```
***********************************************************
   86686 Wed May 20 12:22:44 2015
new/usr/src/cmd/make/bin/main.cc
make: unifdef for two bugfixes conditioned for unknown reasons (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.
 */

/*
 *      main.cc
 *
 *      make program main routine plus some helper routines
 */

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


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

#include <pwd.h>                /* getpwnam() */
#include <setjmp.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/errno.h>          /* ENOENT */
#include <sys/stat.h>           /* fstat() */
#include <fcntl.h>              /* open() */

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

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

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

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

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


/*
 * Defined macros
 */
#define LD_SUPPORT_ENV_VAR      "SGS_SUPPORT_32"
#define LD_SUPPORT_ENV_VAR_32   "SGS_SUPPORT_32"
#define LD_SUPPORT_ENV_VAR_64   "SGS_SUPPORT_64"
#define LD_SUPPORT_MAKE_LIB     "libmakestate.so.1"
#ifdef __i386
#define LD_SUPPORT_MAKE_ARCH    "i386"
#elif __sparc
#define LD_SUPPORT_MAKE_ARCH    "sparc"
#else
#error "Unsupported architecture"
#endif

/*
 * typedefs & structs
 */

/*
 * Static variables
 */
static  char            *argv_zero_string;
static  Boolean         build_failed_ever_seen;
static  Boolean         continue_after_error_ever_seen; /* '-k' */
static  Boolean         dmake_group_specified;          /* '-g' */
static  Boolean         dmake_max_jobs_specified;       /* '-j' */
static  Boolean         dmake_mode_specified;           /* '-m' */
static  Boolean         dmake_add_mode_specified;       /* '-x' */
static  Boolean         dmake_output_mode_specified;    /* '-x DMAKE_OUTPUT_MODE
static  Boolean         dmake_compat_mode_specified;    /* '-x SUN_MAKE_COMPAT_M
static  Boolean         dmake_odir_specified;           /* '-o' */
static  Boolean         dmake_rcfile_specified;         /* '-c' */
static  Boolean         env_wins;                       /* '-e' */
static  Boolean         ignore_default_mk;              /* '-r' */
static  Boolean         list_all_targets;               /* '-T' */
static  int             mf_argc;
static  char            **mf_argv;
static  Dependency_rec  not_auto_depen_struct;
static  Dependency      not_auto_depen = &not_auto_depen_struct;
static  Boolean         pmake_cap_r_specified;          /* '-R' */
static  Boolean         pmake_machinesfile_specified;   /* '-M' */
static  Boolean         stop_after_error_ever_seen;     /* '-S' */
static  Boolean         trace_status;                   /* '-p' */

#ifdef DMAKE_STATISTICS
static  Boolean         getname_stat = false;
#endif

        static  time_t          start_time;
        static  int             g_argc;
        static  char            **g_argv;

/*
 * File table of contents
 */
        extern "C" void         cleanup_after_exit(void);
extern "C" {
        extern  void            dmake_exit_callback(void);
        extern  void            dmake_message_callback(char *);
}
______unchanged_portion_omitted_
#endif

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

                MBSTOWCS(wcs_buffer, ".FAILED");
                failed_name = GETNAME(wcs_buffer, FIND_LENGTH);
                if ((exit_status != 0) && (failed_name->prop != NULL)) {
                        /*
                         * [tolik] switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                        (void) doname(failed_name, false, true);
                } else {
                    if (!trace_status) {
                        /*
                         * Switch DMake to serial mode
                         */
                        dmake_mode_type = serial_mode;
                        no_parallel = true;
                        (void) doname(done, false, true);
                    }
                }
            }
        }
        /*
         * 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);

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

/*
 * Convert the MAKEFLAGS string value into a vector of char *, similar
 * to argv.
 */
static void
setup_makeflags_argv()
{
        char            *cp;
        char            *cp1;
        char            *cp2;
        char            *cp3;
        char            *cp_orig;
        Boolean         add_hyphen;
        int             i;
        char            tmp_char;

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

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

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

                        /* New MAKEFLAGS format */

                        add_hyphen = false;

#ifdef ADDFIX5060758
                        /* Check if MAKEFLAGS value begins with multiple
                         * hyphen characters, and remove all duplicates.
