables used:
 *              commands_done   Set if we do run command
 *              current_line    Set to the line we run a command from
 *              current_target  Set to the target we run a command for
 *              file_number     Used to form temp file name
 *              keep_state      Indicates that .KEEP_STATE is on
 *              make_state      The Name ".make.state", used to check timestamp
 *              parallel        True if currently building in parallel
 *              parallel_process_cnt Count of parallel processes running
 *              quest           Indicates that make -q is on
 *              rewrite_statefile Set if we do run a command
 *              sunpro_dependencies The Name "SUNPRO_DEPENDENCIES", set value
 *              temp_file_directory Used to form temp fie name
 *              temp_file_name  Set to the name of the temp file
 *              touch           Indicates that make -t is on
 */
static Doname
run_command(register Property line, Boolean)
{
        register Doname         result = build_ok;
        register Boolean        remember_only = false;
        register Name           target = line->body.line.target;
        wchar_t                 *string;
        char                    tmp_file_path[MAXPATHLEN];

        if (!line->body.line.is_out_of_date && target->rechecking_target) {
                target->rechecking_target = false;
                return build_ok;
        }

        /*
         * Build the command if we know the target is out of date,
         * or if we want to check cmd consistency.
         */
        if (line->body.line.is_out_of_date || keep_state) {
                /* Hack for handling conditional macros in DMake. */
                if (!line->body.line.dont_rebuild_command_used) {
                        build_command_strings(target, line);
                }
        }
        /* Never mind */
        if (!line->body.line.is_out_of_date) {
                return build_ok;
        }
        /* If quest, then exit(1) because the target is out of date */
        if (quest) {
```



```
                if (posix) {
#ifdef TEAMWARE_MAKE_CMN
                        result = execute_parallel(line, true);
#else
                        result = execute_serial(line);
#endif
                }
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
                exit_status = 1;
#endif
                exit(1);
        }
        /* We actually had to do something this time */
        rewrite_statefile = commands_done = true;
        /*
         * If this is an sccs command, we have to do some extra checking
         * and possibly complain. If the file can't be gotten because it's
         * checked out, we complain and behave as if the command was
         * executed eventhough we ignored the command.
         */
        if (!touch &&
            line->body.line.sccs_command &&
            (target->stat.time != file_doesnt_exist) &&
            ((target->stat.mode & 0222) != 0)) {
                fatal(catgets(catd, 1, 27, "%s is writable so it cannot be sccs
                      target->string_mb);
                target->has_complained = remember_only = true;
        }
        /*
         * If KEEP_STATE is on, we make sure we have the timestamp for
         * .make.state. If .make.state changes during the command run,
         * we reread .make.state after the command. We also setup the
         * environment variable that asks utilities to report dependencies.
         */
        if (!touch &&
            keep_state &&
            !remember_only) {
                (void) exists(make_state);
                if((strlen(temp_file_directory) == 1) &&
                        (temp_file_directory[0] == '/')) {
                   tmp_file_path[0] = '\0';
                } else {
                   strcpy(tmp_file_path, temp_file_directory);
                }
                sprintf(mbs_buffer,
                                NOCATGETS("%s/.make.dependency.%08x.%d.%d"),
                                tmp_file_path,
                                hostid,
                                getpid(),
                                file_number++);
                MBSTOWCS(wcs_buffer, mbs_buffer);
                Boolean fnd;
                temp_file_name = getname_fn(wcs_buffer, FIND_LENGTH, false, &fnd
                temp_file_name->stat.is_file = true;
                int len = 2*MAXPATHLEN + strlen(target->string_mb) + 2;
                wchar_t *to = string = ALLOC_WC(len);
                for (wchar_t *from = wcs_buffer; *from != (int) nul_char; ) {
                        if (*from == (int) space_char) {
                                *to++ = (int) backslash_char;
                        }
                        *to++ = *from++;
                }
                *to++ = (int) space_char;
                MBSTOWCS(to, target->string_mb);
                Name sprodep_name = getname_fn(string, FIND_LENGTH, false, &fnd)
                (void) SETVAR(sunpro_dependencies,
```

```
1763                                sprodep_name,
1764                                false);
1765                        retmem(string);
1766                } else {
1767                        temp_file_name = NULL;
1768                }

1770                /*
1771                 * In case we are interrupted, we need to know what was going on.
1772                 */
1773                current_target = target;
1774                /*
1775                 * We also need to be able to save an empty command instead of the
1776                 * interrupted one in .make.state.
1777                 */
1778                current_line = line;
1779                if (remember_only) {
1780                        /* Empty block!!! */
1781                } else if (touch) {
1782                        result = touch_command(line, target, result);
1783                        if (posix) {
1784 #ifdef TEAMWARE_MAKE_CMN
1785                                result = execute_parallel(line, true);
1786 #else
1787                                result = execute_serial(line);
1788 #endif
1789                        }
1790                } else {
1791                        /*
1792                         * If this is not a touch run, we need to execute the
1793                         * proper command(s) for the target.
1794                         */
1795 #ifdef TEAMWARE_MAKE_CMN
1796                        if (parallel) {
1797                                if (!parallel_ok(target, true)) {
1798                                        /*
1799                                         * We are building in parallel, but
1800                                         * this target must be built in serial.
1801                                         */
1802                                        /*
1803                                         * If nothing else is building,
1804                                         * do this one, else wait.
1805                                         */
1806                                        if (parallel_process_cnt == 0) {
1807 #ifdef TEAMWARE_MAKE_CMN
1808                                                result = execute_parallel(line, true, ta
1809 #else
1810                                                result = execute_serial(line);
1811 #endif
1812                                        } else {
1813                                                current_target = NULL;
1814                                                current_line = NULL;
1815 /*
1816                                                line->body.line.command_used = NULL;
1817  */
1818                                                line->body.line.dont_rebuild_command_use
1819                                                return build_serial;
1820                                        }
1821                                } else {
1822                                        result = execute_parallel(line, false);
1823                                        switch (result) {
1824                                        case build_running:
1825                                                return build_running;
1826                                        case build_serial:
1827                                                if (parallel_process_cnt == 0) {
1828 #ifdef TEAMWARE_MAKE_CMN
```



```
1829                                                result = execute_parallel(line,
1830 #else
1831                                                result = execute_serial(line);
1832 #endif
1833                                        } else {
1834                                                current_target = NULL;
1835                                                current_line = NULL;
1836                                                target->parallel = false;
1837                                                line->body.line.command_used =
1838                                                                        NULL;
1839                                                return build_serial;
1840                                        }
1841                                }
1842                        }
1843                } else {
1844 #endif
1845 #ifdef TEAMWARE_MAKE_CMN
1846                        result = execute_parallel(line, true, target->localhost)
1847 #else
1848                        result = execute_serial(line);
1849 #endif
1850 #ifdef TEAMWARE_MAKE_CMN
1851                }
1852 #endif
1853        }
1854        temp_file_name = NULL;
1855        if (report_dependencies_level == 0){
1856                update_target(line, result);
1857        }
1858        current_target = NULL;
1859        current_line = NULL;
1860        return result;
1861 }
```

unchanged_portion_omitted_

```
**********************************************************
   18661 Wed May 20 11:24:55 2015
new/usr/src/cmd/make/bin/files.cc
make: unifdef SUN5_0 (defined)
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      files.c
 *
 *      Various file related routines:
 *              Figure out if file exists
 *              Wildcard resolution for directory reader
 *              Directory reader
 */


/*
 * Included files
 */
#if defined(SUN5_0) || defined(HP_UX)
#include <dirent.h>             /* opendir() */
#else
#include <sys/dir.h>            /* opendir() */
#endif
#include <errno.h>              /* errno */
#include <mk/defs.h>
#include <mksh/macro.h>         /* getvar() */
#include <mksh/misc.h>          /* get_prop(), append_prop() */
#include <sys/stat.h>           /* lstat() */

/*
 * Defined macros
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
```



```
/*
 * File table of contents
 */
extern  timestruc_t&    exists(register Name target);
extern  void            set_target_stat(register Name target, struct stat buf);
static  timestruc_t&    vpath_exists(register Name target);
static  Name            enter_file_name(wchar_t *name_string, wchar_t *library);
static  Boolean         star_match(register char *string, register char *pattern
static  Boolean         amatch(register wchar_t *string, register wchar_t *patte

/*
 *      exists(target)
 *
 *      Figure out the timestamp for one target.
 *
 *      Return value:
 *                              The time the target was created
 *
 *      Parameters:
 *              target          The target to check
 *
 *      Global variables used:
 *              debug_level     Should we trace the stat call?
 *              recursion_level Used for tracing
 *              vpath_defined   Was the variable VPATH defined in environment?
 */
timestruc_t&
exists(register Name target)
{
        struct stat             buf;
        register int            result;

        /* We cache stat information. */
        if (target->stat.time != file_no_time) {
                return target->stat.time;
        }

        /*
         * If the target is a member, we have to extract the time
         * from the archive.
         */
        if (target->is_member &&
            (get_prop(target->prop, member_prop) != NULL)) {
                return read_archive(target);
        }

        if (debug_level > 1) {
                (void) printf(NOCATGETS("%*sstat(%s)\n"),
                              recursion_level,
                              "",
                              target->string_mb);
        }

        result = lstat_vroot(target->string_mb, &buf, NULL, VROOT_DEFAULT);
        if ((result != -1) && ((buf.st_mode & S_IFMT) == S_IFLNK)) {
                /*
                 * If the file is a symbolic link, we remember that
                 * and then we get the status for the refd file.
                 */
                target->stat.is_sym_link = true;
                result = stat_vroot(target->string_mb, &buf, NULL, VROOT_DEFAULT
        } else {
                target->stat.is_sym_link = false;
        }

        if (result < 0) {
```

```
124                 target->stat.time = file_doesnt_exist;
125                 target->stat.stat_errno = errno;
126                 if ((errno == ENOENT) &&
127                     vpath_defined &&
128 /* azv, fixing bug 1262942, VPATH works with a leaf name
129  * but not a directory name.
130  */
131                     (target->string_mb[0] != (int) slash_char) ) {
132 /* BID_1214655 */
133 /* azv */
134                         vpath_exists(target);
135                         // return vpath_exists(target);
136                 }
137         } else {
138                 /* Save all the information we need about the file */
139                 target->stat.stat_errno = 0;
140                 target->stat.is_file = true;
141                 target->stat.mode = buf.st_mode & 0777;
142                 target->stat.size = buf.st_size;
143                 target->stat.is_dir =
144                   BOOLEAN((buf.st_mode & S_IFMT) == S_IFDIR);
145                 if (target->stat.is_dir) {
146                         target->stat.time = file_is_dir;
147                 } else {
148                         /* target->stat.time = buf.st_mtime; */
149 /* BID_1129806 */
150 /* vis@nbsp.nsk.su */
151 #if defined(linux)
152                         timestruc_t ttime = { buf.st_mtime, 0 };
153                         target->stat.time = MAX(ttime, file_min_time);
154 #else
155                         target->stat.time = MAX(buf.st_mtim, file_min_time);
156 #endif
157                 }
158         }
159         if ((target->colon_splits > 0) &&
160             (get_prop(target->prop, time_prop) == NULL)) {
161                 append_prop(target, time_prop)->body.time.time =
162                   target->stat.time;
163         }
164         return target->stat.time;
165 }
_____unchanged_portion_omitted_

269 /*
270  *      read_dir(dir, pattern, line, library)
271  *
272  *      Used to enter the contents of directories into makes namespace.
273  *      Presence of a file is important when scanning for implicit rules.
274  *      read_dir() is also used to expand wildcards in dependency lists.
275  *
276  *      Return value:
277  *                              Non-0 if we found files to match the pattern
278  *
279  *      Parameters:
280  *              dir             Path to the directory to read
281  *              pattern         Pattern for that files should match or NULL
282  *              line            When we scan using a pattern we enter files
283  *                              we find as dependencies for this line
284  *              library         If we scan for "lib.a(<wildcard-member>)"
285  *
286  *      Global variables used:
287  *              debug_level     Should we trace the dir reading?
288  *              dot             The Name ".", compared against
289  *              sccs_dir_path   The path to the SCCS dir (from PROJECTDIR)
290  *              vpath_defined   Was the variable VPATH defined in environment?
```



```
291  *              vpath_name      The Name "VPATH", use to get macro value
292  */
293 int
294 read_dir(Name dir, wchar_t *pattern, Property line, wchar_t *library)
295 {
296         wchar_t                 file_name[MAXPATHLEN];
297         wchar_t                 *file_name_p = file_name;
298         Name                    file;
299         wchar_t                 plain_file_name[MAXPATHLEN];
300         wchar_t                 *plain_file_name_p;
301         Name                    plain_file;
302         wchar_t                 tmp_wcs_buffer[MAXPATHLEN];
303         DIR                     *dir_fd;
304         int                     m_local_dependency=0;
309 #if defined(SUN5_0) || defined(HP_UX)
305 #define d_fileno d_ino
306         register struct dirent  *dp;
312 #else
313         register struct direct  *dp;
314 #endif
307         wchar_t                 *vpath = NULL;
308         wchar_t                 *p;
309         int                     result = 0;

311         if(dir->hash.length >= MAXPATHLEN) {
312                 return 0;
313         }

315         Wstring wcb(dir);
316         Wstring vps;

318         /* A directory is only read once unless we need to expand wildcards. */
319         if (pattern == NULL) {
320                 if (dir->has_read_dir) {
321                         return 0;
322                 }
323                 dir->has_read_dir = true;
324         }
325         /* Check if VPATH is active and setup list if it is. */
326         if (vpath_defined && (dir == dot)) {
327                 vps.init(getvar(vpath_name));
328                 vpath = vps.get_string();
329         }

331         /*
332          * Prepare the string where we build the full name of the
333          * files in the directory.
334          */
335         if ((dir->hash.length > 1) || (wcb.get_string()[0] != (int) period_char)
336                 (void) wscpy(file_name, wcb.get_string());
337                 MBSTOWCS(wcs_buffer, "/");
338                 (void) wscat(file_name, wcs_buffer);
339                 file_name_p = file_name + wslen(file_name);
340         }

342         /* Open the directory. */
343 vpath_loop:
344         dir_fd = opendir(dir->string_mb);
345         if (dir_fd == NULL) {
346                 return 0;
347         }

349         /* Read all the directory entries. */
350         while ((dp = readdir(dir_fd)) != NULL) {
351                 /* We ignore "." and ".." */
352                 if ((dp->d_fileno == 0) ||
```

```
353                     ((dp->d_name[0] == (int) period_char) &&
354                      ((dp->d_name[1] == 0) ||
355                       ((dp->d_name[1] == (int) period_char) &&
356                        (dp->d_name[2] == 0))))) {
357                         continue;
358                 }
359                 /*
360                  * Build the full name of the file using whatever
361                  * path supplied to the function.
362                  */
363                 MBSTOWCS(tmp_wcs_buffer, dp->d_name);
364                 (void) wscpy(file_name_p, tmp_wcs_buffer);
365                 file = enter_file_name(file_name, library);
366                 if ((pattern != NULL) && amatch(tmp_wcs_buffer, pattern)) {
367                         /*
368                          * If we are expanding a wildcard pattern, we
369                          * enter the file as a dependency for the target.
370                          */
371                         if (debug_level > 0){
372                                 WCSTOMBS(mbs_buffer, pattern);
373                                 (void) printf(catgets(catd, 1, 231, "'%s: %s' du
374                                               line->body.line.target->string_mb,
375                                               file->string_mb,
376                                               mbs_buffer);
377                         }
378                         enter_dependency(line, file, false);
379                         result++;
380                 } else {
381                         /*
382                          * If the file has an SCCS/s. file,
383                          * we will detect that later on.
384                          */
385                         file->stat.has_sccs = NO_SCCS;
386                 /*
387                  * If this is an s. file, we also enter it as if it
388                  * existed in the plain directory.
389                  */
390                 if ((dp->d_name[0] == 's') &&
391                     (dp->d_name[1] == (int) period_char)) {
392
393                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
394                         plain_file_name_p = plain_file_name;
395                         (void) wscpy(plain_file_name_p, tmp_wcs_buffer);
396                         plain_file = GETNAME(plain_file_name, FIND_LENGTH);
397                         plain_file->stat.is_file = true;
398                         plain_file->stat.has_sccs = HAS_SCCS;
399                         /*
400                          * Enter the s. file as a dependency for the
401                          * plain file.
402                          */
403                         maybe_append_prop(plain_file, sccs_prop)->
404                           body.sccs.file = file;
405                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
406                         if ((pattern != NULL) &&
407                             amatch(tmp_wcs_buffer, pattern)) {
408                                 if (debug_level > 0) {
409                                         WCSTOMBS(mbs_buffer, pattern);
410                                         (void) printf(catgets(catd, 1, 232, "'%s
411                                                       line->body.line.target->
412                                                       string_mb,
413                                                       plain_file->string_mb,
414                                                       mbs_buffer);
415                                 }
416                                 enter_dependency(line, plain_file, false);
417                                 result++;
418                         }
```



```
419                         }
420                     }
421                 }
422                 (void) closedir(dir_fd);
423                 if ((vpath != NULL) && (*vpath != (int) nul_char)) {
424                         while ((*vpath != (int) nul_char) &&
425                                (iswspace(*vpath) || (*vpath == (int) colon_char))) {
426                                 vpath++;
427                         }
428                         p = vpath;
429                         while ((*vpath != (int) colon_char) &&
430                                (*vpath != (int) nul_char)) {
431                                 vpath++;
432                         }
433                         if (vpath > p) {
434                                 dir = GETNAME(p, vpath - p);
435                                 goto vpath_loop;
436                         }
437                 }
438         /*
439          * look into SCCS directory only if it's not svr4. For svr4 dont do that.
440          */
441
442         /*
443          * Now read the SCCS directory.
444          * Files in the SCSC directory are considered to be part of the set of
445          * files in the plain directory. They are also entered in their own right.
446          * Prepare the string where we build the true name of the SCCS files.
447          */
448                 (void) wsncpy(plain_file_name,
449                               file_name,
450                               file_name_p - file_name);
451                 plain_file_name[file_name_p - file_name] = 0;
452                 plain_file_name_p = plain_file_name + wslen(plain_file_name);
453
454                 if(!svr4) {
455
456                   if (sccs_dir_path != NULL) {
457                         wchar_t         tmp_wchar;
458                         wchar_t         path[MAXPATHLEN];
459                         char            mb_path[MAXPATHLEN];
460
461                         if (file_name_p - file_name > 0) {
462                                 tmp_wchar = *file_name_p;
463                                 *file_name_p = 0;
464                                 WCSTOMBS(mbs_buffer, file_name);
465                                 (void) sprintf(mb_path, NOCATGETS("%s/%s/SCCS"),
466                                                 sccs_dir_path,
467                                                 mbs_buffer);
468                                 *file_name_p = tmp_wchar;
469                         } else {
470                                 (void) sprintf(mb_path, NOCATGETS("%s/SCCS"), sccs_dir_p
471                         }
472                         MBSTOWCS(path, mb_path);
473                         (void) wscpy(file_name, path);
474                   } else {
475                         MBSTOWCS(wcs_buffer, NOCATGETS("SCCS"));
476                         (void) wscpy(file_name_p, wcs_buffer);
477                   }
478                 } else {
479                         MBSTOWCS(wcs_buffer, NOCATGETS("."));
480                         (void) wscpy(file_name_p, wcs_buffer);
481                 }
482                 /* Internalize the constructed SCCS dir name. */
483                 (void) exists(dir = GETNAME(file_name, FIND_LENGTH));
484                 /* Just give up if the directory file doesnt exist. */
```

```
485         if (!dir->stat.is_file) {
486                 return result;
487         }
488         /* Open the directory. */
489         dir_fd = opendir(dir->string_mb);
490         if (dir_fd == NULL) {
491                 return result;
492         }
493         MBSTOWCS(wcs_buffer, "/");
494         (void) wscat(file_name, wcs_buffer);
495         file_name_p = file_name + wslen(file_name);

497         while ((dp = readdir(dir_fd)) != NULL) {
498                 if ((dp->d_fileno == 0) ||
499                     ((dp->d_name[0] == (int) period_char) &&
500                      ((dp->d_name[1] == 0) ||
501                       ((dp->d_name[1] == (int) period_char) &&
502                        (dp->d_name[2] == 0))))) {
503                         continue;
504                 }
505                 /* Construct and internalize the true name of the SCCS file. */
506                 MBSTOWCS(wcs_buffer, dp->d_name);
507                 (void) wscpy(file_name_p, wcs_buffer);
508                 file = GETNAME(file_name, FIND_LENGTH);
509                 file->stat.is_file = true;
510                 file->stat.has_sccs = NO_SCCS;
511                 /*
512                  * If this is an s. file, we also enter it as if it
513                  * existed in the plain directory.
514                  */
515                 if ((dp->d_name[0] == 's') &&
516                     (dp->d_name[1] == (int) period_char)) {
517
518                         MBSTOWCS(wcs_buffer, dp->d_name + 2);
519                         (void) wscpy(plain_file_name_p, wcs_buffer);
520                         plain_file = GETNAME(plain_file_name, FIND_LENGTH);
521                         plain_file->stat.is_file = true;
522                         plain_file->stat.has_sccs = HAS_SCCS;
523                                 /* if sccs dependency is already set,skip */
524                         if(plain_file->prop) {
525                                 Property sprop = get_prop(plain_file->prop,sccs_
526                                 if(sprop != NULL) {
527                                         if (sprop->body.sccs.file) {
528                                                 goto try_pattern;
529                                         }
530                                 }
531                         }

533                         /*
534                          * Enter the s. file as a dependency for the
535                          * plain file.
536                          */
537                         maybe_append_prop(plain_file, sccs_prop)->
538                           body.sccs.file = file;
539 try_pattern:
540                         MBSTOWCS(tmp_wcs_buffer, dp->d_name + 2);
541                         if ((pattern != NULL) &&
542                             amatch(tmp_wcs_buffer, pattern)) {
543                                 if (debug_level > 0) {
544                                         WCSTOMBS(mbs_buffer, pattern);
545                                         (void) printf(catgets(catd, 1, 233, "'%s
546                                                       line->body.line.target->
547                                                       string_mb,
548                                                       plain_file->string_mb,
549                                                       mbs_buffer);
550                                 }
```



```
551                                 enter_dependency(line, plain_file, false);
552                                 result++;
553                         }
554                 }
555         }
556         (void) closedir(dir_fd);

558         return result;
559 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  102066 Wed May 20 11:24:55 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef SUN5_0 (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 #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
  45 #       include <avo/cleanup.h>
  46 #endif
  47 #endif

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

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

  60 #ifdef DISTRIBUTED
  61 #       include <dm/Avo_AcknowledgeMsg.h>
```



```
  62 #       include <rw/xdrstrea.h>
  63 #       include <dmrc/dmrc.h> /* dmakerc file processing */
  64 #endif

  66 #include <locale.h>             /* setlocale() */
  67 #include <mk/defs.h>
  68 #include <mksdmsi18n/mksdmsi18n.h>      /* libmksdmsi18n_init() */
  69 #include <mksh/macro.h>         /* getvar() */
  70 #include <mksh/misc.h>          /* getmem(), setup_char_semantics() */

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

  78 #include <pwd.h>                /* getpwnam() */
  79 #include <setjmp.h>
  80 #include <signal.h>
  81 #include <stdlib.h>
  82 #include <sys/errno.h>          /* ENOENT */
  83 #include <sys/stat.h>           /* fstat() */
  84 #include <fcntl.h>              /* open() */

  86 #ifdef SUN5_0
  86 #       include <sys/systeminfo.h>      /* sysinfo() */
  88 #endif

  88 #include <sys/types.h>          /* stat() */
  89 #include <sys/wait.h>           /* wait() */
  90 #include <unistd.h>             /* execv(), unlink(), access() */
  91 #include <vroot/report.h>       /* report_dependency(), get_report_file() */

  93 // From read2.cc
  94 extern  Name            normalize_name(register wchar_t *name_string, register i

  96 // From parallel.cc
  97 #if defined(TEAMWARE_MAKE_CMN)
  98 #define MAXJOBS_ADJUST_RFE4694000

 100 #ifdef MAXJOBS_ADJUST_RFE4694000
 101 extern void job_adjust_fini();
 102 #endif /* MAXJOBS_ADJUST_RFE4694000 */
 103 #endif /* TEAMWARE_MAKE_CMN */

 105 #if defined(linux)
 106 #include <ctype.h>
 107 #endif

 109 /*
 110  * Defined macros
 111  */
 112 #define LD_SUPPORT_ENV_VAR      NOCATGETS("SGS_SUPPORT")
 113 #define LD_SUPPORT_MAKE_LIB     NOCATGETS("libmakestate.so.1")

 115 /*
 116  * typedefs & structs
 117  */

 119 /*
 120  * Static variables
 121  */
 122 static  char            *argv_zero_string;
 123 static  Boolean         build_failed_ever_seen;
 124 static  Boolean         continue_after_error_ever_seen; /* '-k' */
 125 static  Boolean         dmake_group_specified;          /* '-g' */
```

```
126 static  Boolean         dmake_max_jobs_specified;       /* '-j' */
127 static  Boolean         dmake_mode_specified;           /* '-m' */
128 static  Boolean         dmake_add_mode_specified;       /* '-x' */
129 static  Boolean         dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE
130 static  Boolean         dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M
131 static  Boolean         dmake_odir_specified;           /* '-o' */
132 static  Boolean         dmake_rcfile_specified;         /* '-c' */
133 static  Boolean         env_wins;                       /* '-e' */
134 static  Boolean         ignore_default_mk;              /* '-r' */
135 static  Boolean         list_all_targets;               /* '-T' */
136 static  int             mf_argc;
137 static  char            **mf_argv;
138 static  Dependency_rec  not_auto_depen_struct;
139 static  Dependency      not_auto_depen = &not_auto_depen_struct;
140 static  Boolean         pmake_cap_r_specified;          /* '-R' */
141 static  Boolean         pmake_machinesfile_specified;   /* '-M' */
142 static  Boolean         stop_after_error_ever_seen;     /* '-S' */
143 static  Boolean         trace_status;                   /* '-p' */

145 #ifdef DMAKE_STATISTICS
146 static  Boolean         getname_stat = false;
147 #endif

149 #if defined(TEAMWARE_MAKE_CMN)
150         static  time_t          start_time;
151         static  int             g_argc;
152         static  char            **g_argv;
153 #ifdef USE_DMS_CCR
154         static  Avo_usage_tracking *usageTracking = NULL;
155 #else
156         static  Avo_cleanup     *cleanup = NULL;
157 #endif
158 #endif

160 /*
161  * File table of contents
162  */
165 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
163         extern "C" void         cleanup_after_exit(void);
167 #else
168         extern  void            cleanup_after_exit(int, ...);
169 #endif

165 #ifdef TEAMWARE_MAKE_CMN
166 extern "C" {
167         extern  void            dmake_exit_callback(void);
168         extern  void            dmake_message_callback(char *);
169 }
170 #endif

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

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

176 static  void            append_makeflags_string(Name, String);
177 static  void            doalarm(int);
178 static  void            enter_argv_values(int , char **, ASCII_Dyn_Array *);
179 static  void            make_targets(int, char **, Boolean);
180 static  int             parse_command_option(char);
181 static  void            read_command_options(int, char **);
182 static  void            read_environment(Boolean);
183 static  void            read_files_and_state(int, char **);
184 static  Boolean         read_makefile(Name, Boolean, Boolean, Boolean);
185 static  void            report_recursion(Name);
186 static  void            set_sgs_support(void);
187 static  void            setup_for_projectdir(void);
```



```
188 static  void            setup_makeflags_argv(void);
189 static  void            report_dir_enter_leave(Boolean entering);

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

193 #ifdef DISTRIBUTED
194         extern  int             dmake_ofd;
195         extern  FILE*           dmake_ofp;
196         extern  int             rxmPid;
197         extern  XDR             xdrs_out;
198 #endif
199 #ifdef TEAMWARE_MAKE_CMN
200         extern  char            verstring[];
201 #endif

203 jmp_buf jmpbuffer;
204 #if !defined(linux)
205 extern nl_catd catd;
206 #endif

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

```
255 #ifndef PARALLEL
256         struct stat             out_stat, err_stat;
257 #endif
258         hostid = gethostid();
259 #ifdef TEAMWARE_MAKE_CMN
260         avo_get_user(NULL, NULL); // Initialize user name
261 #endif
262         bsd_signals();

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

266 #if defined (HP_UX) || defined(linux)
267        /* HP-UX users typically will not have NLSPATH set, and this binary
268         * requires that it be set.  On HP-UX 9.0x, /usr/lib/nls/%L/%N.cat is
269         * the path to set it to.
270         */

272        if (getenv(NOCATGETS("NLSPATH")) == NULL) {
273          putenv(NOCATGETS("NLSPATH=/usr/lib/nls/%L/%N.cat"));
274         }
275 #endif

277 #ifdef DMAKE_STATISTICS
278         if (getenv(NOCATGETS("DMAKE_STATISTICS"))) {
279                 getname_stat = true;
280         }
281 #endif



284         /*
285          * avo_init() sets the umask to 0.  Save it here and restore
286          * it after the avo_init() call.
287          */
288 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
289         um = umask(0);
290         avo_init(argv[0]);
291         umask(um);

293 #ifdef USE_DMS_CCR
294         usageTracking = new Avo_usage_tracking(NOCATGETS("dmake"), argc, argv);
295 #else
296         cleanup = new Avo_cleanup(NOCATGETS("dmake"), argc, argv);
297 #endif
298 #endif

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

304 #ifdef _CHECK_UPDATE_H
305         /* This is for dmake only (not for Solaris make).
306          * Check (in background) if there is an update (dmake patch)
307          * and inform user
308          */
309         {
310                 Avo_err         *err;
311                 char            *dir;
312                 err = avo_find_run_dir(&dir);
313                 if (AVO_OK == err) {
314                         AU_check_update_service(NOCATGETS("Dmake"), dir);
315                 }
316         }
317 #endif /* _CHECK_UPDATE_H */
318 #endif
```



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


323 #if defined(TEAMWARE_MAKE_CMN) || defined(MAKETOOL)
324 /*
325  * I put libmksdmsi18n_init() under #ifdef because it requires avo_i18n_init()
326  * from avo_util library.
327  */
328         libmksdmsi18n_init();
329 #ifdef USE_DMS_CCR
330         libpubdmsi18n_init();
331 #endif
332 #endif


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

339 #ifdef TEAMWARE_MAKE_CMN
340         g_argc = argc;
341         g_argv = (char **) malloc((g_argc + 1) * sizeof(char *));
342         for (i = 0; i < argc; i++) {
343                 g_argv[i] = argv[i];
344         }
345         g_argv[i] = NULL;
346 #endif /* TEAMWARE_MAKE_CMN */

348         /*
349          * Set argv_zero_string to some form of argv[0] for
350          * recursive MAKE builds.
351          */

353         if (*argv[0] == (int) slash_char) {
354                 /* argv[0] starts with a slash */
355                 argv_zero_string = strdup(argv[0]);
356         } else if (strchr(argv[0], (int) slash_char) == NULL) {
357                 /* argv[0] contains no slashes */
358                 argv_zero_string = strdup(argv[0]);
359         } else {
360                 /*
361                  * argv[0] contains at least one slash,
362                  * but doesn't start with a slash
363                  */
364                 char    *tmp_current_path;
365                 char    *tmp_string;

367                 tmp_current_path = get_current_path();
368                 tmp_string = getmem(strlen(tmp_current_path) + 1 +
369                                     strlen(argv[0]) + 1);
370                 (void) sprintf(tmp_string,
371                                "%s/%s",
372                                tmp_current_path,
373                                argv[0]);
374                 argv_zero_string = strdup(tmp_string);
375                 retmem_mb(tmp_string);
376         }

378         /*
379          * The following flags are reset if we don't have the
380          * (.nse_depinfo or .make.state) files locked and only set
381          * AFTER the file has been locked. This ensures that if the user
382          * interrupts the program while file_lock() is waiting to lock
383          * the file, the interrupt handler doesn't remove a lock
384          * that doesn't belong to us.
385          */
```

```
386         make_state_lockfile = NULL;
387         make_state_locked = false;

389 #ifdef NSE
390         nse_depinfo_lockfile[0] = '\0';
391         nse_depinfo_locked = false;
392 #endif

394         /*
395          * look for last slash char in the path to look at the binary
396          * name. This is to resolve the hard link and invoke make
397          * in svr4 mode.
398          */

400         /* Sun OS make standart */
401         svr4 = false;
402         posix = false;
403         if(!strcmp(argv_zero_string, NOCATGETS("/usr/xpg4/bin/make"))) {
404                 svr4 = false;
405                 posix = true;
406         } else {
407                 prognameptr = strrchr(argv[0], '/');
408                 if(prognameptr) {
409                         prognameptr++;
410                 } else {
411                         prognameptr = argv[0];
412                 }
413                 if(!strcmp(prognameptr, NOCATGETS("svr4.make"))) {
414                         svr4 = true;
415                         posix = false;
416                 }
417         }
418 #if !defined(HP_UX) && !defined(linux)
419         if (getenv(USE_SVR4_MAKE) || getenv(NOCATGETS("USE_SVID"))){
420            svr4 = true;
421            posix = false;
422         }
423 #endif

425         /*
426          * Find the dmake_compat_mode: posix, sun, svr4, or gnu_style, .
427          */
428         char * dmake_compat_mode_var = getenv(NOCATGETS("SUN_MAKE_COMPAT_MODE"))
429         if (dmake_compat_mode_var != NULL) {
430                 if (0 == strcasecmp(dmake_compat_mode_var, NOCATGETS("GNU"))) {
431                         gnu_style = true;
432                 }
433                 //svr4 = false;
434                 //posix = false;
435         }

437         /*
438          * Temporary directory set up.
439          */
440         char * tmpdir_var = getenv(NOCATGETS("TMPDIR"));
441         if (tmpdir_var != NULL && *tmpdir_var == '/' && strlen(tmpdir_var) < MAX
442                 strcpy(mbs_buffer, tmpdir_var);
443                 for (tmpdir_var = mbs_buffer+strlen(mbs_buffer);
444                         *(--tmpdir_var) == '/' && tmpdir_var > mbs_buffer;
445                         *tmpdir_var = '\0');
446                 if (strlen(mbs_buffer) + 32 < MAXPATHLEN) { /* 32 = strlen("/dma
447                         sprintf(mbs_buffer2, NOCATGETS("%s/dmake.tst.%d.XXXXXX")
448                                 mbs_buffer, getpid());
449                         int fd = mkstemp(mbs_buffer2);
450                         if (fd >= 0) {
451                                 close(fd);
```



```
452                                 unlink(mbs_buffer2);
453                                 tmpdir = strdup(mbs_buffer);
454                         }
455                 }
456         }

458 #ifndef PARALLEL
459         /* find out if stdout and stderr point to the same place */
460         if (fstat(1, &out_stat) < 0) {
461                 fatal(catgets(catd, 1, 165, "fstat of standard out failed: %s"),
462         }
463         if (fstat(2, &err_stat) < 0) {
464                 fatal(catgets(catd, 1, 166, "fstat of standard error failed: %s"
465         }
466         if ((out_stat.st_dev == err_stat.st_dev) &&
467             (out_stat.st_ino == err_stat.st_ino)) {
468                 stdout_stderr_same = true;
469         } else {
470                 stdout_stderr_same = false;
471         }
472 #else
473         stdout_stderr_same = false;
474 #endif
475         /* Make the vroot package scan the path using shell semantics */
476         set_path_style(0);

478         setup_char_semantics();

480         setup_for_projectdir();

482         /*
483          * If running with .KEEP_STATE, curdir will be set with
484          * the connected directory.
485          */
492 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
486         (void) atexit(cleanup_after_exit);
494 #else
495         (void) on_exit(cleanup_after_exit, (char *) NULL);
496 #endif

488         load_cached_names();

490 /*
491  *      Set command line flags
492  */
493         setup_makeflags_argv();
494         read_command_options(mf_argc, mf_argv);
495         read_command_options(argc, argv);
496         if (debug_level > 0) {
497                 cp = getenv(makeflags->string_mb);
498                 (void) printf(catgets(catd, 1, 167, "MAKEFLAGS value: %s\n"), cp
499         }

501         setup_interrupt(handle_interrupt);

503         read_files_and_state(argc, argv);

505 #ifdef TEAMWARE_MAKE_CMN
506         /*
507          * Find the dmake_output_mode: TXT1, TXT2 or HTML1.
508          */
509         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
510         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
511         prop2 = get_prop(dmake_name2->prop, macro_prop);
512         if (prop2 == NULL) {
513                 /* DMAKE_OUTPUT_MODE not defined, default to TXT1 mode */
```

```
514                 output_mode = txt1_mode;
515         } else {
516                 dmake_value2 = prop2->body.macro.value;
517                 if ((dmake_value2 == NULL) ||
518                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT1")))) {
519                         output_mode = txt1_mode;
520                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("TXT2")))
521                         output_mode = txt2_mode;
522                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("HTML1"))
523                         output_mode = html1_mode;
524                 } else {
525                         warning(catgets(catd, 1, 352, "Unsupported value ‘%s’ fo
526                               dmake_value2->string_mb);
527                 }
528         }
529         /*
530          * Find the dmake_mode: distributed, parallel, or serial.
531          */
532     if ((!pmake_cap_r_specified) &&
533         (!pmake_machinesfile_specified)) {
534         MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
535         dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
536         prop2 = get_prop(dmake_name2->prop, macro_prop);
537         if (prop2 == NULL) {
538                 /* DMAKE_MODE not defined, default to distributed mode */
539                 dmake_mode_type = distributed_mode;
540                 no_parallel = false;
541         } else {
542                 dmake_value2 = prop2->body.macro.value;
543                 if ((dmake_value2 == NULL) ||
544                     (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("distributed"))
545                         dmake_mode_type = distributed_mode;
546                         no_parallel = false;
547                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("parallel
548                         dmake_mode_type = parallel_mode;
549                         no_parallel = false;
550 #ifdef SGE_SUPPORT
551                         grid = false;
552                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("grid")))
553                         dmake_mode_type = parallel_mode;
554                         no_parallel = false;
555                         grid = true;
556 #endif
557                 } else if (IS_EQUAL(dmake_value2->string_mb, NOCATGETS("serial")
558                         dmake_mode_type = serial_mode;
559                         no_parallel = true;
560                 } else {
561                         fatal(catgets(catd, 1, 307, "Unknown dmake mode argument
562                 }
563         }

565         if ((!list_all_targets) &&
566             (report_dependencies_level == 0)) {
567                 /*
568                  * Check to see if either DMAKE_RCFILE or DMAKE_MODE is defined.
569                  * They could be defined in the env, in the makefile, or on the
570                  * command line.
571                  * If neither is defined, and $(HOME)/.dmakerc does not exists,
572                  * then print a message, and default to parallel mode.
573                  */
574 #ifdef DISTRIBUTED
575                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
576                 dmake_name = GETNAME(wcs_buffer, FIND_LENGTH);
577                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
578                 dmake_name2 = GETNAME(wcs_buffer, FIND_LENGTH);
579                 if ((((prop = get_prop(dmake_name->prop, macro_prop)) == NULL) |
```



```
580                     ((dmake_value = prop->body.macro.value) == NULL)) &&
581                    (((prop2 = get_prop(dmake_name2->prop, macro_prop)) == NULL)
582                     ((dmake_value2 = prop2->body.macro.value) == NULL))) {
583                        Boolean empty_dmakerc = true;
584                        char *homedir = getenv(NOCATGETS("HOME"));
585                        if ((homedir != NULL) && (strlen(homedir) < (sizeof(def_
586                                sprintf(def_dmakerc_path, NOCATGETS("%s/.dmakerc
587                                if ((((statval = stat(def_dmakerc_path, &statbuf
588                                        ((statval == 0) && (statbuf.st_size == 0
589                                } else {
590                                        Avo_dmakerc     *rcfile = new Avo_dmaker
591                                        Avo_err         *err = rcfile->read(def_
592                                        if (err) {
593                                                fatal(err->str);
594                                        }
595                                        empty_dmakerc = rcfile->was_empty();
596                                        delete rcfile;
597                                }
598                        }
599                        if (empty_dmakerc) {
600                                if (getenv(NOCATGETS("DMAKE_DEF_PRINTED")) == NU
601                                        putenv(NOCATGETS("DMAKE_DEF_PRINTED=TRUE
602                                        (void) fprintf(stdout, catgets(catd, 1,
603                                        (void) fprintf(stdout, catgets(catd, 1,
604                                }
605                                dmake_mode_type = parallel_mode;
606                                no_parallel = false;
607                        }
608                }
609 #else
610                 if(dmake_mode_type == distributed_mode) {
611                         (void) fprintf(stdout, NOCATGETS("dmake: Distributed mod
612                         (void) fprintf(stdout, NOCATGETS("       Defaulting to p
613                         dmake_mode_type = parallel_mode;
614                         no_parallel = false;
615                 }
616 #endif  /* DISTRIBUTED */
617         }
618     }
619 #endif

621 #ifdef TEAMWARE_MAKE_CMN
622         parallel_flag = true;
623         /* XXX - This is a major hack for DMake/Licensing. */
624         if (getenv(NOCATGETS("DMAKE_CHILD")) == NULL) {
625                 if (!avo_cli_search_license(argv[0], dmake_exit_callback, TRUE,
626                         /*
627                          * If the user can not get a TeamWare license,
628                          * default to serial mode.
629                          */
630                         dmake_mode_type = serial_mode;
631                         no_parallel = true;
632                 } else {
633                         putenv(NOCATGETS("DMAKE_CHILD=TRUE"));
634                 }
635                 start_time = time(NULL);
636                 /*
637                  * XXX - Hack to disable SIGALRM’s from licensing library’s
638                  *       setitimer().
639                  */
640                 value.it_interval.tv_sec = 0;
641                 value.it_interval.tv_usec = 0;
642                 value.it_value.tv_sec = 0;
643                 value.it_value.tv_usec = 0;
644                 (void) setitimer(ITIMER_REAL, &value, NULL);
645         }
```

```
647 //
648 // If dmake is running with -t option, set dmake_mode_type to serial.
649 // This is done because doname() calls touch_command() that runs serially.
650 // If we do not do that, maketool will have problems.
651 //
652         if(touch) {
653                 dmake_mode_type = serial_mode;
654                 no_parallel = true;
655         }
656 #else
657         parallel_flag = false;
658 #endif

660 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
661         /*
662          * Check whether stdout and stderr are physically same.
663          * This is in order to decide whether we need to redirect
664          * stderr separately from stdout.
665          * This check is performed only if __DMAKE_SEPARATE_STDERR
666          * is not set. This variable may be used in order to preserve
667          * the 'old' behaviour.
668          */
669         out_err_same = true;
670         char * dmake_sep_var = getenv(NOCATGETS("__DMAKE_SEPARATE_STDERR"));
671         if (dmake_sep_var == NULL || (0 != strcasecmp(dmake_sep_var, NOCATGETS("
672                 struct stat stdout_stat;
673                 struct stat stderr_stat;
674                 if( (fstat(1, &stdout_stat) == 0)
675                  && (fstat(2, &stderr_stat) == 0) )
676                 {
677                         if( (stdout_stat.st_dev != stderr_stat.st_dev)
678                          || (stdout_stat.st_ino != stderr_stat.st_ino) )
679                         {
680                                 out_err_same = false;
681                         }
682                 }
683         }
684 #endif

686 #ifdef DISTRIBUTED
687         /*
688          * At this point, DMake should startup an rxm with any and all
689          * DMake command line options. Rxm will, among other things,
690          * read the rc file.
691          */
692         if ((!list_all_targets) &&
693             (report_dependencies_level == 0) &&
694             (dmake_mode_type == distributed_mode)) {
695                 startup_rxm();
696         }
697 #endif
698
699 /*
700  *      Enable interrupt handler for alarms
701  */
702         (void) bsd_signal(SIGALRM, (SIG_PF)doalarm);

704 /*
705  *      Check if make should report
706  */
707         if (getenv(sunpro_dependencies->string_mb) != NULL) {
708                 FILE    *report_file;

710                 report_dependency("");
711                 report_file = get_report_file();
```



```
712                 if ((report_file != NULL) && (report_file != (FILE*)-1)) {
713                         (void) fprintf(report_file, "\n");
714                 }
715         }

717 /*
718  *      Make sure SUNPRO_DEPENDENCIES is exported (or not) properly
719  *      and NSE_DEP.
720  */
721         if (keep_state) {
722                 maybe_append_prop(sunpro_dependencies, macro_prop)->
723                   body.macro.exported = true;
724 #ifdef NSE
725                 (void) setenv(NOCATGETS("NSE_DEP"), get_current_path());
726 #endif
727         } else {
728                 maybe_append_prop(sunpro_dependencies, macro_prop)->
729                   body.macro.exported = false;
730         }

732         working_on_targets = true;
733         if (trace_status) {
734                 dump_make_state();
745 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
735                 fclose(stdout);
736                 fclose(stderr);
737                 exit_status = 0;
749 #endif
738                 exit(0);
739         }
740         if (list_all_targets) {
741                 dump_target_list();
754 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
742                 fclose(stdout);
743                 fclose(stderr);
744                 exit_status = 0;
758 #endif
745                 exit(0);
746         }
747         trace_reader = false;

749         /*
750          * Set temp_file_directory to the directory the .make.state
751          * file is written to.
752          */
753         if ((slash_ptr = strrchr(make_state->string_mb, (int) slash_char)) == NU
754                 temp_file_directory = strdup(get_current_path());
755         } else {
756                 *slash_ptr = (int) nul_char;
757                 (void) strcpy(make_state_dir, make_state->string_mb);
758                 *slash_ptr = (int) slash_char;
759                    /* when there is only one slash and it's the first
760                    ** character, make_state_dir should point to '/'.
761                    */
762                 if(make_state_dir[0] == '\0') {
763                    make_state_dir[0] = '/';
764                    make_state_dir[1] = '\0';
765                 }
766                 if (make_state_dir[0] == (int) slash_char) {
767                         temp_file_directory = strdup(make_state_dir);
768                 } else {
769                         char    tmp_current_path2[MAXPATHLEN];
770
771                         (void) sprintf(tmp_current_path2,
772                                        "%s/%s",
773                                        get_current_path(),
```

```
774                                     make_state_dir);
775                         temp_file_directory = strdup(tmp_current_path2);
776                 }
777         }

779 #ifdef DISTRIBUTED
780         building_serial = false;
781 #endif

783         report_dir_enter_leave(true);

785         make_targets(argc, argv, parallel_flag);

787         report_dir_enter_leave(false);

789 #ifdef NSE
790         exit(nse_exit_status());
791 #else
792         if (build_failed_ever_seen) {
807 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
793                 if (posix) {
794                         exit_status = 1;
795                 }
811 #endif
796                 exit(1);
797         }
814 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
798         exit_status = 0;
816 #endif
799         exit(0);
800 #endif
801         /* NOTREACHED */
802 }

804 /*
805  *      cleanup_after_exit()
806  *
807  *      Called from exit(), performs cleanup actions.
808  *
809  *      Parameters:
810  *              status          The argument exit() was called with
811  *              arg             Address of an argument vector to
812  *                              cleanup_after_exit()
813  *
814  *      Global variables used:
815  *              command_changed Set if we think .make.state should be rewritten
816  *              current_line    Is set we set commands_changed
817  *              do_not_exec_rule
818  *                              True if -n flag on
819  *              done            The Name ".DONE", rule we run
820  *              keep_state      Set if .KEEP_STATE seen
821  *              parallel        True if building in parallel
822  *              quest           If -q is on we do not run .DONE
823  *              report_dependencies
824  *                              True if -P flag on
825  *              running_list    List of parallel running processes
826  *              temp_file_name  The temp file is removed, if any
827  *              catd    the message catalog file
828  *              usage_tracking  Should have been constructed in main()
829  *                              should destroyed just before exiting
830  */
849 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
831 extern "C" void
832 cleanup_after_exit(void)
852 #else
853 void cleanup_after_exit(int status, ...)
```



```
854 #endif
833 {
834         Running         rp;
835 #ifdef NSE
836         char            push_cmd[NSE_TFS_PUSH_LEN + 3 +
837                                 (MAXPATHLEN * MB_LEN_MAX) + 12];
838         char            *active;
839 #endif

841 extern long     getname_bytes_count;
842 extern long     getname_names_count;
843 extern long     getname_struct_count;
844 extern long     freename_bytes_count;
845 extern long     freename_names_count;
846 extern long     freename_struct_count;
847 extern long     other_alloc;

849 extern long     env_alloc_num;
850 extern long     env_alloc_bytes;


853 #ifdef DMAKE_STATISTICS
854 if(getname_stat) {
855         printf(NOCATGETS(">>> Getname statistics:\n"));
856         printf(NOCATGETS("  Allocated:\n"));
857         printf(NOCATGETS("        Names: %ld\n"), getname_names_count);
858         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), getname_bytes_c
859         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), getname_struct_
860         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), getname_struct_

862         printf(NOCATGETS("\n  Unallocated: %ld\n"), freename_names_count);
863         printf(NOCATGETS("        Names: %ld\n"), freename_names_count);
864         printf(NOCATGETS("      Strings: %ld Kb (%ld bytes)\n"), freename_bytes_
865         printf(NOCATGETS("      Structs: %ld Kb (%ld bytes)\n"), freename_struct
866         printf(NOCATGETS("  Total bytes: %ld Kb (%ld bytes)\n"), freename_struct

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

870         printf(NOCATGETS("\n>>> Other:\n"));
871         printf(
872                 NOCATGETS("       Env (%ld): %ld Kb (%ld bytes)\n"),
873                 env_alloc_num,
874                 env_alloc_bytes/1000,
875                 env_alloc_bytes
876         );

878 }
879 #endif

881 /*
882 #ifdef DISTRIBUTED
883     if (get_parent() == TRUE) {
884 #endif
885  */

887         parallel = false;
910 #ifdef SUN5_0
888         /* If we used the SVR4_MAKE, don't build .DONE or .FAILED */
889         if (!getenv(USE_SVR4_MAKE)){
913 #endif
890             /* Build the target .DONE or .FAILED if we caught an error */
891             if (!quest && !list_all_targets) {
892                 Name            failed_name;

894                 MBSTOWCS(wcs_buffer, NOCATGETS(".FAILED"));
895                 failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
```

```
#if defined(SUN5_0) || defined(HP_UX) || defined(linux)
                if ((exit_status != 0) && (failed_name->prop != NULL)) {
#else
                if ((status != 0) && (failed_name->prop != NULL)) {
#endif
#ifdef TEAMWARE_MAKE_CMN
                        /*
                         * [tolik] switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
#endif
                        (void) doname(failed_name, false, true);
                } else {
                    if (!trace_status) {
#ifdef TEAMWARE_MAKE_CMN
                        /*
                         * Switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
#endif
                        (void) doname(done, false, true);
                    }
                }
            }
#ifdef SUN5_0
        }
#endif
        /*
         * Remove the temp file utilities report dependencies thru if it
         * is still around
         */
        if (temp_file_name != NULL) {
                (void) unlink(temp_file_name->string_mb);
        }
        /*
         * Do not save the current command in .make.state if make
         * was interrupted.
         */
        if (current_line != NULL) {
                command_changed = true;
                current_line->body.line.command_used = NULL;
        }
        /*
         * For each parallel build process running, remove the temp files
         * and zap the command line so it won't be put in .make.state
         */
        for (rp = running_list; rp != NULL; rp = rp->next) {
                if (rp->temp_file != NULL) {
                        (void) unlink(rp->temp_file->string_mb);
                }
                if (rp->stdout_file != NULL) {
                        (void) unlink(rp->stdout_file);
                        retmem_mb(rp->stdout_file);
                        rp->stdout_file = NULL;
                }
                if (rp->stderr_file != NULL) {
                        (void) unlink(rp->stderr_file);
                        retmem_mb(rp->stderr_file);
                        rp->stderr_file = NULL;
                }
                command_changed = true;
/*
                line = get_prop(rp->target->prop, line_prop);
                if (line != NULL) {
```



```
                        line->body.line.command_used = NULL;
                }
 */
        }
        /* Remove the statefile lock file if the file has been locked */
        if ((make_state_lockfile != NULL) && (make_state_locked)) {
                (void) unlink(make_state_lockfile);
                make_state_lockfile = NULL;
                make_state_locked = false;
        }
        /* Write .make.state */
        write_state_file(1, (Boolean) 1);

#ifdef TEAMWARE_MAKE_CMN
        // Deleting the usage tracking object sends the usage mail
#ifdef USE_DMS_CCR
        //usageTracking->setExitStatus(exit_status, NULL);
        //delete usageTracking;
#else
        cleanup->set_exit_status(exit_status);
        delete cleanup;
#endif
#endif

#ifdef NSE
        /* If running inside an activated environment, push the */
        /* .nse_depinfo file (if written) */
        active = getenv(NSE_VARIANT_ENV);
        if (keep_state &&
            (active != NULL) &&
            !IS_EQUAL(active, NSE_RT_SOURCE_NAME) &&
            !do_not_exec_rule &&
            (report_dependencies_level == 0)) {
                (void) sprintf(push_cmd,
                                "%s %s/%s",
                                NSE_TFS_PUSH,
                                get_current_path(),
                                NSE_DEPINFO);
                (void) system(push_cmd);
        }
#endif

/*
#ifdef DISTRIBUTED
    }
#endif
 */

#if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
        job_adjust_fini();
#endif

#ifdef TEAMWARE_MAKE_CMN
        catclose(catd);
#endif
#ifdef DISTRIBUTED
        if (rxmPid > 0) {
                // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
                Avo_AcknowledgeMsg acknowledgeMsg;
                RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

                int xdrResult = xdr(&xdrs_out, msg);

                if (xdrResult) {
                        fflush(dmake_ofp);
                } else {
```

```
1022 /*
1023                         fatal(catgets(catd, 1, 266, "couldn't tell rxm to exit")
1024  */
1025                         kill(rxmPid, SIGTERM);
1026                 }

1028                 waitpid(rxmPid, NULL, 0);
1029                 rxmPid = 0;
1030         }
1031 #endif
1032 }

1034 /*
1035  *      handle_interrupt()
1036  *
1037  *      This is where C-C traps are caught.
1038  *
1039  *      Parameters:
1040  *
1041  *      Global variables used (except DMake 1.0):
1042  *              current_target          Sometimes the current target is removed
1043  *              do_not_exec_rule        But not if -n is on
1044  *              quest                   or -q
1045  *              running_list            List of parallel running processes
1046  *              touch                   Current target is not removed if -t on
1047  */
1048 void
1049 handle_interrupt(int)
1050 {
1051         Property                member;
1052         Running                 rp;

1054         (void) fflush(stdout);
1055 #ifdef DISTRIBUTED
1056         if (rxmPid > 0) {
1057                 // Tell rxm to exit by sending it an Avo_AcknowledgeMsg
1058                 Avo_AcknowledgeMsg acknowledgeMsg;
1059                 RWCollectable *msg = (RWCollectable *)&acknowledgeMsg;

1061                 int xdrResult = xdr(&xdrs_out, msg);

1063                 if (xdrResult) {
1064                         fflush(dmake_ofp);
1065                 } else {
1066                         kill(rxmPid, SIGTERM);
1067                         rxmPid = 0;
1068                 }
1069         }
1070 #endif
1071         if (childPid > 0) {
1072                 kill(childPid, SIGTERM);
1073                 childPid = -1;
1074         }
1075         for (rp = running_list; rp != NULL; rp = rp->next) {
1076                 if (rp->state != build_running) {
1077                         continue;
1078                 }
1079                 if (rp->pid > 0) {
1080                         kill(rp->pid, SIGTERM);
1081                         rp->pid = -1;
1082                 }
1083         }
1084         if (getpid() == getpgrp()) {
1085                 bsd_signal(SIGTERM, SIG_IGN);
1086                 kill (-getpid(), SIGTERM);
1087         }
```



```
1088 #ifdef TEAMWARE_MAKE_CMN
1089         /* Clean up all parallel/distributed children already finished */
1090         finish_children(false);
1091 #endif

1093         /* Make sure the processes running under us terminate first */

1125 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
1095         while (wait((int *) NULL) != -1);
1127 #else
1128         while (wait((union wait*) NULL) != -1);
1129 #endif
1096         /* Delete the current targets unless they are precious */
1097         if ((current_target != NULL) &&
1098             current_target->is_member &&
1099             ((member = get_prop(current_target->prop, member_prop)) != NULL)) {
1100                 current_target = member->body.member.library;
1101         }
1102         if (!do_not_exec_rule &&
1103             !touch &&
1104             !quest &&
1105             (current_target != NULL) &&
1106             !(current_target->stat.is_precious || all_precious)) {

1108 /* BID_1030811 */
1109 /* azv 16 Oct 95 */
1110                 current_target->stat.time = file_no_time;

1112                 if (exists(current_target) != file_doesnt_exist) {
1113                         (void) fprintf(stderr,
1114                                        "\n*** %s ",
1115                                        current_target->string_mb);
1116                         if (current_target->stat.is_dir) {
1117                                 (void) fprintf(stderr,
1118                                                catgets(catd, 1, 168, "not remove
1119                                                current_target->string_mb);
1120                         } else if (unlink(current_target->string_mb) == 0) {
1121                                 (void) fprintf(stderr,
1122                                                catgets(catd, 1, 169, "removed.\n
1123                                                current_target->string_mb);
1124                         } else {
1125                                 (void) fprintf(stderr,
1126                                                catgets(catd, 1, 170, "could not
1127                                                current_target->string_mb,
1128                                                errmsg(errno));
1129                         }
1130                 }
1131         }
1132         for (rp = running_list; rp != NULL; rp = rp->next) {
1133                 if (rp->state != build_running) {
1134                         continue;
1135                 }
1136                 if (rp->target->is_member &&
1137                     ((member = get_prop(rp->target->prop, member_prop)) !=
1138                      NULL)) {
1139                         rp->target = member->body.member.library;
1140                 }
1141                 if (!do_not_exec_rule &&
1142                     !touch &&
1143                     !quest &&
1144                     !(rp->target->stat.is_precious || all_precious)) {

1146                         rp->target->stat.time = file_no_time;
1147                         if (exists(rp->target) != file_doesnt_exist) {
1148                                 (void) fprintf(stderr,
1149                                                "\n*** %s ",
```

```
1150                                        rp->target->string_mb);
1151                                if (rp->target->stat.is_dir) {
1152                                        (void) fprintf(stderr,
1153                                                catgets(catd, 1, 171, "no
1154                                                rp->target->string_mb);
1155                                } else if (unlink(rp->target->string_mb) == 0) {
1156                                        (void) fprintf(stderr,
1157                                                catgets(catd, 1, 172, "re
1158                                                rp->target->string_mb);
1159                                } else {
1160                                        (void) fprintf(stderr,
1161                                                catgets(catd, 1, 173, "co
1162                                                rp->target->string_mb,
1163                                                errmsg(errno));
1164                                }
1165                        }
1166                }
1167        }

1169 #ifdef SGE_SUPPORT
1170        /* Remove SGE script file */
1171        if (grid) {
1172                unlink(script_file);
1173        }
1174 #endif

1176        /* Have we locked .make.state or .nse_depinfo? */
1177        if ((make_state_lockfile != NULL) && (make_state_locked)) {
1178                unlink(make_state_lockfile);
1179                make_state_lockfile = NULL;
1180                make_state_locked = false;
1181        }
1182 #ifdef NSE
1183        if ((nse_depinfo_lockfile[0] != '\0') && (nse_depinfo_locked)) {
1184                unlink(nse_depinfo_lockfile);
1185                nse_depinfo_lockfile[0] = '\0';
1186                nse_depinfo_locked = false;
1187        }
1188 #endif
1189        /*
1190         * Re-read .make.state file (it might be changed by recursive make)
1191         */
1192        check_state(NULL);

1194        report_dir_enter_leave(false);

1230 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
1196        exit_status = 2;
1232 #endif
1197        exit(2);
1198 }
_____unchanged_portion_omitted_

1656 /*
1657  *      parse_command_option(ch)
1658  *
1659  *      Parse make command line options.
1660  *
1661  *      Return value:
1662  *                              Indicates if any -f -c or -M were seen
1663  *
1664  *      Parameters:
1665  *              ch              The character to parse
1666  *
1667  *      Static variables used:
1668  *              dmake_group_specified   Set for make -g
```



```
1669  *              dmake_max_jobs_specified        Set for make -j
1670  *              dmake_mode_specified    Set for make -m
1671  *              dmake_add_mode_specified        Set for make -x
1672  *              dmake_compat_mode_specified     Set for make -x SUN_MAKE_COMPAT_
1673  *              dmake_output_mode_specified     Set for make -x DMAKE_OUTPUT_MOD
1674  *              dmake_odir_specified    Set for make -o
1675  *              dmake_rcfile_specified  Set for make -c
1676  *              env_wins                Set for make -e
1677  *              ignore_default_mk       Set for make -r
1678  *              trace_status            Set for make -p
1679  *
1680  *      Global variables used:
1681  *              .make.state path & name set for make -K
1682  *              continue_after_error    Set for make -k
1683  *              debug_level             Set for make -d
1684  *              do_not_exec_rule        Set for make -n
1685  *              filter_stderr           Set for make -X
1686  *              ignore_errors_all       Set for make -i
1687  *              no_parallel             Set for make -R
1688  *              quest                   Set for make -q
1689  *              read_trace_level        Set for make -D
1690  *              report_dependencies     Set for make -P
1691  *              send_mtool_msgs         Set for make -K
1692  *              silent_all              Set for make -s
1693  *              touch                   Set for make -t
1694  */
1695 static int
1696 parse_command_option(register char ch)
1697 {
1698        static int              invert_next = 0;
1699        int                     invert_this = invert_next;

1701        invert_next = 0;
1702        switch (ch) {
1703        case '-':                       /* Ignore "--" */
1704                return 0;
1705        case '~':                       /* Invert next option */
1706                invert_next = 1;
1707                return 0;
1708        case 'B':                       /* Obsolete */
1709                return 0;
1710        case 'b':                       /* Obsolete */
1711                return 0;
1712        case 'c':                       /* Read alternative dmakerc file */
1713                if (invert_this) {
1714                        dmake_rcfile_specified = false;
1715                } else {
1716                        dmake_rcfile_specified = true;
1717                }
1718                return 2;
1719        case 'D':                       /* Show lines read */
1720                if (invert_this) {
1721                        read_trace_level--;
1722                } else {
1723                        read_trace_level++;
1724                }
1725                return 0;
1726        case 'd':                       /* Debug flag */
1727                if (invert_this) {
1728                        debug_level--;
1729                } else {
1730 #if defined( HP_UX) || defined(linux)
1731                        if (debug_level < 2)  /* Fixes a bug on HP-UX */
1732 #endif
1733                                debug_level++;
1734                }
```

```
1735                 return 0;
1736 #ifdef NSE
1737         case 'E':
1738                 if (invert_this) {
1739                         nse = false;
1740                 } else {
1741                         nse = true;
1742                 }
1743                 nse_init_source_suffixes();
1744                 return 0;
1745 #endif
1746         case 'e':                       /* Environment override flag */
1747                 if (invert_this) {
1748                         env_wins = false;
1749                 } else {
1750                         env_wins = true;
1751                 }
1752                 return 0;
1753         case 'f':                       /* Read alternative makefile(s) */
1754                 return 1;
1755         case 'g':                       /* Use alternative DMake group */
1756                 if (invert_this) {
1757                         dmake_group_specified = false;
1758                 } else {
1759                         dmake_group_specified = true;
1760                 }
1761                 return 4;
1762         case 'i':                       /* Ignore errors */
1763                 if (invert_this) {
1764                         ignore_errors_all = false;
1765                 } else {
1766                         ignore_errors_all = true;
1767                 }
1768                 return 0;
1769         case 'j':                       /* Use alternative DMake max jobs */
1770                 if (invert_this) {
1771                         dmake_max_jobs_specified = false;
1772                 } else {
1773                         dmake_max_jobs_specified = true;
1774                 }
1775                 return 8;
1776         case 'K':                       /* Read alternative .make.state */
1777                 return 256;
1778         case 'k':                       /* Keep making even after errors */
1779                 if (invert_this) {
1780                         continue_after_error = false;
1781                 } else {
1782                         continue_after_error = true;
1783                         continue_after_error_ever_seen = true;
1784                 }
1785                 return 0;
1786         case 'M':                       /* Read alternative make.machines file
1787                 if (invert_this) {
1788                         pmake_machinesfile_specified = false;
1789                 } else {
1790                         pmake_machinesfile_specified = true;
1791                         dmake_mode_type = parallel_mode;
1792                         no_parallel = false;
1793                 }
1794                 return 16;
1795         case 'm':                       /* Use alternative DMake build mode */
1796                 if (invert_this) {
1797                         dmake_mode_specified = false;
1798                 } else {
1799                         dmake_mode_specified = true;
1800                 }
```



```
1801                 return 32;
1802         case 'x':                       /* Use alternative DMake mode */
1803                 if (invert_this) {
1804                         dmake_add_mode_specified = false;
1805                 } else {
1806                         dmake_add_mode_specified = true;
1807                 }
1808                 return 1024;
1809         case 'N':                       /* Reverse -n */
1810                 if (invert_this) {
1811                         do_not_exec_rule = true;
1812                 } else {
1813                         do_not_exec_rule = false;
1814                 }
1815                 return 0;
1816         case 'n':                       /* Print, not exec commands */
1817                 if (invert_this) {
1818                         do_not_exec_rule = false;
1819                 } else {
1820                         do_not_exec_rule = true;
1821                 }
1822                 return 0;
1823 #ifndef PARALLEL
1824         case 'O':                       /* Send job start & result msgs */
1825                 if (invert_this) {
1826                         send_mtool_msgs = false;
1827                 } else {
1828 #ifdef DISTRIBUTED
1829                         send_mtool_msgs = true;
1830 #endif
1831                 }
1832                 return 128;
1833 #endif
1834         case 'o':                       /* Use alternative dmake output dir */
1835                 if (invert_this) {
1836                         dmake_odir_specified = false;
1837                 } else {
1838                         dmake_odir_specified = true;
1839                 }
1840                 return 512;
1841         case 'P':                       /* Print for selected targets */
1842                 if (invert_this) {
1843                         report_dependencies_level--;
1844                 } else {
1845                         report_dependencies_level++;
1846                 }
1847                 return 0;
1848         case 'p':                       /* Print description */
1849                 if (invert_this) {
1850                         trace_status = false;
1851                         do_not_exec_rule = false;
1852                 } else {
1853                         trace_status = true;
1854                         do_not_exec_rule = true;
1855                 }
1856                 return 0;
1857         case 'q':                       /* Question flag */
1858                 if (invert_this) {
1859                         quest = false;
1860                 } else {
1861                         quest = true;
1862                 }
1863                 return 0;
1864         case 'R':                       /* Don't run in parallel */
1865 #ifdef TEAMWARE_MAKE_CMN
1866                 if (invert_this) {
```

```
1867                            pmake_cap_r_specified = false;
1868                            no_parallel = false;
1869                    } else {
1870                            pmake_cap_r_specified = true;
1871                            dmake_mode_type = serial_mode;
1872                            no_parallel = true;
1873                    }
1874 #else
1875                    warning(catgets(catd, 1, 182, "Ignoring ParallelMake -R option")
1876 #endif
1877                    return 0;
1878            case 'r':                       /* Turn off internal rules */
1879                    if (invert_this) {
1880                            ignore_default_mk = false;
1881                    } else {
1882                            ignore_default_mk = true;
1883                    }
1884                    return 0;
1885            case 'S':                       /* Reverse -k */
1886                    if (invert_this) {
1887                            continue_after_error = true;
1888                    } else {
1889                            continue_after_error = false;
1890                            stop_after_error_ever_seen = true;
1891                    }
1892                    return 0;
1893            case 's':                       /* Silent flag */
1894                    if (invert_this) {
1895                            silent_all = false;
1896                    } else {
1897                            silent_all = true;
1898                    }
1899                    return 0;
1900            case 'T':                       /* Print target list */
1901                    if (invert_this) {
1902                            list_all_targets = false;
1903                            do_not_exec_rule = false;
1904                    } else {
1905                            list_all_targets = true;
1906                            do_not_exec_rule = true;
1907                    }
1908                    return 0;
1909            case 't':                       /* Touch flag */
1910                    if (invert_this) {
1911                            touch = false;
1912                    } else {
1913                            touch = true;
1914                    }
1915                    return 0;
1916            case 'u':                       /* Unconditional flag */
1917                    if (invert_this) {
1918                            build_unconditional = false;
1919                    } else {
1920                            build_unconditional = true;
1921                    }
1922                    return 0;
1923            case 'V':                      /* SVR4 mode */
1924                    svr4 = true;
1925                    return 0;
1926            case 'v':                      /* Version flag */
1927                    if (invert_this) {
1928                    } else {
1929 #ifdef TEAMWARE_MAKE_CMN
1930                            fprintf(stdout, NOCATGETS("dmake: %s\n"), verstring);
1967 #ifdef SUN5_0
1931                            exit_status = 0;
```



```
1969 #endif
1932                            exit(0);
1933 #else
1934                            warning(catgets(catd, 1, 324, "Ignoring DistributedMake
1935 #endif
1936                    }
1937                    return 0;
1938            case 'w':                        /* Unconditional flag */
1939                    if (invert_this) {
1940                            report_cwd = false;
1941                    } else {
1942                            report_cwd = true;
1943                    }
1944                    return 0;
1945 #if 0
1946            case 'X':                       /* Filter stdout */
1947                    if (invert_this) {
1948                            filter_stderr = false;
1949                    } else {
1950                            filter_stderr = true;
1951                    }
1952                    return 0;
1953 #endif
1954            default:
1955                    break;
1956            }
1957            return 0;
1958 }
______unchanged_portion_omitted_

2071 /*
2072  *      read_files_and_state(argc, argv)
2073  *
2074  *      Read the makefiles we care about and the environment
2075  *      Also read the = style command line options
2076  *
2077  *      Parameters:
2078  *              argc            You know what this is
2079  *              argv            You know what this is
2080  *
2081  *      Static variables used:
2082  *              env_wins        make -e, determines if env vars are RO
2083  *              ignore_default_mk make -r, determines if make.rules is read
2084  *              not_auto_depen  dwight
2085  *
2086  *      Global variables used:
2087  *              default_target_to_build Set to first proper target from file
2088  *              do_not_exec_rule Set to false when makfile is made
2089  *              dot             The Name ".", used to read current dir
2090  *              empty_name      The Name "", use as macro value
2091  *              keep_state      Set if KEEP_STATE is in environment
2092  *              make_state      The Name ".make.state", used to read file
2093  *              makefile_type   Set to type of file being read
2094  *              makeflags       The Name "MAKEFLAGS", used to set macro value
2095  *              not_auto        dwight
2096  *              nse             Set if NSE_ENV is in the environment
2097  *              read_trace_level Checked to se if the reader should trace
2098  *              report_dependencies If -P is on we do not read .make.state
2099  *              trace_reader    Set if reader should trace
2100  *              virtual_root    The Name "VIRTUAL_ROOT", used to check value
2101  */
2102 static void
2103 read_files_and_state(int argc, char **argv)
2104 {
2105         wchar_t                 buffer[1000];
2106         wchar_t                 buffer_posix[1000];
```

```
2107         register char           ch;
2108         register char           *cp;
2109         Property                def_make_macro = NULL;
2110         Name                    def_make_name;
2111         Name                    default_makefile;
2112         String_rec              dest;
2113         wchar_t                 destbuffer[STRING_BUFFER_LENGTH];
2114         register int            i;
2115         register int            j;
2116         Name                    keep_state_name;
2117         int                     length;
2118         Name                    Makefile;
2119         register Property       macro;
2120         struct stat             make_state_stat;
2121         Name                    makefile_name;
2122         register int            makefile_next = 0;
2123         register Boolean        makefile_read = false;
2124         String_rec              makeflags_string;
2125         String_rec              makeflags_string_posix;
2126         String_rec *            makeflags_string_current;
2127         Name                    makeflags_value_saved;
2128         register Name           name;
2129         Name                    new_make_value;
2130         Boolean                 save_do_not_exec_rule;
2131         Name                    sdotMakefile;
2132         Name                    sdotmakefile_name;
2133         static wchar_t          state_file_str;
2134         static char             state_file_str_mb[MAXPATHLEN];
2135         static struct _Name     state_filename;
2136         Boolean                 temp;
2137         char                    tmp_char;
2138         wchar_t                 *tmp_wcs_buffer;
2139         register Name           value;
2140         ASCII_Dyn_Array         makeflags_and_macro;
2141         Boolean                 is_xpg4;

2143 /*
2144  *      Remember current mode. It may be changed after reading makefile
2145  *      and we will have to correct MAKEFLAGS variable.
2146  */
2147         is_xpg4 = posix;

2149         MBSTOWCS(wcs_buffer, NOCATGETS("KEEP_STATE"));
2150         keep_state_name = GETNAME(wcs_buffer, FIND_LENGTH);
2151         MBSTOWCS(wcs_buffer, NOCATGETS("Makefile"));
2152         Makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2153         MBSTOWCS(wcs_buffer, NOCATGETS("makefile"));
2154         makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2155         MBSTOWCS(wcs_buffer, NOCATGETS("s.makefile"));
2156         sdotmakefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
2157         MBSTOWCS(wcs_buffer, NOCATGETS("s.Makefile"));
2158         sdotMakefile = GETNAME(wcs_buffer, FIND_LENGTH);

2160 /*
2161  *      Set flag if NSE is active
2162  */
2163 #ifdef NSE
2164         if (getenv(NOCATGETS("NSE_ENV")) != NULL) {
2165                 nse = true;
2166         }
2167 #endif

2169 /*
2170  *      initialize global dependency entry for .NOT_AUTO
2171  */
2172         not_auto_depen->next = NULL;
```



```
2173         not_auto_depen->name = not_auto;
2174         not_auto_depen->automatic = not_auto_depen->stale = false;

2176 /*
2177  *      Read internal definitions and rules.
2178  */
2179         if (read_trace_level > 1) {
2180                 trace_reader = true;
2181         }
2182         if (!ignore_default_mk) {
2221 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
2183                 if (svr4) {
2184                         MBSTOWCS(wcs_buffer, NOCATGETS("svr4.make.rules"));
2185                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2186                 } else {
2187                         MBSTOWCS(wcs_buffer, NOCATGETS("make.rules"));
2188                         default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2189                 }
2229 #else
2230                 MBSTOWCS(wcs_buffer, NOCATGETS("default.mk"));
2231                 default_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2232 #endif
2190                 default_makefile->stat.is_file = true;

2192                 (void) read_makefile(default_makefile,
2193                                      true,
2194                                      false,
2195                                      true);
2196         }

2198         /*
2199          * If the user did not redefine the MAKE macro in the
2200          * default makefile (make.rules), then we'd like to
2201          * change the macro value of MAKE to be some form
2202          * of argv[0] for recursive MAKE builds.
2203          */
2204         MBSTOWCS(wcs_buffer, NOCATGETS("MAKE"));
2205         def_make_name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2206         def_make_macro = get_prop(def_make_name->prop, macro_prop);
2207         if ((def_make_macro != NULL) &&
2208             (IS_EQUAL(def_make_macro->body.macro.value->string_mb,
2209                       NOCATGETS("make")))) {
2210                 MBSTOWCS(wcs_buffer, argv_zero_string);
2211                 new_make_value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2212                 (void) SETVAR(def_make_name,
2213                               new_make_value,
2214                               false);
2215         }

2217         default_target_to_build = NULL;
2218         trace_reader = false;

2220 /*
2221  *      Read environment args. Let file args which follow override unless
2222  *      -e option seen. If -e option is not mentioned.
2223  */
2224         read_environment(env_wins);
2225         if (getvar(virtual_root)->hash.length == 0) {
2226                 maybe_append_prop(virtual_root, macro_prop)
2227                   ->body.macro.exported = true;
2228                 MBSTOWCS(wcs_buffer, "/");
2229                 (void) SETVAR(virtual_root,
2230                               GETNAME(wcs_buffer, FIND_LENGTH),
2231                               false);
2232         }
```

```
2234 /*
2235  * We now scan mf_argv and argv to see if we need to set
2236  * any of the DMake-added options/variables in MAKEFLAGS.
2237  */

2239         makeflags_and_macro.start = 0;
2240         makeflags_and_macro.size = 0;
2241         enter_argv_values(mf_argc, mf_argv, &makeflags_and_macro);
2242         enter_argv_values(argc, argv, &makeflags_and_macro);

2244 /*
2245  *      Set MFLAGS and MAKEFLAGS
2246  *
2247  *      Before reading makefile we do not know exactly which mode
2248  *      (posix or not) is used. So prepare two MAKEFLAGS strings
2249  *      for both posix and solaris modes because they are different.
2250  */
2251         INIT_STRING_FROM_STACK(makeflags_string, buffer);
2252         INIT_STRING_FROM_STACK(makeflags_string_posix, buffer_posix);
2253         append_char((int) hyphen_char, &makeflags_string);
2254         append_char((int) hyphen_char, &makeflags_string_posix);

2256         switch (read_trace_level) {
2257         case 2:
2258                 append_char('D', &makeflags_string);
2259                 append_char('D', &makeflags_string_posix);
2260         case 1:
2261                 append_char('D', &makeflags_string);
2262                 append_char('D', &makeflags_string_posix);
2263         }
2264         switch (debug_level) {
2265         case 2:
2266                 append_char('d', &makeflags_string);
2267                 append_char('d', &makeflags_string_posix);
2268         case 1:
2269                 append_char('d', &makeflags_string);
2270                 append_char('d', &makeflags_string_posix);
2271         }
2272 #ifdef NSE
2273         if (nse) {
2274                 append_char('E', &makeflags_string);
2275         }
2276 #endif
2277         if (env_wins) {
2278                 append_char('e', &makeflags_string);
2279                 append_char('e', &makeflags_string_posix);
2280         }
2281         if (ignore_errors_all) {
2282                 append_char('i', &makeflags_string);
2283                 append_char('i', &makeflags_string_posix);
2284         }
2285         if (continue_after_error) {
2286                 if (stop_after_error_ever_seen) {
2287                         append_char('S', &makeflags_string_posix);
2288                         append_char((int) space_char, &makeflags_string_posix);
2289                         append_char((int) hyphen_char, &makeflags_string_posix);
2290                 }
2291                 append_char('k', &makeflags_string);
2292                 append_char('k', &makeflags_string_posix);
2293         } else {
2294                 if (stop_after_error_ever_seen
2295                     && continue_after_error_ever_seen) {
2296                         append_char('k', &makeflags_string_posix);
2297                         append_char((int) space_char, &makeflags_string_posix);
2298                         append_char((int) hyphen_char, &makeflags_string_posix);
2299                         append_char('S', &makeflags_string_posix);
```



```
2300                 }
2301         }
2302         if (do_not_exec_rule) {
2303                 append_char('n', &makeflags_string);
2304                 append_char('n', &makeflags_string_posix);
2305         }
2306         switch (report_dependencies_level) {
2307         case 4:
2308                 append_char('P', &makeflags_string);
2309                 append_char('P', &makeflags_string_posix);
2310         case 3:
2311                 append_char('P', &makeflags_string);
2312                 append_char('P', &makeflags_string_posix);
2313         case 2:
2314                 append_char('P', &makeflags_string);
2315                 append_char('P', &makeflags_string_posix);
2316         case 1:
2317                 append_char('P', &makeflags_string);
2318                 append_char('P', &makeflags_string_posix);
2319         }
2320         if (trace_status) {
2321                 append_char('p', &makeflags_string);
2322                 append_char('p', &makeflags_string_posix);
2323         }
2324         if (quest) {
2325                 append_char('q', &makeflags_string);
2326                 append_char('q', &makeflags_string_posix);
2327         }
2328         if (silent_all) {
2329                 append_char('s', &makeflags_string);
2330                 append_char('s', &makeflags_string_posix);
2331         }
2332         if (touch) {
2333                 append_char('t', &makeflags_string);
2334                 append_char('t', &makeflags_string_posix);
2335         }
2336         if (build_unconditional) {
2337                 append_char('u', &makeflags_string);
2338                 append_char('u', &makeflags_string_posix);
2339         }
2340         if (report_cwd) {
2341                 append_char('w', &makeflags_string);
2342                 append_char('w', &makeflags_string_posix);
2343         }
2344 #ifndef PARALLEL
2345         /* -c dmake_rcfile */
2346         if (dmake_rcfile_specified) {
2347                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_RCFILE"));
2348                 dmake_rcfile = GETNAME(wcs_buffer, FIND_LENGTH);
2349                 append_makeflags_string(dmake_rcfile, &makeflags_string);
2350                 append_makeflags_string(dmake_rcfile, &makeflags_string_posix);
2351         }
2352         /* -g dmake_group */
2353         if (dmake_group_specified) {
2354                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_GROUP"));
2355                 dmake_group = GETNAME(wcs_buffer, FIND_LENGTH);
2356                 append_makeflags_string(dmake_group, &makeflags_string);
2357                 append_makeflags_string(dmake_group, &makeflags_string_posix);
2358         }
2359         /* -j dmake_max_jobs */
2360         if (dmake_max_jobs_specified) {
2361                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MAX_JOBS"));
2362                 dmake_max_jobs = GETNAME(wcs_buffer, FIND_LENGTH);
2363                 append_makeflags_string(dmake_max_jobs, &makeflags_string);
2364                 append_makeflags_string(dmake_max_jobs, &makeflags_string_posix)
2365         }
```

```
2366         /* -m dmake_mode */
2367         if (dmake_mode_specified) {
2368                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_MODE"));
2369                 dmake_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2370                 append_makeflags_string(dmake_mode, &makeflags_string);
2371                 append_makeflags_string(dmake_mode, &makeflags_string_posix);
2372         }
2373         /* -x dmake_compat_mode */
2374 //      if (dmake_compat_mode_specified) {
2375 //              MBSTOWCS(wcs_buffer, NOCATGETS("SUN_MAKE_COMPAT_MODE"));
2376 //              dmake_compat_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2377 //              append_makeflags_string(dmake_compat_mode, &makeflags_string);
2378 //              append_makeflags_string(dmake_compat_mode, &makeflags_string_pos
2379 //      }
2380         /* -x dmake_output_mode */
2381         if (dmake_output_mode_specified) {
2382                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_OUTPUT_MODE"));
2383                 dmake_output_mode = GETNAME(wcs_buffer, FIND_LENGTH);
2384                 append_makeflags_string(dmake_output_mode, &makeflags_string);
2385                 append_makeflags_string(dmake_output_mode, &makeflags_string_pos
2386         }
2387         /* -o dmake_odir */
2388         if (dmake_odir_specified) {
2389                 MBSTOWCS(wcs_buffer, NOCATGETS("DMAKE_ODIR"));
2390                 dmake_odir = GETNAME(wcs_buffer, FIND_LENGTH);
2391                 append_makeflags_string(dmake_odir, &makeflags_string);
2392                 append_makeflags_string(dmake_odir, &makeflags_string_posix);
2393         }
2394         /* -M pmake_machinesfile */
2395         if (pmake_machinesfile_specified) {
2396                 MBSTOWCS(wcs_buffer, NOCATGETS("PMAKE_MACHINESFILE"));
2397                 pmake_machinesfile = GETNAME(wcs_buffer, FIND_LENGTH);
2398                 append_makeflags_string(pmake_machinesfile, &makeflags_string);
2399                 append_makeflags_string(pmake_machinesfile, &makeflags_string_po
2400         }
2401         /* -R */
2402         if (pmake_cap_r_specified) {
2403                 append_char((int) space_char, &makeflags_string);
2404                 append_char((int) hyphen_char, &makeflags_string);
2405                 append_char('R', &makeflags_string);
2406                 append_char((int) space_char, &makeflags_string_posix);
2407                 append_char((int) hyphen_char, &makeflags_string_posix);
2408                 append_char('R', &makeflags_string_posix);
2409         }
2410 #endif

2412 /*
2413  *      Make sure MAKEFLAGS is exported
2414  */
2415         maybe_append_prop(makeflags, macro_prop)->
2416           body.macro.exported = true;

2418         if (makeflags_string.buffer.start[1] != (int) nul_char) {
2419                 if (makeflags_string.buffer.start[1] != (int) space_char) {
2420                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2421                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2422                                       GETNAME(makeflags_string.buffer.start,
2423                                               FIND_LENGTH),
2424                                       false);
2425                 } else {
2426                         MBSTOWCS(wcs_buffer, NOCATGETS("MFLAGS"));
2427                         (void) SETVAR(GETNAME(wcs_buffer, FIND_LENGTH),
2428                                       GETNAME(makeflags_string.buffer.start + 2,
2429                                               FIND_LENGTH),
2430                                       false);
2431                 }
```



```
2432         }

2434 /*
2435  *      Add command line macro to POSIX makeflags_string
2436  */
2437         if (makeflags_and_macro.start) {
2438                 tmp_char = (char) space_char;
2439                 cp = makeflags_and_macro.start;
2440                 do {
2441                         append_char(tmp_char, &makeflags_string_posix);
2442                 } while ( tmp_char = *cp++ );
2443                 retmem_mb(makeflags_and_macro.start);
2444         }

2446 /*
2447  *      Now set the value of MAKEFLAGS macro in accordance
2448  *      with current mode.
2449  */
2450         macro = maybe_append_prop(makeflags, macro_prop);
2451         temp = (Boolean) macro->body.macro.read_only;
2452         macro->body.macro.read_only = false;
2453         if(posix || gnu_style) {
2454                 makeflags_string_current = &makeflags_string_posix;
2455         } else {
2456                 makeflags_string_current = &makeflags_string;
2457         }
2458         if (makeflags_string_current->buffer.start[1] == (int) nul_char) {
2459                 makeflags_value_saved =
2460                         GETNAME( makeflags_string_current->buffer.start + 1
2461                                , FIND_LENGTH
2462                                );
2463         } else {
2464                 if (makeflags_string_current->buffer.start[1] != (int) space_cha
2465                         makeflags_value_saved =
2466                                 GETNAME( makeflags_string_current->buffer.start
2467                                        , FIND_LENGTH
2468                                        );
2469                 } else {
2470                         makeflags_value_saved =
2471                                 GETNAME( makeflags_string_current->buffer.start
2472                                        , FIND_LENGTH
2473                                        );
2474                 }
2475         }
2476         (void) SETVAR( makeflags
2477                      , makeflags_value_saved
2478                      , false
2479                      );
2480         macro->body.macro.read_only = temp;

2482 /*
2483  *      Read command line "-f" arguments and ignore -c, g, j, K, M, m, O and o a
2484  */
2485         save_do_not_exec_rule = do_not_exec_rule;
2486         do_not_exec_rule = false;
2487         if (read_trace_level > 0) {
2488                 trace_reader = true;
2489         }

2491         for (i = 1; i < argc; i++) {
2492                 if (argv[i] &&
2493                     (argv[i][0] == (int) hyphen_char) &&
2494                     (argv[i][1] == 'f') &&
2495                     (argv[i][2] == (int) nul_char)) {
2496                         argv[i] = NULL;         /* Remove -f */
2497                         if (i >= argc - 1) {
```

```
2498                                fatal(catgets(catd, 1, 190, "No filename argumen
2499                        }
2500                        MBSTOWCS(wcs_buffer, argv[++i]);
2501                        primary_makefile = GETNAME(wcs_buffer, FIND_LENGTH);
2502                        (void) read_makefile(primary_makefile, true, true, true)
2503                        argv[i] = NULL;         /* Remove filename */
2504                        makefile_read = true;
2505                } else if (argv[i] &&
2506                           (argv[i][0] == (int) hyphen_char) &&
2507                           (argv[i][1] == 'c' ||
2508                            argv[i][1] == 'g' ||
2509                            argv[i][1] == 'j' ||
2510                            argv[i][1] == 'K' ||
2511                            argv[i][1] == 'M' ||
2512                            argv[i][1] == 'm' ||
2513                            argv[i][1] == 'O' ||
2514                            argv[i][1] == 'o') &&
2515                           (argv[i][2] == (int) nul_char)) {
2516                        argv[i] = NULL;
2517                        argv[++i] = NULL;
2518                }
2519        }

2521 /*
2522  *      If no command line "-f" args then look for "makefile", and then for
2523  *      "Makefile" if "makefile" isn't found.
2524  */
2525        if (!makefile_read) {
2526                (void) read_dir(dot,
2527                                (wchar_t *) NULL,
2528                                (Property) NULL,
2529                                (wchar_t *) NULL);
2530            if (!posix) {
2531                if (makefile_name->stat.is_file) {
2532                        if (Makefile->stat.is_file) {
2533                                warning(catgets(catd, 1, 310, "Both 'makefile' a
2534                        }
2535                        primary_makefile = makefile_name;
2536                        makefile_read = read_makefile(makefile_name,
2537                                                      false,
2538                                                      false,
2539                                                      true);
2540                }
2541                if (!makefile_read &&
2542                    Makefile->stat.is_file) {
2543                        primary_makefile = Makefile;
2544                        makefile_read = read_makefile(Makefile,
2545                                                      false,
2546                                                      false,
2547                                                      true);
2548                }
2549            } else {

2551                enum sccs_stat save_m_has_sccs = NO_SCCS;
2552                enum sccs_stat save_M_has_sccs = NO_SCCS;

2554                if (makefile_name->stat.is_file) {
2555                        if (Makefile->stat.is_file) {
2556                                warning(catgets(catd, 1, 191, "Both 'makefile' a
2557                        }
2558                }
2559                if (makefile_name->stat.is_file) {
2560                        if (makefile_name->stat.has_sccs == NO_SCCS) {
2561                           primary_makefile = makefile_name;
2562                           makefile_read = read_makefile(makefile_name,
2563                                                         false,
```



```
2564                                                         false,
2565                                                         true);
2566                        } else {
2567                          save_m_has_sccs = makefile_name->stat.has_sccs;
2568                          makefile_name->stat.has_sccs = NO_SCCS;
2569                          primary_makefile = makefile_name;
2570                          makefile_read = read_makefile(makefile_name,
2571                                                        false,
2572                                                        false,
2573                                                        true);
2574                        }
2575                }
2576                if (!makefile_read &&
2577                    Makefile->stat.is_file) {
2578                        if (Makefile->stat.has_sccs == NO_SCCS) {
2579                           primary_makefile = Makefile;
2580                           makefile_read = read_makefile(Makefile,
2581                                                         false,
2582                                                         false,
2583                                                         true);
2584                        } else {
2585                          save_M_has_sccs = Makefile->stat.has_sccs;
2586                          Makefile->stat.has_sccs = NO_SCCS;
2587                          primary_makefile = Makefile;
2588                          makefile_read = read_makefile(Makefile,
2589                                                        false,
2590                                                        false,
2591                                                        true);
2592                        }
2593                }
2594                if (!makefile_read &&
2595                        makefile_name->stat.is_file) {
2596                           makefile_name->stat.has_sccs = save_m_has_sccs;
2597                           primary_makefile = makefile_name;
2598                           makefile_read = read_makefile(makefile_name,
2599                                                         false,
2600                                                         false,
2601                                                         true);
2602                }
2603                if (!makefile_read &&
2604                    Makefile->stat.is_file) {
2605                           Makefile->stat.has_sccs = save_M_has_sccs;
2606                           primary_makefile = Makefile;
2607                           makefile_read = read_makefile(Makefile,
2608                                                         false,
2609                                                         false,
2610                                                         true);
2611                }
2612            }
2613        }
2614        do_not_exec_rule = save_do_not_exec_rule;
2615        allrules_read = makefile_read;
2616        trace_reader = false;

2618 /*
2619  *      Now get current value of MAKEFLAGS and compare it with
2620  *      the saved value we set before reading makefile.
2621  *      If they are different then MAKEFLAGS is subsequently set by
2622  *      makefile, just leave it there. Otherwise, if make mode
2623  *      is changed by using .POSIX target in makefile we need
2624  *      to correct MAKEFLAGS value.
2625  */
2626        Name mf_val = getvar(makeflags);
2627        if( (posix != is_xpg4)
2628         && (!strcmp(mf_val->string_mb, makeflags_value_saved->string_mb)))
2629        {
```

```
2630                if (makeflags_string_posix.buffer.start[1] == (int) nul_char) {
2631                        (void) SETVAR(makeflags,
2632                                      GETNAME(makeflags_string_posix.buffer.star
2633                                              FIND_LENGTH),
2634                                      false);
2635                } else {
2636                        if (makeflags_string_posix.buffer.start[1] != (int) spac
2637                                (void) SETVAR(makeflags,
2638                                              GETNAME(makeflags_string_posix.buf
2639                                                      FIND_LENGTH),
2640                                              false);
2641                        } else {
2642                                (void) SETVAR(makeflags,
2643                                              GETNAME(makeflags_string_posix.buf
2644                                                      FIND_LENGTH),
2645                                              false);
2646                        }
2647                }
2648        }

2650        if (makeflags_string.free_after_use) {
2651                retmem(makeflags_string.buffer.start);
2652        }
2653        if (makeflags_string_posix.free_after_use) {
2654                retmem(makeflags_string_posix.buffer.start);
2655        }
2656        makeflags_string.buffer.start = NULL;
2657        makeflags_string_posix.buffer.start = NULL;

2659        if (posix) {
2660                /*
2661                 * If the user did not redefine the ARFLAGS macro in the
2662                 * default makefile (make.rules), then we'd like to
2663                 * change the macro value of ARFLAGS to be in accordance
2664                 * with "POSIX" requirements.
2665                 */
2666                MBSTOWCS(wcs_buffer, NOCATGETS("ARFLAGS"));
2667                name = GETNAME(wcs_buffer, wslen(wcs_buffer));
2668                macro = get_prop(name->prop, macro_prop);
2669                if ((macro != NULL) && /* Maybe (macro == NULL) || ? */
2670                    (IS_EQUAL(macro->body.macro.value->string_mb,
2671                              NOCATGETS("rv")))) {
2672                        MBSTOWCS(wcs_buffer, NOCATGETS("-rv"));
2673                        value = GETNAME(wcs_buffer, wslen(wcs_buffer));
2674                        (void) SETVAR(name,
2675                                      value,
2676                                      false);
2677                }
2678        }

2680        if (!posix && !svr4) {
2681                set_sgs_support();
2682        }


2685 /*
2686  *      Make sure KEEP_STATE is in the environment if KEEP_STATE is on.
2687  */
2688        macro = get_prop(keep_state_name->prop, macro_prop);
2689        if ((macro != NULL) &&
2690            macro->body.macro.exported) {
2691                keep_state = true;
2692        }
2693        if (keep_state) {
2694                if (macro == NULL) {
2695                        macro = maybe_append_prop(keep_state_name,
```



```
2696                                                  macro_prop);
2697                }
2698                macro->body.macro.exported = true;
2699                (void) SETVAR(keep_state_name,
2700                              empty_name,
2701                              false);

2703                /*
2704                 *      Read state file
2705                 */

2707                /* Before we read state, let's make sure we have
2708                ** right state file.
2709                */
2710                /* just in case macro references are used in make_state file
2711                ** name, we better expand them at this stage using expand_value.
2712                */
2713                INIT_STRING_FROM_STACK(dest, destbuffer);
2714                expand_value(make_state, &dest, false);

2716                make_state = GETNAME(dest.buffer.start, FIND_LENGTH);

2718                if(!stat(make_state->string_mb, &make_state_stat)) {
2719                   if(!(make_state_stat.st_mode & S_IFREG) ) {
2720                        /* copy the make_state structure to the other
2721                        ** and then let make_state point to the new
2722                        ** one.
2723                        */
2724                      memcpy(&state_filename, make_state,sizeof(state_filename))
2725                      state_filename.string_mb = state_file_str_mb;
2726                /* Just a kludge to avoid two slashes back to back */
2727                      if((make_state->hash.length == 1)&&
2728                              (make_state->string_mb[0] == '/')) {
2729                         make_state->hash.length = 0;
2730                         make_state->string_mb[0] = '\0';
2731                      }
2732                      sprintf(state_file_str_mb,NOCATGETS("%s%s"),
2733                       make_state->string_mb,NOCATGETS("/.make.state"));
2734                      make_state = &state_filename;
2735                        /* adjust the length to reflect the appended string */
2736                      make_state->hash.length += 12;
2737                   }
2738                } else { /* the file doesn't exist or no permission */
2739                   char tmp_path[MAXPATHLEN];
2740                   char *slashp;

2742                   if (slashp = strrchr(make_state->string_mb, '/')) {
2743                      strncpy(tmp_path, make_state->string_mb,
2744                                (slashp - make_state->string_mb));
2745                        tmp_path[slashp - make_state->string_mb]=0;
2746                      if(strlen(tmp_path)) {
2747                        if(stat(tmp_path, &make_state_stat)) {
2748                          warning(catgets(catd, 1, 192, "directory %s for .KEEP_
2749                        }
2750                        if (access(tmp_path, F_OK) != 0) {
2751                          warning(catgets(catd, 1, 193, "can't access dir %s"),t
2752                        }
2753                      }
2754                   }
2755                }
2756                if (report_dependencies_level != 1) {
2757                        Makefile_type   makefile_type_temp = makefile_type;
2758                        makefile_type = reading_statefile;
2759                        if (read_trace_level > 1) {
2760                                trace_reader = true;
2761                        }
```

```
2762                        (void) read_simple_file(make_state,
2763                                                false,
2764                                                false,
2765                                                false,
2766                                                false,
2767                                                false,
2768                                                true);
2769                        trace_reader = false;
2770                        makefile_type = makefile_type_temp;
2771                }
2772        }
2773 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   26575 Wed May 20 11:24:56 2015
new/usr/src/cmd/make/bin/misc.cc
make: unifdef SUN5_0 (defined)
**********************************************************
```

_____*unchanged_portion_omitted_*

```
 115 /*****************************************
 116  *
 117  *      String manipulation
 118  */

 120 /*****************************************
 121  *
 122  *      Nameblock property handling
 123  */

 125 /*****************************************
 126  *
 127  *      Error message handling
 128  */

 130 /*
 131  *      fatal(format, args...)
 132  *
 133  *      Print a message and die
 134  *
 135  *      Parameters:
 136  *              format          printf type format string
 137  *              args            Arguments to match the format
 138  *
 139  *      Global variables used:
 140  *              fatal_in_progress Indicates if this is a recursive call
 141  *              parallel_process_cnt Do we need to wait for anything?
 142  *              report_pwd      Should we report the current path?
 143  */
 144 /*VARARGS*/
 145 void
 146 fatal(char * message, ...)
 147 {
 148         va_list args;

 150         va_start(args, message);
 151         (void) fflush(stdout);
 152 #ifdef DISTRIBUTED
 153         (void) fprintf(stderr, catgets(catd, 1, 262, "dmake: Fatal error: "));
 154 #else
 155         (void) fprintf(stderr, catgets(catd, 1, 263, "make: Fatal error: "));
 156 #endif
 157         (void) vfprintf(stderr, message, args);
 158         (void) fprintf(stderr, "\n");
 159         va_end(args);
 160         if (report_pwd) {
 161                 (void) fprintf(stderr,
 162                                catgets(catd, 1, 156, "Current working directory
 163                                get_current_path());
 164         }
 165         (void) fflush(stderr);
 166         if (fatal_in_progress) {
 167 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 167                 exit_status = 1;
 169 #endif
 168                 exit(1);
 169         }
 170         fatal_in_progress = true;
 171 #ifdef TEAMWARE_MAKE_CMN
```



```
 172         /* Let all parallel children finish */
 173         if ((dmake_mode_type == parallel_mode) &&
 174             (parallel_process_cnt > 0)) {
 175                 (void) fprintf(stderr,
 176                                catgets(catd, 1, 157, "Waiting for %d %s to finis
 177                                parallel_process_cnt,
 178                                parallel_process_cnt == 1 ?
 179                                catgets(catd, 1, 158, "job") : catgets(catd, 1, 1
 180                 (void) fflush(stderr);
 181         }

 183         while (parallel_process_cnt > 0) {
 184 #ifdef DISTRIBUTED
 185                 if (dmake_mode_type == distributed_mode) {
 186                         (void) await_dist(false);
 187                 } else {
 188                         await_parallel(true);
 189                 }
 190 #else
 191                 await_parallel(true);
 192 #endif
 193                 finish_children(false);
 194         }
 195 #endif

 197 #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
 198         job_adjust_fini();
 199 #endif

 203 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 201         exit_status = 1;
 205 #endif
 202         exit(1);
 203 }
```

_____*unchanged_portion_omitted_*

```
 273 /*
 274  *      get_current_path()
 275  *
 276  *      Stuff current_path with the current path if it isnt there already.
 277  *
 278  *      Parameters:
 279  *
 280  *      Global variables used:
 281  */
 282 char *
 283 get_current_path(void)
 284 {
 285         char                    pwd[(MAXPATHLEN * MB_LEN_MAX)];
 286         static char             *current_path;

 288         if (current_path == NULL) {
 293 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 289                 getcwd(pwd, sizeof(pwd));
 295 #else
 296                 (void) getwd(pwd);
 297 #endif
 290                 if (pwd[0] == (int) nul_char) {
 291                         pwd[0] = (int) slash_char;
 292                         pwd[1] = (int) nul_char;
 293 #ifdef DISTRIBUTED
 294                         current_path = strdup(pwd);
 295                 } else if (IS_EQUALN(pwd, NOCATGETS("/tmp_mnt"), 8)) {
 296                         current_path = strdup(pwd + 8);
 297                 } else {
 298                         current_path = strdup(pwd);
```

```
                }
#else
                }
                current_path = strdup(pwd);
#endif
        }
        return current_path;
}
```

*unchanged_portion_omitted_*

```
/*************************************
 *
 *      main() support
 */

/*
 *      load_cached_names()
 *
 *      Load the vector of cached names
 *
 *      Parameters:
 *
 *      Global variables used:
 *              Many many pointers to Name blocks.
 */
void
load_cached_names(void)
{
        char            *cp;
        Name            dollar;

        /* Load the cached_names struct */
        MBSTOWCS(wcs_buffer, NOCATGETS(".BUILT_LAST_MAKE_RUN"));
        built_last_make_run = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("@"));
        c_at = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(" *conditionals* "));
        conditionals = GETNAME(wcs_buffer, FIND_LENGTH);
        /*
         * A version of make was released with NSE 1.0 that used
         * VERSION-1.1 but this version is identical to VERSION-1.0.
         * The version mismatch code makes a special case for this
         * situation.  If the version number is changed from 1.0
         * it should go to 1.2.
         */
        MBSTOWCS(wcs_buffer, NOCATGETS("VERSION-1.0"));
        current_make_version = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".SVR4"));
        svr4_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".POSIX"));
        posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".DEFAULT"));
        default_rule_name = GETNAME(wcs_buffer, FIND_LENGTH);
#ifdef NSE
        MBSTOWCS(wcs_buffer, NOCATGETS(".DERIVED_SRC"));
        derived_src= GETNAME(wcs_buffer, FIND_LENGTH);
#endif
        MBSTOWCS(wcs_buffer, NOCATGETS("$"));
        dollar = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".DONE"));
        done = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("."));
        dot = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE"));
        dot_keep_state = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".KEEP_STATE_FILE"));
```



```
        dot_keep_state_file = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(""));
        empty_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(" FORCE"));
        force = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("HOST_ARCH"));
        host_arch = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("HOST_MACH"));
        host_mach = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".IGNORE"));
        ignore_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".INIT"));
        init = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".LOCAL"));
        localhost_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".make.state"));
        make_state = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("MAKEFLAGS"));
        makeflags = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".MAKE_VERSION"));
        make_version = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".NO_PARALLEL"));
        no_parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".NOT_AUTO"));
        not_auto = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".PARALLEL"));
        parallel_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("PATH"));
        path_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("+"));
        plus = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".PRECIOUS"));
        precious = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("?"));
        query = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("^"));
        hat = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".RECURSIVE"));
        recursive_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET"));
        sccs_get_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".SCCS_GET_POSIX"));
        sccs_get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".GET"));
        get_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".GET_POSIX"));
        get_posix_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("SHELL"));
        shell_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".SILENT"));
        silent_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".SUFFIXES"));
        suffixes_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, SUNPRO_DEPENDENCIES);
        sunpro_dependencies = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_ARCH"));
        target_arch = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("TARGET_MACH"));
        target_mach = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("VIRTUAL_ROOT"));
        virtual_root = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS("VPATH"));
        vpath_name = GETNAME(wcs_buffer, FIND_LENGTH);
        MBSTOWCS(wcs_buffer, NOCATGETS(".WAIT"));
        wait_name = GETNAME(wcs_buffer, FIND_LENGTH);
```

```
696         wait_name->state = build_ok;

698         /* Mark special targets so that the reader treats them properly */
699         svr4_name->special_reader = svr4_special;
700         posix_name->special_reader = posix_special;
701         built_last_make_run->special_reader = built_last_make_run_special;
702         default_rule_name->special_reader = default_special;
703 #ifdef NSE
704         derived_src->special_reader= derived_src_special;
705 #endif
706         dot_keep_state->special_reader = keep_state_special;
707         dot_keep_state_file->special_reader = keep_state_file_special;
708         ignore_name->special_reader = ignore_special;
709         make_version->special_reader = make_version_special;
710         no_parallel_name->special_reader = no_parallel_special;
711         parallel_name->special_reader = parallel_special;
712         localhost_name->special_reader = localhost_special;
713         precious->special_reader = precious_special;
714         sccs_get_name->special_reader = sccs_get_special;
715         sccs_get_posix_name->special_reader = sccs_get_posix_special;
716         get_name->special_reader = get_special;
717         get_posix_name->special_reader = get_posix_special;
718         silent_name->special_reader = silent_special;
719         suffixes_name->special_reader = suffixes_special;

721         /* The value of $$ is $ */
722         (void) SETVAR(dollar, dollar, false);
723         dollar->dollar = false;

725         /* Set the value of $(SHELL) */
726         #ifdef HP_UX
727         MBSTOWCS(wcs_buffer, NOCATGETS("/bin/posix/sh"));
728         #else
737         #if defined(SUN5_0)
729         if (posix) {
730           MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
731         } else {
732           MBSTOWCS(wcs_buffer, NOCATGETS("/bin/sh"));
733         }
743         #else  /* ^SUN5_0 */
744         MBSTOWCS(wcs_buffer, NOCATGETS("/bin/sh"));
745         #endif /* ^SUN5_0 */
734         #endif
735         (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), false);

737         /*
738          * Use " FORCE" to simulate a FRC dependency for :: type
739          * targets with no dependencies.
740          */
741         (void) append_prop(force, line_prop);
742         force->stat.time = file_max_time;

744         /* Make sure VPATH is defined before current dir is read */
745         if ((cp = getenv(vpath_name->string_mb)) != NULL) {
746                 MBSTOWCS(wcs_buffer, cp);
747                 (void) SETVAR(vpath_name,
748                               GETNAME(wcs_buffer, FIND_LENGTH),
749                               false);
750         }

752         /* Check if there is NO PATH variable. If not we construct one. */
753         if (getenv(path_name->string_mb) == NULL) {
754                 vroot_path = NULL;
755                 add_dir_to_path(NOCATGETS("."), &vroot_path, -1);
756                 add_dir_to_path(NOCATGETS("/bin"), &vroot_path, -1);
757                 add_dir_to_path(NOCATGETS("/usr/bin"), &vroot_path, -1);
```



```
758         }
759 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   61524 Wed May 20 11:24:56 2015
new/usr/src/cmd/make/bin/parallel.cc
make: unifdef SUN5_0 (defined)
**********************************************************
______unchanged_portion_omitted_

2053 /*
2054  * This function replaces the makesh binary.
2055  */
2056
2057 #ifdef SGE_SUPPORT
2058 #define DO_CHECK(f)     if (f <= 0) { \
2059                                 fprintf(stderr, \
2060                                         catgets(catd, 1, 347, "Could not write t
2061                                                 script_file, errmsg(errno)); \
2062                                 _exit(1); \
2063                         }
2064 #endif /* SGE_SUPPORT */

2066 static pid_t
2067 run_rule_commands(char *host, char **commands)
2068 {
2069         Boolean         always_exec;
2070         Name            command;
2071         Boolean         ignore;
2072         int             length;
2073         Doname          result;
2074         Boolean         silent_flag;
2075 #ifdef SGE_SUPPORT
2076         wchar_t         *wcmd, *tmp_wcs_buffer = NULL;
2077         char            *cmd, *tmp_mbs_buffer = NULL;
2078         FILE            *scrfp;
2079         Name            shell = getvar(shell_name);
2080 #else
2081         wchar_t         *tmp_wcs_buffer;
2082 #endif /* SGE_SUPPORT */

2084         childPid = fork();
2085         switch (childPid) {
2086         case -1:        /* Error */
2087                 fatal(catgets(catd, 1, 337, "Could not fork child process for dm
2088                       errmsg(errno));
2089                 break;
2090         case 0:         /* Child */
2091                 /* To control the processed targets list is not the child's busi
2092                 running_list = NULL;
2093 #if defined(REDIRECT_ERR)
2094                 if(out_err_same) {
2095                         redirect_io(stdout_file, (char*)NULL);
2096                 } else {
2097                         redirect_io(stdout_file, stderr_file);
2098                 }
2099 #else
2100                 redirect_io(stdout_file, (char*)NULL);
2101 #endif
2102 #ifdef SGE_SUPPORT
2103                 if (grid) {
2104                         int fdes = mkstemp(script_file);
2105                         if ((fdes < 0) || (scrfp = fdopen(fdes, "w")) == NULL) {
2106                                 fprintf(stderr,
2107                                         catgets(catd, 1, 341, "Could not create
2108                                         script_file, errmsg(errno));
2109                                 _exit(1);
2110                         }
2111                         if (IS_EQUAL(shell->string_mb, "")) {
```



```
2112                                 shell = shell_name;
2113                         }
2114                 }
2115 #endif /* SGE_SUPPORT */
2116                 for (commands = commands;
2117                      (*commands != (char *)NULL);
2118                      commands++) {
2119                         silent_flag = silent;
2120                         ignore = false;
2121                         always_exec = false;
2122                         while ((**commands == (int) at_char) ||
2123                                (**commands == (int) hyphen_char) ||
2124                                (**commands == (int) plus_char)) {
2125                                 if (**commands == (int) at_char) {
2126                                         silent_flag = true;
2127                                 }
2128                                 if (**commands == (int) hyphen_char) {
2129                                         ignore = true;
2130                                 }
2131                                 if (**commands == (int) plus_char) {
2132                                         always_exec = true;
2133                                 }
2134                                 (*commands)++;
2135                         }
2136 #ifdef SGE_SUPPORT
2137                         if (grid) {
2138                                 if ((length = strlen(*commands)) >= MAXPATHLEN /
2139                                         wcmd = tmp_wcs_buffer = ALLOC_WC(length
2140                                         (void) mbstowcs(tmp_wcs_buffer, *command
2141                                 } else {
2142                                         MBSTOWCS(wcs_buffer, *commands);
2143                                         wcmd = wcs_buffer;
2144                                         cmd = mbs_buffer;
2145                                 }
2146                                 wchar_t *from = wcmd + wslen(wcmd);
2147                                 wchar_t *to = from + (from - wcmd);
2148                                 *to = (int) nul_char;
2149                                 while (from > wcmd) {
2150                                         *--to = *--from;
2151                                         if (*from == (int) newline_char) { // ne
2152                                                 *--to = *--from;
2153                                         } else if (wschr(char_semantics_char, *f
2154                                                 *--to = (int) backslash_char;
2155                                         }
2156                                 }
2157                                 if (length >= MAXPATHLEN*MB_LEN_MAX/2) { // size
2158                                         cmd = tmp_mbs_buffer = getmem((length *
2159                                         (void) wcstombs(tmp_mbs_buffer, to, (len
2160                                 } else {
2161                                         WCSTOMBS(mbs_buffer, to);
2162                                         cmd = mbs_buffer;
2163                                 }
2164                                 char *mbst, *mbend;
2165                                 if ((length > 0) &&
2166                                     !silent_flag) {
2167                                         for (mbst = cmd; (mbend = strstr(mbst, "
2168                                                 *mbend = '\0';
2169                                                 DO_CHECK(fprintf(scrfp, NOCATGET
2170                                                 *mbend = '\\';
2171                                         }
2172                                         DO_CHECK(fprintf(scrfp, NOCATGETS("/usr/
2173                                 }
2174                                 if (!do_not_exec_rule ||
2175                                     !working_on_targets ||
2176                                     always_exec) {
2177 #if defined(linux)
```

```
2178                                    if (0 != strcmp(shell->string_mb, (char*
2179                                            DO_CHECK(fprintf(scrfp, NOCATGET
2180                                    } else
2181 #endif
2182                                    DO_CHECK(fprintf(scrfp, NOCATGETS("%s -c
2183                                    DO_CHECK(fputs(NOCATGETS("__DMAKECMDEXIT
2184                                    if (ignore) {
2185                                            DO_CHECK(fprintf(scrfp, NOCATGET
2186                                                    catgets(catd, 1, 343, "\
2187                                                    catgets(catd, 1, 344, "(
2188                                    } else {
2189                                            DO_CHECK(fprintf(scrfp, NOCATGET
2190                                                    catgets(catd, 1, 342, "\
2191                                    }
2192                                    if (silent_flag) {
2193                                            DO_CHECK(fprintf(scrfp, NOCATGET
2194                                                    catgets(catd, 1, 345, "T
2195                                            for (mbst = cmd; (mbend = strstr
2196                                                    *mbend = '\0';
2197                                                    DO_CHECK(fprintf(scrfp,
2198                                                    *mbend = '\\';
2199                                            }
2200                                            DO_CHECK(fprintf(scrfp, NOCATGET
2201                                    }
2202                                    if (!ignore) {
2203                                            DO_CHECK(fputs(NOCATGETS("\texit
2204                                    }
2205                                    DO_CHECK(fputs(NOCATGETS("fi\n"), scrfp)
2206                            }
2207                            if (tmp_wcs_buffer) {
2208                                    retmem_mb(tmp_mbs_buffer);
2209                                    tmp_mbs_buffer = NULL;
2210                            }
2211                            if (tmp_wcs_buffer) {
2212                                    retmem(tmp_wcs_buffer);
2213                                    tmp_wcs_buffer = NULL;
2214                            }
2215                            continue;
2216                    }
2217 #endif /* SGE_SUPPORT */
2218                    if ((length = strlen(*commands)) >= MAXPATHLEN) {
2219                            tmp_wcs_buffer = ALLOC_WC(length + 1);
2220                            (void) mbstowcs(tmp_wcs_buffer, *commands, lengt
2221                            command = GETNAME(tmp_wcs_buffer, FIND_LENGTH);
2222                            retmem(tmp_wcs_buffer);
2223                    } else {
2224                            MBSTOWCS(wcs_buffer, *commands);
2225                            command = GETNAME(wcs_buffer, FIND_LENGTH);
2226                    }
2227                    if ((command->hash.length > 0) &&
2228                        !silent_flag) {
2229                            (void) printf("%s\n", command->string_mb);
2230                    }
2231                    result = dosys(command,
2232                                   ignore,
2233                                   false,
2234                                   false, /* bugs #4085164 & #4990057 */
2235                                   /* BOOLEAN(silent_flag && ignore), */
2236                                   always_exec,
2237                                   (Name) NULL,
2238                                   false);
2239                    if (result == build_failed) {
2240                            if (silent_flag) {
2241                                    (void) printf(catgets(catd, 1, 152, "The
2242                            }
2243                            if (!ignore) {
```



```
2244                                            _exit(1);
2245                                    }
2246                            }
2247                    }
2248 #ifndef SGE_SUPPORT
2249                    _exit(0);
2250 #else
2251                    if (!grid) {
2252                            _exit(0);
2253                    }
2254                    DO_CHECK(fputs(NOCATGETS("exit 0\n"), scrfp));
2255                    if (fclose(scrfp) != 0) {
2256                            fprintf(stderr,
2257                                    catgets(catd, 1, 346, "Could not close file: %s:
2258                                    script_file, errmsg(errno));
2259                            _exit(1);
2260                    }
2261                    {

2263 #define DEFAULT_QRSH_TRIES_NUMBER       1
2264 #define DEFAULT_QRSH_TIMEOUT            0

2266                    static char     *sge_env_var = NULL;
2267                    static int      qrsh_tries_number = DEFAULT_QRSH_TRIES_NUMBER;
2268                    static int      qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2269 #define SGE_DEBUG
2270 #ifdef SGE_DEBUG
2271                    static  Boolean do_not_remove = false;
2272 #endif /* SGE_DEBUG */
2273                    if (sge_env_var == NULL) {
2274                            sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TRIES")
2275                            if (sge_env_var != NULL) {
2276                                    qrsh_tries_number = atoi(sge_env_var);
2277                                    if (qrsh_tries_number < 1 || qrsh_tries_number >
2278                                            qrsh_tries_number = DEFAULT_QRSH_TRIES_N
2279                                    }
2280                            }
2281                            sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_TIMEOUT
2282                            if (sge_env_var != NULL) {
2283                                    qrsh_timeout = atoi(sge_env_var);
2284                                    if (qrsh_timeout <= 0) {
2285                                            qrsh_timeout = DEFAULT_QRSH_TIMEOUT;
2286                                    }
2287                            } else {
2288                                    sge_env_var = "";
2289                            }
2290 #ifdef SGE_DEBUG
2291                            sge_env_var = getenv(NOCATGETS("__SPRO_DMAKE_SGE_DEBUG")
2292                            if (sge_env_var == NULL) {
2293                                    sge_env_var = "";
2294                            }
2295                            if (strstr(sge_env_var, NOCATGETS("noqrsh")) != NULL)
2296                                    qrsh_tries_number = 0;
2297                            if (strstr(sge_env_var, NOCATGETS("donotremove")) != NUL
2298                                    do_not_remove = true;
2299 #endif /* SGE_DEBUG */
2300                    }
2301                    for (int i = qrsh_tries_number; ; i--)
2302                    if ((childPid = fork()) < 0) {
2303                            fatal(catgets(catd, 1, 348, "Could not fork child proces
2304                                    errmsg(errno));
2305                            _exit(1);
2306                    } else if (childPid == 0) {
2307                            enable_interrupt((void (*) (int))SIG_DFL);
2308                            if (i > 0) {
2309                                    static char qrsh_cmd[50+MAXPATHLEN] = NOCATGETS(
```

```
2310                                    static char *fname_ptr = NULL;
2311                                    static char *argv[] = { NOCATGETS("sh"),
2312                                                            NOCATGETS("-fce"),
2313                                                            qrsh_cmd,
2314                                                            NULL};
2315                                    if (fname_ptr == NULL) {
2316                                            fname_ptr = qrsh_cmd + strlen(qrsh_cmd);
2317                                    }
2318                                    strcpy(fname_ptr, script_file);
2319                                    (void) execve(NOCATGETS("/bin/sh"), argv, enviro
2320                            } else {
2321                                    static char *argv[] = { NOCATGETS("sh"),
2322                                                            script_file,
2323                                                            NULL};
2324                                    (void) execve(NOCATGETS("/bin/sh"), argv, enviro
2325                            }
2326                            fprintf(stderr,
2327                                    catgets(catd, 1, 349, "Could not load ‘qrsh’: %s
2328                                    errmsg(errno));
2329                            _exit(1);
2330                    } else {
2331 #if defined (HP_UX) || defined (linux) || defined (SUN5_0)
2331                            int             status;
2333 #else
2334                            union wait      status;
2335 #endif
2332                            pid_t pid;
2333                            while ((pid = wait(&status)) != childPid) {
2334                                    if (pid == -1) {
2335                                            fprintf(stderr,
2336                                                    catgets(catd, 1, 350, "wait() fa
2337                                                    errmsg(errno));
2338                                            _exit(1);
2339                                    }
2340                            }
2341                            if (status != 0 && i > 0) {
2342                                    if (i > 1) {
2343                                            sleep(qrsh_timeout);
2344                                    }
2345                                    continue;
2346                            }
2347 #ifdef SGE_DEBUG
2348                            if (do_not_remove) {
2349                                    if (status) {
2350                                            fprintf(stderr,
2351                                                    NOCATGETS("SGE script failed: %s
2352                                                    script_file);
2353                                    }
2354                                    _exit(status ? 1 : 0);
2355                            }
2356 #endif /* SGE_DEBUG */
2357                            (void) unlink(script_file);
2358                            _exit(status ? 1 : 0);
2359                    }
2360                    }
2361 #endif /* SGE_SUPPORT */
2362                    break;
2363            default:
2364                    break;
2365            }
2366            return childPid;
2367 }
```

_____***unchanged_portion_omitted_***

```
2467 #endif
```

```
**********************************************************
   57421 Wed May 20 11:24:57 2015
new/usr/src/cmd/make/bin/read.cc
make: unifdef SUN5_0 (defined)
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 *      read.c
 *
 *      This file contains the makefile reader.
 */

/*
 * Included files
 */
#include <avo/avo_alloca.h>             /* alloca() */
#include <errno.h>              /* errno */
#include <fcntl.h>              /* fcntl() */
#include <mk/defs.h>
#include <mksh/macro.h>         /* expand_value(), expand_macro() */
#include <mksh/misc.h>          /* getmem() */
#include <mksh/read.h>          /* get_next_block_fn() */
#include <sys/uio.h>            /* read() */
#include <unistd.h>             /* read(), unlink() */

#if defined(HP_UX) || defined(linux)
#include <avo/types.h>
extern "C" Avo_err *avo_find_run_dir(char **dirp);
#endif

/*
 * typedefs & structs
 */

/*
 * Static variables
 */

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

/*
 * File table of contents
```



```
 */
static  void            parse_makefile(register Name true_makefile_name, registe
static  Source          push_macro_value(register Source bp, register wchar_t *b
extern  void            enter_target_groups_and_dependencies(Name_vector target,
extern  Name            normalize_name(register wchar_t *name_string, register i

/*
 *      read_simple_file(makefile_name, chase_path, doname_it,
 *              complain, must_exist, report_file, lock_makefile)
 *
 *      Make the makefile and setup to read it. Actually read it if it is stdio
 *
 *      Return value:
 *                              false if the read failed
 *
 *      Parameters:
 *              makefile_name   Name of the file to read
 *              chase_path      Use the makefile path when opening file
 *              doname_it       Call doname() to build the file first
 *              complain        Print message if doname/open fails
 *              must_exist      Generate fatal if file is missing
 *              report_file     Report file when running -P
 *              lock_makefile   Lock the makefile when reading
 *
 *      Static variables used:
 *
 *      Global variables used:
 *              do_not_exec_rule Is -n on?
 *              file_being_read Set to the name of the new file
 *              line_number     The number of the current makefile line
 *              makefiles_used  A list of all makefiles used, appended to
 */


Boolean
read_simple_file(register Name makefile_name, register Boolean chase_path, regis
{
        static short            max_include_depth;
        register Property       makefile = maybe_append_prop(makefile_name,
                                                             makefile_prop);
        Boolean                 forget_after_parse = false;
        static pathpt           makefile_path;
        register int            n;
        char                    *path;
        register Source         source = ALLOC(Source);
        Property                orig_makefile = makefile;
        Dependency              *dpp;
        Dependency              dp;
        register int            length;
        wchar_t                 *previous_file_being_read = file_being_read;
        int                     previous_line_number = line_number;
        wchar_t                 previous_current_makefile[MAXPATHLEN];
        Makefile_type           save_makefile_type;
        Name                    normalized_makefile_name;
        register wchar_t        *string_start;
        register wchar_t        *string_end;


#if defined(HP_UX) || defined(linux)
        Avo_err                 *findrundir_err;
        char                    *run_dir, makerules_dir[BUFSIZ];
#endif

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

#ifdef NSE
```

```
128         if (report_file){
129                 wscpy(previous_current_makefile, current_makefile);
130                 wscpy(current_makefile, wcb);
131         }
132 #endif
133         if (max_include_depth++ >= 40) {
134                 fatal(catgets(catd, 1, 66, "Too many nested include statements")
135         }
136         if (makefile->body.makefile.contents != NULL) {
137                 retmem(makefile->body.makefile.contents);
138         }
139         source->inp_buf =
140           source->inp_buf_ptr =
141             source->inp_buf_end = NULL;
142         source->error_converting = false;
143         makefile->body.makefile.contents = NULL;
144         makefile->body.makefile.size = 0;
145         if ((makefile_name->hash.length != 1) ||
146             (wcb[0] != (int) hyphen_char)) {
147                 if ((makefile->body.makefile.contents == NULL) &&
148                     (doname_it)) {
149                         if (makefile_path == NULL) {
150                                 add_dir_to_path(".",
151                                                 &makefile_path,
152                                                 -1);
153 #ifdef SUN5_0
153                                 add_dir_to_path(NOCATGETS("/usr/share/lib/make")
154                                                 &makefile_path,
155                                                 -1);
156                                 add_dir_to_path(NOCATGETS("/etc/default"),
157                                                 &makefile_path,
158                                                 -1);
160 #elif defined(HP_UX)
161                                 findrundir_err = avo_find_run_dir(&run_dir);
162                                 if (! findrundir_err) {
163                                         (void) sprintf(makerules_dir, NOCATGETS(
164                                         add_dir_to_path(makerules_dir,
165                                                         &makefile_path,
166                                                         -1);
167                                 }

169                                 add_dir_to_path(NOCATGETS("/opt/SUNWspro/share/l
170                                                 &makefile_path,
171                                                 -1);
172                                 add_dir_to_path(NOCATGETS("/usr/share/lib/make")
173                                                 &makefile_path,
174                                                 -1);
175 #elif defined(linux)
176                                 findrundir_err = avo_find_run_dir(&run_dir);
177                                 if (! findrundir_err) {
178                                         (void) sprintf(makerules_dir, NOCATGETS(
179                                         add_dir_to_path(makerules_dir,
180                                                         &makefile_path,
181                                                         -1);
182                                 }

184                                 add_dir_to_path(NOCATGETS("/usr/SUNWspro/lib"),
185                                                 &makefile_path,
186                                                 -1);
187                                 add_dir_to_path(NOCATGETS("/opt/SUNWspro/share/l
188                                                 &makefile_path,
189                                                 -1);
190                                 add_dir_to_path(NOCATGETS("/usr/share/lib/make")
191                                                 &makefile_path,
192                                                 -1);
193 #else
```



```
194                                 add_dir_to_path(NOCATGETS("/usr/include/make"),
195                                                 &makefile_path,
196                                                 -1);
197 #endif
159                         }
160                         save_makefile_type = makefile_type;
161                         makefile_type = reading_nothing;
162                         if (doname(makefile_name, true, false) == build_dont_kno
163                                 /* Try normalized filename */
164                                 string_start=get_wstring(makefile_name->string_m
165                                 for (string_end=string_start+1; *string_end != L
166                                 normalized_makefile_name=normalize_name(string_s
167                                 if ((strcmp(makefile_name->string_mb, normalized
168                                         (doname(normalized_makefile_name, true,
169                                         n = access_vroot(makefile_name->string_m
170                                                  4,
171                                                  chase_path ?
172                                                  makefile_path : NULL,
173                                                  VROOT_DEFAULT);
174                                         if (n == 0) {
175                                                 get_vroot_path((char **) NULL,
176                                                        &path,
177                                                        (char **) NULL);
178                                                 if ((path[0] == (int) period_cha
179                                                     (path[1] == (int) slash_char
180                                                       path += 2;
181                                                 }
182                                                 MBSTOWCS(wcs_buffer, path);
183                                                 makefile_name = GETNAME(wcs_buff
184                                                                FIND_LENGTH);
185                                         }
186                                 }
187                                 retmem(string_start);
188                                 /*
189                                  * Commented out: retmem_mb(normalized_makefile_
190                                  * We have to return this memory, but it seems t
191                                  * in dmake or in Sun C++ 5.7 compiler (it works
192                                  * is compiled using Sun C++ 5.6).
193                                  */
194                                 // retmem_mb(normalized_makefile_name->string_mb
195                         }
196                         makefile_type = save_makefile_type;
197                 }
198                 source->string.free_after_use = false;
199                 source->previous = NULL;
200                 source->already_expanded = false;
201                 /* Lock the file for read, but not when -n. */
202                 if (lock_makefile &&
203                     !do_not_exec_rule) {

205                          make_state_lockfile = getmem(strlen(make_state->string_
206                          (void) sprintf(make_state_lockfile,
207                                         NOCATGETS("%s.lock"),
208                                         make_state->string_mb);
209                         (void) file_lock(make_state->string_mb,
210                                          make_state_lockfile,
211                                          (int *) &make_state_locked,
212                                          0);
213                         if(!make_state_locked) {
214                                 printf(NOCATGETS("-- NO LOCKING for read\n"));
215                                 retmem_mb(make_state_lockfile);
216                                 make_state_lockfile = 0;
217                                 return failed;
218                         }
219                 }
220                 if (makefile->body.makefile.contents == NULL) {
```

```
221                         save_makefile_type = makefile_type;
222                         makefile_type = reading_nothing;
223                         if ((doname_it) &&
224                             (doname(makefile_name, true, false) == build_failed)
225                                 if (complain) {
226                                         (void) fprintf(stderr,
227 #ifdef DISTRIBUTED
228                                                 catgets(catd, 1, 67, "dma
229 #else
230                                                 catgets(catd, 1, 237, "ma
231 #endif
232                                                 makefile_name->string_mb)
233                                 }
234                                 max_include_depth--;
235                                 makefile_type = save_makefile_type;
236                                 return failed;
237                         }
238                         makefile_type = save_makefile_type;
239                         //
240                         // Before calling exists() make sure that we have the ri
241                         //
242                         makefile_name->stat.time = file_no_time;

244                         if (exists(makefile_name) == file_doesnt_exist) {
245                                 if (complain ||
246                                     (makefile_name->stat.stat_errno != ENOENT))
247                                         if (must_exist) {
248                                                 fatal(catgets(catd, 1, 68, "Can'
249                                                       makefile_name->string_mb,
250                                                       errmsg(makefile_name->
251                                                             stat.stat_errno));
252                                         } else {
253                                                 warning(catgets(catd, 1, 69, "Ca
254                                                         makefile_name->string_mb
255                                                         errmsg(makefile_name->
256                                                               stat.stat_errno))
257                                         }
258                                 }
259                                 max_include_depth--;
260                                 if(make_state_locked && (make_state_lockfile !=
261                                         (void) unlink(make_state_lockfile);
262                                         retmem_mb(make_state_lockfile);
263                                         make_state_lockfile = NULL;
264                                         make_state_locked = false;
265                                 }
266                                 retmem(wcb);
267                                 retmem_mb((char *)source);
268                                 return failed;
269                         }
270                         /*
271                          * These values are the size and bytes of
272                          * the MULTI-BYTE makefile.
273                          */
274                         orig_makefile->body.makefile.size =
275                           makefile->body.makefile.size =
276                             source->bytes_left_in_file =
277                               makefile_name->stat.size;
278                         if (report_file) {
279                                 for (dpp = &makefiles_used;
280                                      *dpp != NULL;
281                                      dpp = &(*dpp)->next);
282                                 dp = ALLOC(Dependency);
283                                 dp->next = NULL;
284                                 dp->name = makefile_name;
285                                 dp->automatic = false;
286                                 dp->stale = false;
```



```
287                                 dp->built = false;
288                                 *dpp = dp;
289                         }
290                         source->fd = open_vroot(makefile_name->string_mb,
291                                                 O_RDONLY,
292                                                 0,
293                                                 NULL,
294                                                 VROOT_DEFAULT);
295                         if (source->fd < 0) {
296                                 if (complain || (errno != ENOENT)) {
297                                         if (must_exist) {
298                                                 fatal(catgets(catd, 1, 70, "Can'
299                                                       makefile_name->string_mb,
300                                                       errmsg(errno));
301                                         } else {
302                                                 warning(catgets(catd, 1, 71, "Ca
303                                                         makefile_name->string_mb
304                                                         errmsg(errno));
305                                         }
306                                 }
307                                 max_include_depth--;
308                                 return failed;
309                         }
310                         (void) fcntl(source->fd, F_SETFD, 1);
311                         orig_makefile->body.makefile.contents =
312                           makefile->body.makefile.contents =
313                             source->string.text.p =
314                               source->string.buffer.start =
315                                 ALLOC_WC((int) (makefile_name->stat.size + 2));
316                         if (makefile_type == reading_cpp_file) {
317                                 forget_after_parse = true;
318                         }
319                         source->string.text.end = source->string.text.p;
320                         source->string.buffer.end =
321                           source->string.text.p + makefile_name->stat.size;
322                 } else {
323                         /* Do we ever reach here? */
324                         source->fd = -1;
325                         source->string.text.p =
326                           source->string.buffer.start =
327                             makefile->body.makefile.contents;
328                         source->string.text.end =
329                           source->string.buffer.end =
330                             source->string.text.p + makefile->body.makefile.size
331                         source->bytes_left_in_file =
332                           makefile->body.makefile.size;
333                 }
334                 file_being_read = wcb;
335         } else {
336                 char            *stdin_text_p;
337                 char            *stdin_text_end;
338                 char            *stdin_buffer_start;
339                 char            *stdin_buffer_end;
340                 char            *p_mb;
341                 int             num_mb_chars;
342                 size_t          num_wc_chars;

344                 MBSTOWCS(wcs_buffer, NOCATGETS("Standard in"));
345                 makefile_name = GETNAME(wcs_buffer, FIND_LENGTH);
346                 /*
347                  * Memory to read standard in, then convert it
348                  * to wide char strings.
349                  */
350                 stdin_buffer_start =
351                   stdin_text_p = getmem(length = 1024);
352                 stdin_buffer_end = stdin_text_p + length;
```

```
353                 MBSTOWCS(wcs_buffer, NOCATGETS("standard input"));
354                 file_being_read = (wchar_t *) wsdup(wcs_buffer);
355                 line_number = 0;
356                 while ((n = read(fileno(stdin),
357                                  stdin_text_p,
358                                  length)) > 0) {
359                         length -= n;
360                         stdin_text_p += n;
361                         if (length == 0) {
362                                 p_mb = getmem(length = 1024 +
363                                               (stdin_buffer_end -
364                                                stdin_buffer_start));
365                                 (void) strncpy(p_mb,
366                                                stdin_buffer_start,
367                                                (stdin_buffer_end -
368                                                 stdin_buffer_start));
369                                 retmem_mb(stdin_buffer_start);
370                                 stdin_text_p = p_mb +
371                                   (stdin_buffer_end - stdin_buffer_start);
372                                 stdin_buffer_start = p_mb;
373                                 stdin_buffer_end =
374                                   stdin_buffer_start + length;
375                                 length = 1024;
376                         }
377                 }
378                 if (n < 0) {
379                         fatal(catgets(catd, 1, 72, "Error reading standard input
380                               errmsg(errno));
381                 }
382                 stdin_text_p = stdin_buffer_start;
383                 stdin_text_end = stdin_buffer_end - length;
384                 num_mb_chars = stdin_text_end - stdin_text_p;

386                 /*
387                  * Now, convert the sequence of multibyte chars into
388                  * a sequence of corresponding wide character codes.
389                  */
390                 source->string.free_after_use = false;
391                 source->previous = NULL;
392                 source->bytes_left_in_file = 0;
393                 source->fd = -1;
394                 source->already_expanded = false;
395                 source->string.buffer.start =
396                   source->string.text.p = ALLOC_WC(num_mb_chars + 1);
397                 source->string.buffer.end =
398                     source->string.text.p + num_mb_chars;
399                 num_wc_chars = mbstowcs(source->string.text.p,
400                                         stdin_text_p,
401                                         num_mb_chars);
402                 if ((int) num_wc_chars >= 0) {
403                         source->string.text.end =
404                           source->string.text.p + num_wc_chars;
405                 }
406                 (void) retmem_mb(stdin_text_p);
407         }
408         line_number = 1;
409         if (trace_reader) {
410                 (void) printf(catgets(catd, 1, 73, ">>>>>>>>>>>>>>>> Reading mak
411                               makefile_name->string_mb);
412         }
413         parse_makefile(makefile_name, source);
414         if (trace_reader) {
415                 (void) printf(catgets(catd, 1, 74, ">>>>>>>>>>>>>>>> End of make
416                               makefile_name->string_mb);
417         }
418 #ifdef NSE
```



```
419         if (report_file && (previous_current_makefile[0] != NULL)) {
420                 wscpy(current_makefile, previous_current_makefile);
421         }
422 #endif
423         if(file_being_read) {
424                 retmem(file_being_read);
425         }
426         file_being_read = previous_file_being_read;
427         line_number = previous_line_number;
428         makefile_type = reading_nothing;
429         max_include_depth--;
430         if (make_state_locked) {
431                 /* Unlock .make.state. */
432                 unlink(make_state_lockfile);
433                 make_state_locked = false;
434                 retmem_mb(make_state_lockfile);
435         }
436         if (forget_after_parse) {
437                 retmem(makefile->body.makefile.contents);
438                 makefile->body.makefile.contents = NULL;
439         }
440         retmem_mb((char *)source);
441         return succeeded;
442 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   52335 Wed May 20 11:24:58 2015
new/usr/src/cmd/make/bin/read2.cc
make: unifdef SUN5_0 (defined)
**********************************************************
______unchanged_portion_omitted_


1101 /*
1102  *      special_reader(target, depes, command)
1103  *
1104  *      Read the pseudo targets make knows about
1105  *      This handles the special targets that should not be entered as regular
1106  *      target/dependency sets.
1107  *
1108  *      Parameters:
1109  *              target          The special target
1110  *              depes           The list of dependencies it was entered with
1111  *              command         The command it was entered with
1112  *
1113  *      Static variables used:
1114  *              built_last_make_run_seen Set to indicate .BUILT_LAST... seen
1115  *
1116  *      Global variables used:
1117  *              all_parallel    Set to indicate that everything runs parallel
1118  *              svr4            Set when ".SVR4" target is read
1119  *              svr4_name       The Name ".SVR4"
1120  *              posix           Set when ".POSIX" target is read
1121  *              posix_name      The Name ".POSIX"
1122  *              current_make_version The Name "<current version number>"
1123  *              default_rule    Set when ".DEFAULT" target is read
1124  *              default_rule_name The Name ".DEFAULT", used for tracing
1125  *              dot_keep_state  The Name ".KEEP_STATE", used for tracing
1126  *              ignore_errors   Set if ".IGNORE" target is read
1127  *              ignore_name     The Name ".IGNORE", used for tracing
1128  *              keep_state      Set if ".KEEP_STATE" target is read
1129  *              no_parallel_name The Name ".NO_PARALLEL", used for tracing
1130  *              only_parallel   Set to indicate only some targets runs parallel
1131  *              parallel_name   The Name ".PARALLEL", used for tracing
1132  *              precious        The Name ".PRECIOUS", used for tracing
1133  *              sccs_get_name   The Name ".SCCS_GET", used for tracing
1134  *              sccs_get_posix_name The Name ".SCCS_GET_POSIX", used for tracing
1135  *              get_name        The Name ".GET", used for tracing
1136  *              sccs_get_rule   Set when ".SCCS_GET" target is read
1137  *              silent          Set when ".SILENT" target is read
1138  *              silent_name     The Name ".SILENT", used for tracing
1139  *              trace_reader    Indicates that we should echo stuff we read
1140  */
1141 void
1142 special_reader(Name target, register Name_vector depes, Cmd_line command)
1143 {
1144         register int            n;

1146         switch (target->special_reader) {

1148         case svr4_special:
1149                 if (depes->used != 0) {
1150                         fatal_reader(catgets(catd, 1, 98, "Illegal dependencies
1151                                      target->string_mb);
1152                 }
1153                 svr4  = true;
1154                 posix  = false;
1155                 keep_state = false;
1156                 all_parallel = false;
1157                 only_parallel = false;
1158                 if (trace_reader) {
```



```
1159                         (void) printf("%s:\n", svr4_name->string_mb);
1160                 }
1161                 break;

1163         case posix_special:
1164                 if(svr4)
1165                   break;
1166                 if (depes->used != 0) {
1167                         fatal_reader(catgets(catd, 1, 99, "Illegal dependencies
1168                                      target->string_mb);
1169                 }
1170                 posix  = true;
1171                         /* with posix on, use the posix get rule */
1172                 sccs_get_rule = sccs_get_posix_rule;
1173                         /* turn keep state off being SunPro make specific */
1174                 keep_state = false;
1175                 #if defined(SUN5_0)
1175                 /* Use /usr/xpg4/bin/sh on Solaris */
1176                 MBSTOWCS(wcs_buffer, NOCATGETS("/usr/xpg4/bin/sh"));
1177                 (void) SETVAR(shell_name, GETNAME(wcs_buffer, FIND_LENGTH), fals
1179                 #endif
1178                 if (trace_reader) {
1179                         (void) printf("%s:\n", posix_name->string_mb);
1180                 }
1181                 break;

1183         case built_last_make_run_special:
1184                 built_last_make_run_seen = true;
1185                 break;

1187         case default_special:
1188                 if (depes->used != 0) {
1189                         warning(catgets(catd, 1, 100, "Illegal dependency list f
1190                                 target->string_mb);
1191                 }
1192                 default_rule = command;
1193                 if (trace_reader) {
1194                         (void) printf("%s:\n",
1195                                       default_rule_name->string_mb);
1196                         print_rule(command);
1197                 }
1198                 break;

1200 #ifdef NSE
1201         case derived_src_special:
1202                 for (; depes != NULL; depes= depes->next)
1203                         for (n= 0; n < depes->used; n++) {
1204                                 if (trace_reader)
1205                                         (void)printf("%s:\t%s\n",
1206                                         precious->string_mb,
1207                                         depes->names[n]->string_mb);
1208                                 depes->names[n]->stat.is_derived_src= true;
1209                         };
1210                 break;
1211 #endif

1213         case ignore_special:
1214                 if ((depes->used != 0) &&(!posix)){
1215                         fatal_reader(catgets(catd, 1, 101, "Illegal dependencies
1216                                      target->string_mb);
1217                 }
1218                 if (depes->used == 0)
1219                 {
1220                    ignore_errors_all = true;
1221                 }
1222                 if(svr4) {
```

```
1223                    ignore_errors_all = true;
1224                    break;
1225                }
1226                for (; depes != NULL; depes = depes->next) {
1227                        for (n = 0; n < depes->used; n++) {
1228                                depes->names[n]->ignore_error_mode = true;
1229                        }
1230                }
1231                if (trace_reader) {
1232                        (void) printf("%s:\n", ignore_name->string_mb);
1233                }
1234                break;

1236        case keep_state_special:
1237                if(svr4)
1238                  break;
1239                        /* ignore keep state, being SunPro make specific */
1240                if(posix)
1241                  break;
1242                if (depes->used != 0) {
1243                        fatal_reader(catgets(catd, 1, 102, "Illegal dependencies
1244                                        target->string_mb);
1245                }
1246                keep_state = true;
1247                if (trace_reader) {
1248                        (void) printf("%s:\n",
1249                                        dot_keep_state->string_mb);
1250                }
1251                break;

1253        case keep_state_file_special:
1254                if(svr4)
1255                  break;
1256                if(posix)
1257                  break;
1258                        /* it's not necessary to specify KEEP_STATE, if this
1259                        ** is given, so set the keep_state.
1260                        */
1261                keep_state = true;
1262                if (depes->used != 0) {
1263                   if((!make_state) ||(!strcmp(make_state->string_mb,NOCATGETS("
1264                     make_state = depes->names[0];
1265                   }
1266                }
1267                break;
1268        case make_version_special:
1269                if(svr4)
1270                  break;
1271                if (depes->used != 1) {
1272                        fatal_reader(catgets(catd, 1, 103, "Illegal dependency l
1273                                        target->string_mb);
1274                }
1275                if (depes->names[0] != current_make_version) {
1276                        /*
1277                         * Special case the fact that version 1.0 and 1.1
1278                         * are identical.
1279                         */
1280                        if (!IS_EQUAL(depes->names[0]->string_mb,
1281                                        NOCATGETS("VERSION-1.1")) ||
1282                            !IS_EQUAL(current_make_version->string_mb,
1283                                        NOCATGETS("VERSION-1.0"))) {
1284                                /*
1285                                 * Version mismatches should cause the
1286                                 * .make.state file to be skipped.
1287                                 * This is currently not true - it is read
1288                                 * anyway.
```



```
1289                                 */
1290                                warning(catgets(catd, 1, 104, "Version mismatch
1291                                        current_make_version->string_mb,
1292                                        depes->names[0]->string_mb);
1293                        }
1294                }
1295                break;

1297        case no_parallel_special:
1298                if(svr4)
1299                  break;
1300                /* Set the no_parallel bit for all the targets on */
1301                /* the dependency list */
1302                if (depes->used == 0) {
1303                        /* only those explicitly made parallel */
1304                        only_parallel = true;
1305                        all_parallel = false;
1306                }
1307                for (; depes != NULL; depes = depes->next) {
1308                        for (n = 0; n < depes->used; n++) {
1309                                if (trace_reader) {
1310                                        (void) printf("%s:\t%s\n",
1311                                                        no_parallel_name->string_m
1312                                                        depes->names[n]->string_mb
1313                                }
1314                                depes->names[n]->no_parallel = true;
1315                                depes->names[n]->parallel = false;
1316                        }
1317                }
1318                break;

1320        case parallel_special:
1321                if(svr4)
1322                  break;
1323                if (depes->used == 0) {
1324                        /* everything runs in parallel */
1325                        all_parallel = true;
1326                        only_parallel = false;
1327                }
1328                /* Set the parallel bit for all the targets on */
1329                /* the dependency list */
1330                for (; depes != NULL; depes = depes->next) {
1331                        for (n = 0; n < depes->used; n++) {
1332                                if (trace_reader) {
1333                                        (void) printf("%s:\t%s\n",
1334                                                        parallel_name->string_mb,
1335                                                        depes->names[n]->string_mb
1336                                }
1337                                depes->names[n]->parallel = true;
1338                                depes->names[n]->no_parallel = false;
1339                        }
1340                }
1341                break;

1343        case localhost_special:
1344                if(svr4)
1345                  break;
1346                /* Set the no_parallel bit for all the targets on */
1347                /* the dependency list */
1348                if (depes->used == 0) {
1349                        /* only those explicitly made parallel */
1350                        only_parallel = true;
1351                        all_parallel = false;
1352                }
1353                for (; depes != NULL; depes = depes->next) {
1354                        for (n = 0; n < depes->used; n++) {
```

```
1355                                if (trace_reader) {
1356                                        (void) printf("%s:\t%s\n",
1357                                                      localhost_name->string_mb,
1358                                                      depes->names[n]->string_mb
1359                                }
1360                                depes->names[n]->no_parallel = true;
1361                                depes->names[n]->parallel = false;
1362                                depes->names[n]->localhost = true;
1363                        }
1364                }
1365                break;

1367        case precious_special:
1368                if (depes->used == 0) {
1369                        /* everything is precious      */
1370                        all_precious = true;
1371                } else {
1372                        all_precious = false;
1373                }
1374                if(svr4) {
1375                  all_precious = true;
1376                  break;
1377                }
1378                /* Set the precious bit for all the targets on */
1379                /* the dependency list */
1380                for (; depes != NULL; depes = depes->next) {
1381                        for (n = 0; n < depes->used; n++) {
1382                                if (trace_reader) {
1383                                        (void) printf("%s:\t%s\n",
1384                                                      precious->string_mb,
1385                                                      depes->names[n]->string_mb
1386                                }
1387                                depes->names[n]->stat.is_precious = true;
1388                        }
1389                }
1390                break;

1392        case sccs_get_special:
1393                if (depes->used != 0) {
1394                        fatal_reader(catgets(catd, 1, 105, "Illegal dependencies
1395                                     target->string_mb);
1396                }
1397                sccs_get_rule = command;
1398                sccs_get_org_rule = command;
1399                if (trace_reader) {
1400                        (void) printf("%s:\n", sccs_get_name->string_mb);
1401                        print_rule(command);
1402                }
1403                break;

1405        case sccs_get_posix_special:
1406                if (depes->used != 0) {
1407                        fatal_reader(catgets(catd, 1, 106, "Illegal dependencies
1408                                     target->string_mb);
1409                }
1410                sccs_get_posix_rule = command;
1411                if (trace_reader) {
1412                        (void) printf("%s:\n", sccs_get_posix_name->string_mb);
1413                        print_rule(command);
1414                }
1415                break;

1417        case get_posix_special:
1418                if (depes->used != 0) {
1419                        fatal_reader(catgets(catd, 1, 107, "Illegal dependencies
1420                                     target->string_mb);
```



```
1421                }
1422                get_posix_rule = command;
1423                if (trace_reader) {
1424                        (void) printf("%s:\n", get_posix_name->string_mb);
1425                        print_rule(command);
1426                }
1427                break;

1429        case get_special:
1430                if(!svr4) {
1431                  break;
1432                }
1433                if (depes->used != 0) {
1434                        fatal_reader(catgets(catd, 1, 108, "Illegal dependencies
1435                                     target->string_mb);
1436                }
1437                get_rule = command;
1438                sccs_get_rule = command;
1439                if (trace_reader) {
1440                        (void) printf("%s:\n", get_name->string_mb);
1441                        print_rule(command);
1442                }
1443                break;

1445        case silent_special:
1446                if ((depes->used != 0) && (!posix)){
1447                        fatal_reader(catgets(catd, 1, 109, "Illegal dependencies
1448                                     target->string_mb);
1449                }
1450                if (depes->used == 0)
1451                {
1452                   silent_all = true;
1453                }
1454                if(svr4) {
1455                  silent_all = true;
1456                  break;
1457                }
1458                for (; depes != NULL; depes = depes->next) {
1459                        for (n = 0; n < depes->used; n++) {
1460                                depes->names[n]->silent_mode = true;
1461                        }
1462                }
1463                if (trace_reader) {
1464                        (void) printf("%s:\n", silent_name->string_mb);
1465                }
1466                break;

1468        case suffixes_special:
1469                read_suffixes_list(depes);
1470                break;

1472        default:

1474                fatal_reader(catgets(catd, 1, 110, "Internal error: Unknown spec
1475        }
1476 }
```

_____***unchanged_portion_omitted_***

```
1861 /*
1862  *      fatal_reader(format, args...)
1863  *
1864  *      Parameters:
1865  *              format          printf style format string
1866  *              args            arguments to match the format
1867  *
1868  *      Global variables used:
```

```
1869  *              file_being_read Name of the makefile being read
1870  *              line_number     Line that is being read
1871  *              report_pwd      Indicates whether current path should be shown
1872  *              temp_file_name  When reading tempfile we report that name
1873  */
1874 /*VARARGS*/
1875 void
1876 fatal_reader(char * pattern, ...)
1877 {
1878         va_list args;
1879         char    message[1000];

1881         va_start(args, pattern);
1882         if (file_being_read != NULL) {
1883                 WCSTOMBS(mbs_buffer, file_being_read);
1884                 if (line_number != 0) {
1885                         (void) sprintf(message,
1886                                        catgets(catd, 1, 112, "%s, line %d: %s"),
1887                                        mbs_buffer,
1888                                        line_number,
1889                                        pattern);
1890                 } else {
1891                         (void) sprintf(message,
1892                                        "%s: %s",
1893                                        mbs_buffer,
1894                                        pattern);
1895                 }
1896                 pattern = message;
1897         }

1899         (void) fflush(stdout);
1900 #ifdef DISTRIBUTED
1901         (void) fprintf(stderr, catgets(catd, 1, 113, "dmake: Fatal error in read
1902 #else
1903         (void) fprintf(stderr, catgets(catd, 1, 238, "make: Fatal error in reade
1904 #endif
1905         (void) vfprintf(stderr, pattern, args);
1906         (void) fprintf(stderr, "\n");
1907         va_end(args);

1909         if (temp_file_name != NULL) {
1910                 (void) fprintf(stderr,
1911 #ifdef DISTRIBUTED
1912                                catgets(catd, 1, 114, "dmake: Temp-file %s not re
1913 #else
1914                                catgets(catd, 1, 239, "make: Temp-file %s not rem
1915 #endif
1916                                temp_file_name->string_mb);
1917                 temp_file_name = NULL;
1918         }

1920         if (report_pwd) {
1921                 (void) fprintf(stderr,
1922                                catgets(catd, 1, 115, "Current working directory
1923                                get_current_path());
1924         }
1925         (void) fflush(stderr);
1928 #if defined(SUN5_0) || defined(HP_UX)
1926         exit_status = 1;
1930 #endif
1927         exit(1);
1928 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   16548 Wed May 20 11:24:59 2015
new/usr/src/cmd/make/include/mk/defs.h
make: unifdef SUN5_0 (defined)
**********************************************************
```

```
#ifndef _MK_DEFS_H
#define _MK_DEFS_H
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Included files
 */
#ifdef DISTRIBUTED
#       include <dm/Avo_AcknowledgeMsg.h>
#       include <dm/Avo_DoJobMsg.h>
#       include <dm/Avo_JobResultMsg.h>
#endif

#include <mksh/defs.h>

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


/*
 * Defined macros
 */

#define SKIPSPACE(x)    while (*x &&                            \
                               ((*x == (int) space_char) ||     \
                                (*x == (int) tab_char) ||       \
                                (*x == (int) comma_char))) {    \
                                       x++;                     \
                        }

#define SKIPWORD(x)     while (*x &&                            \
                               (*x != (int) space_char) &&      \
                               (*x != (int) tab_char) &&        \
                               (*x != (int) newline_char) &&    \
                               (*x != (int) comma_char) &&      \
                               (*x != (int) equal_char)) {      \
                                       x++;                     \
```



```
                        }

#define SKIPTOEND(x)    while (*x &&                            \
                               (*x != (int) newline_char)) {    \
                                       x++;                     \
                        }

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

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

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

#define SETVAR(name, value, append) \
                                setvar_daemon(name, value, append, no_daemon, \
                                              true, debug_level)
#ifdef SUN5_0
#define MAX(a,b)                (((a)>(b))?(a):(b))
/*
 * New feature added to SUN5_0 make,  invoke the vanilla svr4 make when
 * the USE_SVR4_MAKE environment variable is set.
 */
#define SVR4_MAKE               "/usr/ccs/lib/svr4.make"
#define USE_SVR4_MAKE           "USE_SVR4_MAKE"
#endif
/*
 * The standard MAXHOSTNAMELEN is 64. We want 32.
 */
#define MAX_HOSTNAMELEN         32


/*
 * typedefs & structs
 */
typedef enum {
        no_state,
        scan_name_state,
        scan_command_state,
        enter_dependencies_state,
        enter_conditional_state,
        enter_equal_state,
        illegal_bytes_state,
        illegal_eoln_state,
        poorly_formed_macro_state,
        exit_state
} Reader_state;
```

*unchanged_portion_omitted_*

```
**********************************************************
   24418 Wed May 20 11:25:00 2015
new/usr/src/cmd/make/include/mksh/defs.h
make: unifdef SUN5_0 (defined)
**********************************************************
______unchanged_portion_omitted_
  84 #define BOOLEAN(expr)           ((expr) ? true : false)

  86 /*
  87  * Some random constants (in an enum so dbx knows their values)
  88  */
  89 enum {
  90         update_delay = 30,              /* time between rstat checks */
  91 #ifdef sun386
  92         ar_member_name_len = 14,
  93 #else
  94 #if defined(SUN5_0) || defined(linux)
  94         ar_member_name_len = 1024,
  96 #else
  97         ar_member_name_len = 15,
  98 #endif
  95 #endif

  97         hashsize = 2048                 /* size of hash table */
  98 };
_____unchanged_portion_omitted_

 935 /*
 936  *      extern declarations for all global variables.
 937  *      The actual declarations are in globals.cc
 938  */
 939 extern char             char_semantics[];
 940 extern wchar_t          char_semantics_char[];
 941 extern Macro_list       cond_macro_list;
 942 extern Boolean          conditional_macro_used;
 943 extern Boolean          do_not_exec_rule;               /* '-n' */
 944 extern Boolean          dollarget_seen;
 945 extern Boolean          dollarless_flag;
 946 extern Name             dollarless_value;
 947 extern char             **environ;
 948 extern Envvar           envvar;
 953 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 949 extern int              exit_status;
 955 #endif
 950 extern wchar_t          *file_being_read;
 951 /* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) *
 952 extern Boolean          gnu_style;
 953 extern Name_set         hashtab;
 954 extern Name             host_arch;
 955 extern Name             host_mach;
 956 extern int              line_number;
 957 extern char             *make_state_lockfile;
 958 extern Boolean          make_word_mentioned;
 959 extern Makefile_type    makefile_type;
 960 extern char             mbs_buffer[];
 961 extern Name             path_name;
 962 extern Boolean          posix;
 963 extern Name             query;
 964 extern Boolean          query_mentioned;
 965 extern Name             hat;
 966 extern Boolean          reading_environment;
 967 extern Name             shell_name;
 968 extern Boolean          svr4;
 969 extern Name             target_arch;
 970 extern Name             target_mach;
 971 extern Boolean          tilde_rule;
```



```
 972 extern wchar_t          wcs_buffer[];
 973 extern Boolean          working_on_targets;
 974 extern Name             virtual_root;
 975 extern Boolean          vpath_defined;
 976 extern Name             vpath_name;
 977 extern Boolean          make_state_locked;
 978 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
 979 extern Boolean          out_err_same;
 980 #endif
 981 extern pid_t            childPid;
 982 extern nl_catd          libmksh_catd;

 984 /*
 985  * RFE 1257407: make does not use fine granularity time info available from stat
 986  * High resolution time comparison.
 987  */

 989 inline int
 990 operator==(const timestruc_t &t1, const timestruc_t &t2) {
 991         return ((t1.tv_sec == t2.tv_sec) && (t1.tv_nsec == t2.tv_nsec));
 992 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    1898 Wed May 20 11:25:00 2015
new/usr/src/cmd/make/include/vroot/args.h
make: unifdef SUN5_0 (defined)
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 1999 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */


#ifndef _ARGS_H_
#define _ARGS_H_

#include <sys/syscall.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/param.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>

typedef enum { rw_read, rw_write} rwt, *rwpt;

extern  void    translate_with_thunk(register char *filename, int (*thunk) (char

union Args {
        struct { int mode;} access;
        struct { int mode;} chmod;
        struct { int user; int group;} chown;
        struct { int mode;} creat;
        struct { char **argv; char **environ;} execve;
        struct { struct stat *buffer;} lstat;
        struct { int mode;} mkdir;
        struct { char *name; int mode;} mount;
        struct { int flags; int mode;} open;
        struct { char *buffer; int buffer_size;} readlink;
        struct { struct stat *buffer;} stat;
#ifndef SUN5_0
        struct { struct statfs *buffer;} statfs;
#endif
        struct { int length;} truncate;
        struct { struct timeval *time;} utimes;
};
_____unchanged_portion_omitted_
```

```
**********************************************************
    1790 Wed May 20 11:25:01 2015
new/usr/src/cmd/make/lib/bsd/bsd.cc
make: unifdef SUN5_0 (defined)
**********************************************************
```

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


#include <signal.h>

#include <bsd/bsd.h>

/* External references.
 */

/* Forward references.
 */

/* Static data.
 */
```



```
extern SIG_PF
bsd_signal (int Signal, SIG_PF Handler)
{
  auto SIG_PF                   previous_handler;
#ifdef SUN5_0
#ifdef sun
  previous_handler = sigset (Signal, Handler);
#else
  auto struct sigaction         new_action;
  auto struct sigaction         old_action;

  new_action.sa_flags = SA_SIGINFO;
  new_action.sa_handler = (void (*) ()) Handler;
  (void) sigemptyset (&new_action.sa_mask);
  (void) sigaddset (&new_action.sa_mask, Signal);

  (void) sigaction (Signal, &new_action, &old_action);

  previous_handler = (SIG_PF) old_action.sa_handler;
#endif
#elif defined(linux)
  previous_handler = sigset (Signal, Handler);
#else
  previous_handler = signal (Signal, Handler);
#endif
  return previous_handler;
}
```

```
 60 extern void
 61 bsd_signals (void)
 62 {
 63   static int                    initialized = 0;

 65   if (initialized == 0)
 66     {
 67       initialized = 1;
 74 #if !defined(SUN5_0) && !defined(linux)
 75 #if defined(SIGHUP)
 76       (void) bsd_signal (SIGHUP, SIG_DFL);
 77 #endif
 78 #if defined(SIGINT)
 79       (void) bsd_signal (SIGINT, SIG_DFL);
 80 #endif
 81 #if defined(SIGQUIT)
 82       (void) bsd_signal (SIGQUIT, SIG_DFL);
 83 #endif
 84 #if defined(SIGILL)
 85       (void) bsd_signal (SIGILL, SIG_DFL);
 86 #endif
 87 #if defined(SIGTRAP)
 88       (void) bsd_signal (SIGTRAP, SIG_DFL);
 89 #endif
 90 #if defined(SIGIOT)
 91       (void) bsd_signal (SIGIOT, SIG_DFL);
 92 #endif
 93 #if defined(SIGABRT)
 94       (void) bsd_signal (SIGABRT, SIG_DFL);
 95 #endif
 96 #if defined(SIGEMT)
 97       (void) bsd_signal (SIGEMT, SIG_DFL);
 98 #endif
 99 #if defined(SIGFPE)
100       (void) bsd_signal (SIGFPE, SIG_DFL);
101 #endif
102 #if defined(SIGBUS)
103       (void) bsd_signal (SIGBUS, SIG_DFL);
104 #endif
105 #if defined(SIGSEGV)
106       (void) bsd_signal (SIGSEGV, SIG_DFL);
107 #endif
108 #if defined(SIGSYS)
109       (void) bsd_signal (SIGSYS, SIG_DFL);
110 #endif
111 #if defined(SIGPIPE)
112       (void) bsd_signal (SIGPIPE, SIG_DFL);
113 #endif
114 #if defined(SIGALRM)
115       (void) bsd_signal (SIGALRM, SIG_DFL);
116 #endif
117 #if defined(SIGTERM)
118       (void) bsd_signal (SIGTERM, SIG_DFL);
119 #endif
120 #if defined(SIGUSR1)
121       (void) bsd_signal (SIGUSR1, SIG_DFL);
122 #endif
123 #if defined(SIGUSR2)
124       (void) bsd_signal (SIGUSR2, SIG_DFL);
125 #endif
126 #if defined(SIGCLD)
127       (void) bsd_signal (SIGCLD, SIG_DFL);
128 #endif
129 #if defined(SIGCHLD)
130       (void) bsd_signal (SIGCHLD, SIG_DFL);
131 #endif
```



```
132 #if defined(SIGPWR)
133       (void) bsd_signal (SIGPWR, SIG_DFL);
134 #endif
135 #if defined(SIGWINCH)
136       (void) bsd_signal (SIGWINCH, SIG_DFL);
137 #endif
138 #if defined(SIGURG)
139       (void) bsd_signal (SIGURG, SIG_DFL);
140 #endif
141 #if defined(SIGIO)
142       (void) bsd_signal (SIGIO, SIG_DFL);
143 #else
144 #if defined(SIGPOLL)
145       (void) bsd_signal (SIGPOLL, SIG_DFL);
146 #endif
147 #endif
148 #if defined(SIGTSTP)
149       (void) bsd_signal (SIGTSTP, SIG_DFL);
150 #endif
151 #if defined(SIGCONT)
152       (void) bsd_signal (SIGCONT, SIG_DFL);
153 #endif
154 #if defined(SIGTTIN)
155       (void) bsd_signal (SIGTTIN, SIG_DFL);
156 #endif
157 #if defined(SIGTTOU)
158       (void) bsd_signal (SIGTTOU, SIG_DFL);
159 #endif
160 #if defined(SIGVTALRM)
161       (void) bsd_signal (SIGVTALRM, SIG_DFL);
162 #endif
163 #if defined(SIGPROF)
164       (void) bsd_signal (SIGPROF, SIG_DFL);
165 #endif
166 #if defined(SIGXCPU)
167       (void) bsd_signal (SIGXCPU, SIG_DFL);
168 #endif
169 #if defined(SIGXFSZ)
170       (void) bsd_signal (SIGXFSZ, SIG_DFL);
171 #endif
172 #endif
 68     }

 70   return;
 71 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   22088 Wed May 20 11:25:01 2015
new/usr/src/cmd/make/lib/mksh/dosys.cc
make: unifdef SUN5_0 (defined)
**********************************************************
______unchanged_portion_omitted_

 599 /*
 600  *      await(ignore_error, silent_error, target, command, running_pid)
 601  *
 602  *      Wait for one child process and analyzes
 603  *      the returned status when the child process terminates.
 604  *
 605  *      Return value:
 606  *                              Returns true if commands ran OK
 607  *
 608  *      Parameters:
 609  *              ignore_error    Should we abort on error?
 610  *              silent_error    Should error messages be suppressed for dmake?
 611  *              target          The target we are building, for error msgs
 612  *              command         The command we ran, for error msgs
 613  *              running_pid     The pid of the process we are waiting for
 614  *
 615  *      Static variables used:
 616  *              filter_file     The fd for the filter file
 617  *              filter_file_name The name of the filter file
 618  *
 619  *      Global variables used:
 620  *              filter_stderr   Set if -X is on
 621  */
 622 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
 623 Boolean
 624 await(register Boolean ignore_error, register Boolean silent_error, Name target,
 625 #else
 626 Boolean
 627 await(register Boolean ignore_error, register Boolean silent_error, Name target,
 628 #endif
 629 {
 630 #ifdef SUN5_0
 630         int                     status;
 632 #else
 633 #ifndef WEXITSTATUS
 634 #define WEXITSTATUS(stat)       stat.w_T.w_Retcode
 635 #endif
 636 #ifndef WTERMSIG
 637 #define WTERMSIG(stat)          stat.w_T.w_Termsig
 638 #endif
 639 #ifndef WCOREDUMP
 640 #define WCOREDUMP(stat)         stat.w_T.w_Coredump
 641 #endif
 642 #if defined (HP_UX) || defined (linux)
 643         int                     status;
 644 #else
 645         union wait              status;
 646 #endif
 647 #endif
 631         char                    *buffer;
 632         int                     core_dumped;
 633         int                     exit_status;
 634 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
 635         Avo_CmdOutput           *make_output_msg;
 636 #endif
 637         FILE                    *outfp;
 638         register pid_t          pid;
 639         struct stat             stat_buff;
 640         int                     termination_signal;
```



```
 641         char                    tmp_buf[MAXPATHLEN];
 642 #if defined(DISTRIBUTED) || defined(MAKETOOL) /* tolik */
 643         RWCollectable           *xdr_msg;
 644 #endif

 646         while ((pid = wait(&status)) != running_pid) {
 647                 if (pid == -1) {
 648                         fatal_mksh(catgets(libmksdmsi18n_catd, 1, 98, "wait() fa
 649                 }
 650         }
 651         (void) fflush(stdout);
 652         (void) fflush(stderr);

 671 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 654         if (status == 0) {

 656 #ifdef PRINT_EXIT_STATUS
 657                 warning_mksh(NOCATGETS("I'm in await(), and status is 0."));
 658 #endif

 660                 return succeeded;
 661         }

 663 #ifdef PRINT_EXIT_STATUS
 664         warning_mksh(NOCATGETS("I'm in await(), and status is *NOT* 0."));
 665 #endif

 685 #else
 686         if (status.w_status == 0) {
 687                 return succeeded;
 688         }
 689 #endif

 668         exit_status = WEXITSTATUS(status);

 670 #ifdef PRINT_EXIT_STATUS
 671         warning_mksh(NOCATGETS("I'm in await(), and exit_status is %d."), exit_s
 672 #endif

 674         termination_signal = WTERMSIG(status);
 675         core_dumped = WCOREDUMP(status);

 677         /*
 678          * If the child returned an error, we now try to print a
 679          * nice message about it.
 680          */
 681         SEND_MTOOL_MSG(
 682                 make_output_msg = new Avo_CmdOutput();
 683                 (void) sprintf(tmp_buf, "%d", job_msg_id);
 684                 make_output_msg->appendOutput(AVO_STRDUP(tmp_buf));
 685         );

 687         tmp_buf[0] = (int) nul_char;
 688         if (!silent_error) {
 689                 if (exit_status != 0) {
 690                         (void) fprintf(stdout,
 691                                         catgets(libmksdmsi18n_catd, 1, 103, "***
 692                                         exit_status);
 693                         SEND_MTOOL_MSG(
 694                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
 695                                                 catgets(libmksdmsi18n_catd, 1, 10
 696                                                 exit_status);
 697                         );
 698                 } else {
 722 #if ! defined(SUN5_0) && ! defined(HP_UX) && ! defined(linux)
 723                         if (termination_signal > NSIG) {
```

```
724 #endif
699                                 (void) fprintf(stdout,
700                                         catgets(libmksdmsi18n_catd, 1, 10
701                                         termination_signal);
702                                 SEND_MTOOL_MSG(
703                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
704                                                 catgets(libmksdmsi18n_cat
705                                                 termination_signal);
706                                 );
733 #if ! defined(SUN5_0) && ! defined(HP_UX) && ! defined(linux)
734                             } else {
735                                 (void) fprintf(stdout,
736                                         "*** %s",
737                                         sys_siglist[termination_signal]);
738                                 SEND_MTOOL_MSG(
739                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
740                                                 "*** %s",
741                                                 sys_siglist[termination_s
742                                 );
743                             }
744 #endif
707                             if (core_dumped) {
708                                 (void) fprintf(stdout,
709                                         catgets(libmksdmsi18n_catd, 1, 10
710                                 SEND_MTOOL_MSG(
711                                         (void) sprintf(&tmp_buf[strlen(tmp_buf)]
712                                                 catgets(libmksdmsi18n_cat
713                                 );
714                             }
715                 }
716                 if (ignore_error) {
717                         (void) fprintf(stdout,
718                                 catgets(libmksdmsi18n_catd, 1, 109, " (ig
719                         SEND_MTOOL_MSG(
720                                 (void) sprintf(&tmp_buf[strlen(tmp_buf)],
721                                         catgets(libmksdmsi18n_catd, 1, 11
722                         );
723                 }
724                 (void) fprintf(stdout, "\n");
725                 (void) fflush(stdout);
726                 SEND_MTOOL_MSG(
727                         make_output_msg->appendOutput(AVO_STRDUP(tmp_buf));
728                 );
729         }
730         SEND_MTOOL_MSG(
731                 xdr_msg = (RWCollectable*) make_output_msg;
732                 xdr(xdrs_p, xdr_msg);
733                 delete make_output_msg;
734         );

736 #ifdef PRINT_EXIT_STATUS
737         warning_mksh(NOCATGETS("I'm in await(), returning failed."));
738 #endif

740         return failed;
741 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3085 Wed May 20 11:25:02 2015
new/usr/src/cmd/make/lib/mksh/globals.cc
make: unifdef SUN5_0 (defined)
**********************************************************
_____unchanged_portion_omitted_
  83 Macro_list      cond_macro_list;
  84 Boolean         conditional_macro_used;
  85 Boolean         do_not_exec_rule;              /* '-n' */
  86 Boolean         dollarget_seen;
  87 Boolean         dollarless_flag;
  88 Name            dollarless_value;
  89 Envvar          envvar;
  90 #ifdef lint
  91 char            **environ;
  92 #endif
  93 #ifdef SUN5_0
  93 int             exit_status;
  95 #endif
  94 wchar_t         *file_being_read;
  95 /* Variable gnu_style=true if env. var. SUN_MAKE_COMPAT_MODE=GNU (RFE 4866328) *
  96 Boolean         gnu_style = false;
  97 Name_set        hashtab;
  98 Name            host_arch;
  99 Name            host_mach;
 100 int             line_number;
 101 char            *make_state_lockfile;
 102 Boolean         make_word_mentioned;
 103 Makefile_type   makefile_type = reading_nothing;
 104 char            mbs_buffer[(MAXPATHLEN * MB_LEN_MAX)];
 105 Name            path_name;
 106 Boolean         posix = true;
 107 Name            hat;
 108 Name            query;
 109 Boolean         query_mentioned;
 110 Boolean         reading_environment;
 111 Name            shell_name;
 112 Boolean         svr4 = false;
 113 Name            target_arch;
 114 Name            target_mach;
 115 Boolean         tilde_rule;
 116 Name            virtual_root;
 117 Boolean         vpath_defined;
 118 Name            vpath_name;
 119 wchar_t         wcs_buffer[MAXPATHLEN];
 120 Boolean         working_on_targets;
 121 #if defined (TEAMWARE_MAKE_CMN) && defined(REDIRECT_ERR)
 122 Boolean         out_err_same;
 123 #endif
 124 pid_t           childPid = -1;  // This variable is used for killing child's pro
 125                                 // Such as qrsh, running command, etc.

 127 /*
 128  * timestamps defined in defs.h
 129  */
 130 const timestruc_t file_no_time          = { -1, 0 };
 131 const timestruc_t file_doesnt_exist     = { 0, 0 };
 132 const timestruc_t file_is_dir           = { 1, 0 };
 133 const timestruc_t file_min_time         = { 2, 0 };
 134 const timestruc_t file_max_time         = { INT_MAX, 0 };
```

```
**********************************************************
   24731 Wed May 20 11:25:03 2015
new/usr/src/cmd/make/lib/mksh/misc.cc
make: unifdef SUN5_0 (defined)
**********************************************************
```

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


/*
 *      misc.cc
 *
 *      This file contains various unclassified routines. Some main groups:
 *              getname
 *              Memory allocation
 *              String handling
 *              Property handling
 *              Error message handling
 *              Make internal state dumping
 *              main routine support
 */

/*
 * Included files
 */
#include <bsd/bsd.h>            /* bsd_signal() */
#include <mksh/i18n.h>          /* get_char_semantics_value() */
#include <mksh/misc.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <stdlib.h>             /* mbstowcs() */
#include <sys/signal.h>         /* SIG_DFL */
#include <sys/wait.h>           /* wait() */

#ifdef SUN5_0
#include <string.h>             /* strerror() */
#endif

#if defined (HP_UX) || defined (linux)
#include <unistd.h>
#endif

/*
 * Defined macros
```



```
 */

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
#ifdef SUN5_0
extern "C" {
        void            (*sigivalue)(int) = SIG_DFL;
        void            (*sigqvalue)(int) = SIG_DFL;
        void            (*sigtvalue)(int) = SIG_DFL;
        void            (*sighvalue)(int) = SIG_DFL;
}
#else
static  void            (*sigivalue)(int) = (void (*) (int)) SIG_DFL;
static  void            (*sigqvalue)(int) = (void (*) (int)) SIG_DFL;
static  void            (*sigtvalue)(int) = (void (*) (int)) SIG_DFL;
static  void            (*sighvalue)(int) = (void (*) (int)) SIG_DFL;
#endif

long    getname_bytes_count = 0;
long    getname_names_count = 0;
long    getname_struct_count = 0;

long    freename_bytes_count = 0;
long    freename_names_count = 0;
long    freename_struct_count = 0;

long    expandstring_count = 0;
long    getwstring_count = 0;

/*
 * File table of contents
 */
static void     expand_string(register String string, register int length);

#define FATAL_ERROR_MSG_SIZE 200

/*
 *      getmem(size)
 *
 *      malloc() version that checks the returned value.
 *
 *      Return value:
 *                              The memory chunk we allocated
 *
 *      Parameters:
 *              size            The size of the chunk we need
 *
 *      Global variables used:
 */
char *
getmem(register int size)
{
        register char           *result = (char *) malloc((unsigned) size);
        if (result == NULL) {
                char buf[FATAL_ERROR_MSG_SIZE];
                sprintf(buf, NOCATGETS("*** Error: malloc(%d) failed: %s\n"), si
                strcat(buf, catgets(libmksdmsi18n_catd, 1, 126, "mksh: Fatal err
                fputs(buf, stderr);
#ifdef SUN5_0
                exit_status = 1;
#endif
```

```
117                 exit(1);
118         }
119         return result;
120 }
______unchanged_portion_omitted_

258 /*
259  *      enable_interrupt(handler)
260  *
261  *      This routine sets a new interrupt handler for the signals make
262  *      wants to deal with.
263  *
264  *      Parameters:
265  *              handler         The function installed as interrupt handler
266  *
267  *      Static variables used:
268  *              sigivalue       The original signal handler
269  *              sigqvalue       The original signal handler
270  *              sigtvalue       The original signal handler
271  *              sighvalue       The original signal handler
272  */
273 void
274 enable_interrupt(register void (*handler) (int))
275 {
287 #ifdef SUN5_0
276         if (sigivalue != SIG_IGN) {
289 #else
290         if (sigivalue != (void (*) (int)) SIG_IGN) {
291 #endif
277                 (void) bsd_signal(SIGINT, (SIG_PF) handler);
278         }
294 #ifdef SUN5_0
279         if (sigqvalue != SIG_IGN) {
296 #else
297         if (sigqvalue != (void (*) (int)) SIG_IGN) {
298 #endif
280                 (void) bsd_signal(SIGQUIT, (SIG_PF) handler);
281         }
301 #ifdef SUN5_0
282         if (sigtvalue != SIG_IGN) {
303 #else
304         if (sigtvalue != (void (*) (int)) SIG_IGN) {
305 #endif
283                 (void) bsd_signal(SIGTERM, (SIG_PF) handler);
284         }
308 #ifdef SUN5_0
285         if (sighvalue != SIG_IGN) {
310 #else
311         if (sighvalue != (void (*) (int)) SIG_IGN) {
312 #endif
286                 (void) bsd_signal(SIGHUP, (SIG_PF) handler);
287         }
288 }
______unchanged_portion_omitted_

334 /*
335  *      errmsg(errnum)
336  *
337  *      Return the error message for a system call error
338  *
339  *      Return value:
340  *                              An error message string
341  *
342  *      Parameters:
343  *              errnum          The number of the error we want to describe
344  *
```



```
345  *      Global variables used:
346  *              sys_errlist     A vector of error messages
347  *              sys_nerr        The size of sys_errlist
348  */
349 char *
350 errmsg(int errnum)
351 {
352 #ifdef linux
353         return strerror(errnum);
354 #else // linux

356         extern int              sys_nerr;
357 #ifdef SUN4_x
358         extern char             *sys_errlist[];
359 #endif
360         char                    *errbuf;

362         if ((errnum < 0) || (errnum > sys_nerr)) {
363                 errbuf = getmem(6+1+11+1);
364                 (void) sprintf(errbuf, catgets(libmksdmsi18n_catd, 1, 127, "Erro
365                 return errbuf;
366         } else {
367 #ifdef SUN4_x
368                 return(sys_errlist[errnum]);
369 #endif
397 #ifdef SUN5_0
370                 return strerror(errnum);
399 #endif

372         }
373 #endif // linux
374 }

376 static char static_buf[MAXPATHLEN*3];

378 /*
379  *      fatal_mksh(format, args...)
380  *
381  *      Print a message and die
382  *
383  *      Parameters:
384  *              format          printf type format string
385  *              args            Arguments to match the format
386  */
387 /*VARARGS*/
388 void
389 fatal_mksh(const char *message, ...)
390 {
391         va_list args;
392         char    *buf = static_buf;
393         char    *mksh_fat_err = catgets(libmksdmsi18n_catd, 1, 128, "mksh: Fatal
394         char    *cur_wrk_dir = catgets(libmksdmsi18n_catd, 1, 129, "Current work
395         int     mksh_fat_err_len = strlen(mksh_fat_err);

397         va_start(args, message);
398         (void) fflush(stdout);
399         (void) strcpy(buf, mksh_fat_err);
400         size_t buf_len = vsnprintf(static_buf + mksh_fat_err_len,
401                                    sizeof(static_buf) - mksh_fat_err_len,
402                                    message, args)
403                         + mksh_fat_err_len
404                         + strlen(cur_wrk_dir)
405                         + strlen(get_current_path_mksh())
406                         + 3; // "\n\n"
407         va_end(args);
408         if (buf_len >= sizeof(static_buf)) {
```

```
409                 buf = getmem(buf_len);
410                 (void) strcpy(buf, mksh_fat_err);
411                 va_start(args, message);
412                 (void) vsprintf(buf + mksh_fat_err_len, message, args);
413                 va_end(args);
414         }
415         (void) strcat(buf, "\n");
416 /*
417         if (report_pwd) {
418  */
419         if (1) {
420                 (void) strcat(buf, cur_wrk_dir);
421                 (void) strcat(buf, get_current_path_mksh());
422                 (void) strcat(buf, "\n");
423         }
424         (void) fputs(buf, stderr);
425         (void) fflush(stderr);
426         if (buf != static_buf) {
427                 retmem_mb(buf);
428         }
458 #ifdef SUN5_0
429         exit_status = 1;
460 #endif
430         exit(1);
431 }

433 /*
434  *      fatal_reader_mksh(format, args...)
435  *
436  *      Parameters:
437  *              format          printf style format string
438  *              args            arguments to match the format
439  */
440 /*VARARGS*/
441 void
442 fatal_reader_mksh(const char * pattern, ...)
443 {
444         va_list args;
445         char    message[1000];

447         va_start(args, pattern);
448 /*
449         if (file_being_read != NULL) {
450                 WCSTOMBS(mbs_buffer, file_being_read);
451                 if (line_number != 0) {
452                         (void) sprintf(message,
453                                        catgets(libmksdmsi18n_catd, 1, 130, "%s,
454                                        mbs_buffer,
455                                        line_number,
456                                        pattern);
457                 } else {
458                         (void) sprintf(message,
459                                        "%s: %s",
460                                        mbs_buffer,
461                                        pattern);
462                 }
463                 pattern = message;
464         }
465  */

467         (void) fflush(stdout);
468         (void) fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 131, "mksh: Fatal
469         (void) vfprintf(stderr, pattern, args);
470         (void) fprintf(stderr, "\n");
471         va_end(args);
```



```
473 /*
474         if (temp_file_name != NULL) {
475                 (void) fprintf(stderr,
476                                catgets(libmksdmsi18n_catd, 1, 132, "mksh: Temp-f
477                                temp_file_name->string_mb);
478                 temp_file_name = NULL;
479         }
480  */

482 /*
483         if (report_pwd) {
484  */
485         if (1) {
486                 (void) fprintf(stderr,
487                                catgets(libmksdmsi18n_catd, 1, 133, "Current work
488                                get_current_path_mksh());
489         }
490         (void) fflush(stderr);
522 #ifdef SUN5_0
491         exit_status = 1;
524 #endif
492         exit(1);
493 }
_____unchanged_portion_omitted_

527 /*
528  *      get_current_path_mksh()
529  *
530  *      Stuff current_path with the current path if it isnt there already.
531  *
532  *      Parameters:
533  *
534  *      Global variables used:
535  */
536 char *
537 get_current_path_mksh(void)
538 {
539         char                    pwd[(MAXPATHLEN * MB_LEN_MAX)];
540         static char             *current_path;

542         if (current_path == NULL) {
576 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
543                 getcwd(pwd, sizeof(pwd));
578 #else
579                 (void) getwd(pwd);
580 #endif
544                 if (pwd[0] == (int) nul_char) {
545                         pwd[0] = (int) slash_char;
546                         pwd[1] = (int) nul_char;
547                 }
548                 current_path = strdup(pwd);
549         }
550         return current_path;
551 }
_____unchanged_portion_omitted_

818 /*
819  *      handle_interrupt_mksh()
820  *
821  *      This is where C-C traps are caught.
822  */
823 void
824 handle_interrupt_mksh(int)
825 {
826         (void) fflush(stdout);
827         /* Make sure the processes running under us terminate first. */
```

```
 828         if (childPid > 0) {
 829                 kill(childPid, SIGTERM);
 830                 childPid = -1;
 831         }
 869 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 832         while (wait((int *) NULL) != -1);
 871 #if defined(SUN5_0)
 833         exit_status = 2;
 873 #endif
 874 #else
 875         while (wait((union wait *) NULL) != -1);
 876 #endif
 834         exit(2);
 835 }

 837 /*
 838  *      setup_interrupt()
 839  *
 840  *      This routine saves the original interrupt handler pointers
 841  *
 842  *      Parameters:
 843  *
 844  *      Static variables used:
 845  *              sigivalue       The original signal handler
 846  *              sigqvalue       The original signal handler
 847  *              sigtvalue       The original signal handler
 848  *              sighvalue       The original signal handler
 849  */
 850 void
 851 setup_interrupt(register void (*handler) (int))
 852 {
 896 #ifdef SUN5_0
 853         sigivalue = bsd_signal(SIGINT, SIG_IGN);
 854         sigqvalue = bsd_signal(SIGQUIT, SIG_IGN);
 855         sigtvalue = bsd_signal(SIGTERM, SIG_IGN);
 856         sighvalue = bsd_signal(SIGHUP, SIG_IGN);
 901 #else
 902         sigivalue = (void (*) (int)) bsd_signal(SIGINT, SIG_IGN);
 903         sigqvalue = (void (*) (int)) bsd_signal(SIGQUIT, SIG_IGN);
 904         sigtvalue = (void (*) (int)) bsd_signal(SIGTERM, SIG_IGN);
 905         sighvalue = (void (*) (int)) bsd_signal(SIGHUP, SIG_IGN);
 906 #endif
 857         enable_interrupt(handler);
 858 }
```

_____unchanged_portion_omitted_

```
**********************************************************
     966 Wed May 20 11:25:04 2015
new/usr/src/cmd/make/lib/vroot/Makefile
make: unifdef SUN5_0 (defined)
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 # Copyright 2015, Richard Lowe.

  14 LIBRARY =       libvroot.a
  15 VERS =          .1
  16 OBJECTS =       access.o        \
  17                 args.o          \
  18                 chdir.o         \
  19                 chmod.o         \
  20                 chown.o         \
  21                 chroot.o        \
  22                 creat.o         \
  23                 execve.o        \
  24                 lock.o          \
  25                 lstat.o         \
  26                 mkdir.o         \
  27                 mount.o         \
  28                 open.o          \
  29                 readlink.o      \
  30                 report.o        \
  31                 rmdir.o         \
  32                 stat.o          \
  33                 statfs.o        \
  33                 truncate.o      \
  34                 unlink.o        \
  36                 unmount.o       \
  35                 utimes.o        \
  36                 vroot.o         \
  37                 setenv.o

  39 include $(SRC)/lib/Makefile.lib
  40 include ../../Makefile.com

  42 LIBS = $(LIBRARY)
  43 SRCDIR = ../
  44 MAPFILES=
  45 CPPFLAGS += -D_FILE_OFFSET_BITS=64

  47 all: $(LIBS)

  49 install: all

  51 lint:

  53 include $(SRC)/lib/Makefile.targ
```

```
**********************************************************
    5406 Wed May 20 11:25:04 2015
new/usr/src/cmd/make/lib/vroot/lock.cc
make: unifdef SUN5_0 (defined)
**********************************************************
```

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

#include <avo/intl.h>   /* for NOCATGETS */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/errno.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <vroot/vroot.h>
#include <mksdmsi18n/mksdmsi18n.h>
#include <signal.h>
#include <errno.h>                      /* errno */

#if !defined(linux)
extern  char            *sys_errlist[];
extern  int             sys_nerr;
#endif

static  void            file_lock_error(char *msg, char *file, char *str, int ar

#define BLOCK_INTERUPTS sigfillset(&newset) ; \
        sigprocmask(SIG_SETMASK, &newset, &oldset)

#define UNBLOCK_INTERUPTS \
        sigprocmask(SIG_SETMASK, &oldset, &newset)

/*
 * This code stolen from the NSE library and changed to not depend
 * upon any NSE routines or header files.
 *
 * Simple file locking.
 * Create a symlink to a file.  The "test and set" will be
 * atomic as creating the symlink provides both functions.
 *
 * The timeout value specifies how long to wait for stale locks
```



```
 * to disappear.  If the lock is more than 'timeout' seconds old
 * then it is ok to blow it away.  This part has a small window
 * of vunerability as the operations of testing the time,
 * removing the lock and creating a new one are not atomic.
 * It would be possible for two processes to both decide to blow
 * away the lock and then have process A remove the lock and establish
 * its own, and then then have process B remove the lock which accidentily
 * removes A's lock rather than the stale one.
 *
 * A further complication is with the NFS.  If the file in question is
 * being served by an NFS server, then its time is set by that server.
 * We can not use the time on the client machine to check for a stale
 * lock.  Therefore, a temp file on the server is created to get
 * the servers current time.
 *
 * Returns an error message.  NULL return means the lock was obtained.
 *
 * 12/6/91 Added the parameter "file_locked".  Before this parameter
 * was added, the calling procedure would have to wait for file_lock()
 * to return before it sets the flag. If the user interrupted "make"
 * between the time the lock was acquired and the time file_lock()
 * returns, make wouldn't know that the file has been locked, and therefore
 * it wouldn' remove the lock. Setting the flag right after locking the file
 * makes this window much smaller.
 */

int
file_lock(char *name, char *lockname, int *file_locked, int timeout)
{
        int             counter = 0;
        static char     msg[MAXPATHLEN+1];
        int             printed_warning = 0;
        int             r;
        struct stat     statb;
        sigset_t newset;
        sigset_t oldset;

        *file_locked = 0;
        if (timeout <= 0) {
                timeout = 120;
        }
        for (;;) {
                BLOCK_INTERUPTS;
                r = symlink(name, lockname);
                if (r == 0) {
                        *file_locked = 1;
                        UNBLOCK_INTERUPTS;
                        return 0; /* success */
                }
                UNBLOCK_INTERUPTS;

                if (errno != EEXIST) {
                        file_lock_error(msg, name, (char *)NOCATGETS("symlink(%s
                            (int) name, (int) lockname);
                        fprintf(stderr, "%s", msg);
                        return errno;
                }

                counter = 0;
                for (;;) {
                        sleep(1);
                        r = lstat(lockname, &statb);
                        if (r == -1) {
                                /*
                                 * The lock must have just gone away - try
                                 * again.
```

```
128                                      */
129                                     break;
130                             }

132                             if ((counter > 5) && (!printed_warning)) {
133                                     /* Print waiting message after 5 secs */
134 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
134                                     (void) getcwd(msg, MAXPATHLEN);
136 #else
137                                     (void) getwd(msg);
138 #endif
135                                     fprintf(stderr,
136                                             catgets(libmksdmsi18n_catd, 1, 162, "fil
137                                             name);
138                                     fprintf(stderr,
139                                             catgets(libmksdmsi18n_catd, 1, 163, "fil
140                                             lockname);
141                                     fprintf(stderr,
142                                             catgets(libmksdmsi18n_catd, 1, 144, "Cur
143                                             msg);

145                                     printed_warning = 1;
146                             }

148                             if (++counter > timeout ) {
149                                     /*
150                                      * Waited enough - return an error..
151                                      */
152                                     return EEXIST;
153                             }
154                     }
155             }
156             /* NOTREACHED */
157 }

159 /*
160  * Format a message telling why the lock could not be created.
161  */
162 static  void
163 file_lock_error(char *msg, char *file, char *str, int arg1, int arg2)
164 {
165             int             len;

167             sprintf(msg, catgets(libmksdmsi18n_catd, 1, 145, "Could not lock file ‘%
168             len = strlen(msg);
169             sprintf(&msg[len], str, arg1, arg2);
170             strcat(msg, catgets(libmksdmsi18n_catd, 1, 146, " failed - "));
171 #if !defined(linux)
172             if (errno < sys_nerr) {
173 #ifdef SUN4_x
174                     strcat(msg, sys_errlist[errno]);
175 #endif
180 #ifdef SUN5_0
176                     strcat(msg, strerror(errno));
182 #endif
177             } else {
178                     len = strlen(msg);
179                     sprintf(&msg[len], NOCATGETS("errno %d"), errno);
180             }
181 #else
182             strcat(msg, strerror(errno));
183 #endif
184 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8880 Wed May 20 11:25:05 2015
new/usr/src/cmd/make/lib/vroot/report.cc
make: unifdef SUN5_0 (defined)
**********************************************************
______unchanged_portion_omitted_

 62 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 62 extern "C" {
 63 static void
 64 close_report_file(void)
 65 {
 66         (void)fputs("\n", report_file);
 67         (void)fclose(report_file);
 68 }
______unchanged_portion_omitted_
 71 #else
 72 static void
 73 close_report_file(int, ...)
 74 {
 75         (void)fputs("\n", report_file);
 76         (void)fclose(report_file);
 77 }
 78 #endif

 71 static void
 72 clean_up(FILE *nse_depinfo_fp, FILE *merge_fp, char *nse_depinfo_file, char *mer
 73 {
 74         fclose(nse_depinfo_fp);
 75         fclose(merge_fp);
 76         fclose(command_output_fp);
 77         unlink(command_output_tmpfile);
 78         if (unlinkf)
 79                 unlink(merge_file);
 80         else
 81                 rename(merge_file, nse_depinfo_file);
 82 }


 85 /*
 86  *  Update the file, if necessary.  We don't want to rewrite
 87  *  the file if we don't have to because we don't want the time of the file
 88  *  to change in that case.
 89  */

100 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
 91 extern "C" {
 92 static void
 93 close_file(void)
104 #else
105 static void
106 close_file(int, ...)
107 #endif
 94 {
 95         char            line[MAXPATHLEN+2];
 96         char            buf[MAXPATHLEN+2];
 97         FILE            *nse_depinfo_fp;
 98         FILE            *merge_fp;
 99         char            nse_depinfo_file[MAXPATHLEN];
100         char            merge_file[MAXPATHLEN];
101         char            lock_file[MAXPATHLEN];
102         int             err;
103         int             len;
104         int             changed = 0;
105         int             file_locked;
```



```
107         fprintf(command_output_fp, "\n");
108         fclose(command_output_fp);
109         if ((command_output_fp = fopen(command_output_tmpfile, "r")) == NULL) {
110                 return;
111         }
112         sprintf(nse_depinfo_file, "%s/%s", search_dir, NSE_DEPINFO);
113         sprintf(merge_file, NOCATGETS("%s/.tmp%s.%d"), search_dir, NSE_DEPINFO,
114         sprintf(lock_file, "%s/%s", search_dir, NSE_DEPINFO_LOCK);
115         err = file_lock(nse_depinfo_file, lock_file, &file_locked, 0);
116         if (err) {
117                 if (warning_ptr != (void (*) (char *, ...)) NULL) {
118                         (*warning_ptr)(catgets(libmksdmsi18n_catd, 1, 147, "Coul
119                     }
120                 unlink(command_output_tmpfile);
121                 return;
122         }
123         /* If .nse_depinfo file doesn't exist */
124         if ((nse_depinfo_fp = fopen(nse_depinfo_file, "r+")) == NULL) {
125                 if (is_path) {
126                         if ((nse_depinfo_fp =
127                              fopen(nse_depinfo_file, "w")) == NULL) {
128                                 fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 1
129                                     nse_depinfo_file);
130                                 unlink(command_output_tmpfile);

132                                 unlink(lock_file);
133                                 return;
134                         }
135                         while (fgets(line, MAXPATHLEN+2, command_output_fp)
136                                != NULL) {
137                                 fprintf(nse_depinfo_fp, "%s", line);
138                         }
139                         fclose(command_output_fp);
140                 }
141                 fclose(nse_depinfo_fp);
142                 if (file_locked) {
143                         unlink(lock_file);
144                 }
145                 unlink(command_output_tmpfile);
146                 return;
147         }
148         if ((merge_fp = fopen(merge_file, "w")) == NULL) {
149                 fprintf(stderr, catgets(libmksdmsi18n_catd, 1, 149, "Cannot open
150                 if (file_locked) {
151                         unlink(lock_file);
152                 }
153                 unlink(command_output_tmpfile);
154                 return;
155         }
156         len = strlen(sfile);
157         while (fgets(line, MAXPATHLEN+2, nse_depinfo_fp) != NULL) {
158                 if (strncmp(line, sfile, len) == 0 && line[len] == ':') {
159                         while (fgets(buf, MAXPATHLEN+2, command_output_fp)
160                                != NULL) {
161                                 if (is_path) {
162                                         fprintf(merge_fp, "%s", buf);
163                                         if (strcmp(line, buf)) {
164                                                 /* changed */
165                                                 changed = 1;
166                                         }
167                                 }
168                                 if (buf[strlen(buf)-1] == '\n') {
169                                         break;
170                                 }
171                         }
172                         if (changed || !is_path) {
```

```
173                                 while (fgets(line, MAXPATHLEN, nse_depinfo_fp)
174                                        != NULL) {
175                                         fputs(line, merge_fp);
176                                 }
177                                 clean_up(nse_depinfo_fp, merge_fp,
178                                         nse_depinfo_file, merge_file, 0);
179                         } else {
180                                 clean_up(nse_depinfo_fp, merge_fp,
181                                         nse_depinfo_file, merge_file, 1);
182                         }
183                         if (file_locked) {
184                                 unlink(lock_file);
185                         }
186                         unlink(command_output_tmpfile);
187                         return;
188                 } /* entry found */
189                 fputs(line, merge_fp);
190         }
191         /* Entry never found.  Add it if there is a search path */
192         if (is_path) {
193                 while (fgets(line, MAXPATHLEN+2, command_output_fp) != NULL) {
194                         fprintf(nse_depinfo_fp, "%s", line);
195                 }
196         }
197         clean_up(nse_depinfo_fp, merge_fp, nse_depinfo_file, merge_file, 1);
198         if (file_locked) {
199                 unlink(lock_file);
200         }
201 }

217 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
203 } // extern "C"
219 #endif

205 static void
206 report_dep(char *iflag, char *filename)
207 {

209         if (command_output_fp == NULL) {
210                 sprintf(command_output_tmpfile,
211                         NOCATGETS("%s/%s.%d.XXXXXX"), tmpdir, NSE_DEPINFO, getpi
212                 int fd = mkstemp(command_output_tmpfile);
213                 if ((fd < 0) || (command_output_fp = fdopen(fd, "w")) == NULL) {
214                         return;
215                 }
216                 if ((search_dir = getenv(NOCATGETS("NSE_DEP"))) == NULL) {
217                         return;
218                 }
235 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
219                 atexit(close_file);
237 #else
238                 on_exit(close_file, 0);
239 #endif
220                 strcpy(sfile, filename);
221                 if (iflag == NULL || *iflag == '\0') {
222                         return;
223                 }
224                 fprintf(command_output_fp, "%s:", sfile);
225         }
226         fprintf(command_output_fp, " ");
227         fprintf(command_output_fp, iflag);
228         if (iflag != NULL) {
229                 is_path = 1;
230         }
231 }
```

*unchanged_portion_omitted_*


```
251 void
252 report_search_path(char *iflag)
253 {
254         char            curdir[MAXPATHLEN];
255         char            *sdir;
256         char            *newiflag;
257         char            filename[MAXPATHLEN];
258         char            *p, *ptr;

260         if ((sdir = getenv(NOCATGETS("NSE_DEP"))) == NULL) {
261                 return;
262         }
263         if ((p= getenv(SUNPRO_DEPENDENCIES)) == NULL) {
264                 return;
265         }
266         ptr = strchr(p, ' ');
267         if( ! ptr ) {
268                 return;
269         }
270         sprintf(filename, NOCATGETS("%s-CPP"), ptr+1);
291 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
271         getcwd(curdir, sizeof(curdir));
293 #else
294         getwd(curdir);
295 #endif
272         if (strcmp(curdir, sdir) != 0 && strlen(iflag) > 2 &&
273             iflag[2] != '/') {
274                 /* Makefile must have had an "cd xx; cc ..." */
275                 /* Modify the -I path to be relative to the cd */
276                 newiflag = (char *)malloc(strlen(iflag) + strlen(curdir) + 2);
277                 sprintf(newiflag, "-%c%s/%s", iflag[1], curdir, &iflag[2]);
278                 report_dep(newiflag, filename);
279         } else {
280                 report_dep(iflag, filename);
281         }
282 }

284 void
285 report_dependency(const char *name)
286 {
287         register char   *filename;
288         char            buffer[MAXPATHLEN+1];
289         register char   *p;
290         register char   *p2;
291         char            nse_depinfo_file[MAXPATHLEN];

293         if (report_file == NULL) {
294                 if ((filename= getenv(SUNPRO_DEPENDENCIES)) == NULL) {
295                         report_file = (FILE *)-1;
296                         return;
297                 }
298                 if (strlen(filename) == 0) {
299                         report_file = (FILE *)-1;
300                         return;
301                 }
302                 (void)strcpy(buffer, name);
303                 name = buffer;
304                 p = strchr(filename, ' ');
305                 if(p) {
306                         *p= 0;
307                 } else {
308                         report_file = (FILE *)-1;
309                         return;
310                 }
311                 if ((report_file= fopen(filename, "a")) == NULL) {
```

```
312                         if ((report_file= fopen(filename, "w")) == NULL) {
313                                 report_file= (FILE *)-1;
314                                 return;
315                         }
316                 }
341 #if defined(SUN5_0) || defined(HP_UX) || defined(linux)
317                 atexit(close_report_file);
343 #else
344                 (void)on_exit(close_report_file, (char *)report_file);
345 #endif
318                 if ((p2= strchr(p+1, ' ')) != NULL)
319                         *p2= 0;
320                 target_being_reported_for= (char *)malloc((unsigned)(strlen(p+1)
321                 (void)strcpy(target_being_reported_for, p+1);
322                 (void)fputs(p+1, report_file);
323                 (void)fputs(":", report_file);
324                 *p= ' ';
325                 if (p2 != NULL)
326                         *p2= ' ';
327         }
328         if (report_file == (FILE *)-1)
329                 return;
330         (void)fputs(name, report_file);
331         (void)fputs(" ", report_file);
332 }
```

_____***unchanged_portion_omitted_***