                         * Usually it happens when the next command is
                         * used: $(MAKE) -$(MAKEFLAGS)
                         *
                         * This was a workaround for BugID 5060758, but
                         * appears to have survived as a fix in make.
                         * This is a workaround for BugID 5060758.
                         */
                        while (*cp) {
                                if (*cp != (int) hyphen_char) {
                                        break;
                                }
                                cp++;
                                if (*cp == (int) hyphen_char) {
                                        /* There are two hyphens. Skip one */
                                        cp_orig = cp;
                                        cp++;
                                }
                                if (!(*cp)) {
                                        /* There are hyphens only. Skip all */
                                        cp_orig = cp;
                                        break;
                                }
                        }
#endif
                } else {

                        /* Old MAKEFLAGS format */

                        add_hyphen = true;
                }
        }

        /* Find the number of arguments in MAKEFLAGS */
        while (cp && *cp) {
                /* Skip white spaces */
                while (cp && *cp && isspace(*cp)) {
                        cp++;
                }
                if (cp && *cp) {
                        /* Increment arg count */
                        mf_argc++;
                        /* Go to next white space */
                        while (cp && *cp && !isspace(*cp)) {
                                if(*cp == (int) backslash_char) {
                                        cp++;
                                }
                                cp++;
                        }
                }
        }
        /* Allocate memory for the new MAKEFLAGS argv */
        mf_argv = (char **) malloc((mf_argc + 1) * sizeof(char *));
        mf_argv[0] = (char *)"MAKEFLAGS";
        /*
         * Convert the MAKEFLAGS string value into a vector of char *,
         * similar to argv.
         */
        cp = cp_orig;
        for (i = 1; i < mf_argc; i++) {
                /* Skip white spaces */
                while (cp && *cp && isspace(*cp)) {
                        cp++;
                }
                if (cp && *cp) {
                        cp_orig = cp;
                        /* Go to next white space */
                        while (cp && *cp && !isspace(*cp)) {
                                if(*cp == (int) backslash_char) {
                                        cp++;
                                }
                                cp++;
```

```
                        }
                        tmp_char = *cp;
                        *cp = (int) nul_char;
                        if (add_hyphen) {
                                mf_argv[i] = getmem(2 + strlen(cp_orig));
                                mf_argv[i][0] = '\0';
                                (void) strcat(mf_argv[i], "-");
                                // (void) strcat(mf_argv[i], cp_orig);
                                unquote_str(cp_orig, mf_argv[i]+1);
                        } else {
                                mf_argv[i] = getmem(2 + strlen(cp_orig));
                                //mf_argv[i] = strdup(cp_orig);
                                unquote_str(cp_orig, mf_argv[i]);
                        }
                        *cp = tmp_char;
                }
        }
        mf_argv[i] = NULL;
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   19307 Wed May 20 12:22:45 2015
new/usr/src/cmd/make/bin/misc.cc
make: unifdef for two bugfixes conditioned for unknown reasons (defined)
**********************************************************
/*
 * 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 2005 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 <errno.h>
#include <mk/defs.h>
#include <mksh/macro.h>         /* SETVAR() */
#include <mksh/misc.h>          /* enable_interrupt() */
#include <stdarg.h>             /* va_list, va_start(), va_end() */
#include <vroot/report.h>       /* SUNPRO_DEPENDENCIES */
#include <libintl.h>


#define MAXJOBS_ADJUST_RFE4694000

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


/*
 * Defined macros
 */
```

```
/*
 * typedefs & structs
 */

/*
 * Static variables
 */

/*
 * File table of contents
 */
static  void            print_rule(register Name target);
static  void            print_target_n_deps(register Name target);

/*****************************************
 *
 *      getname
 */

/*****************************************
 *
 *      Memory allocation
 */

/*
 *      free_chain()
 *
 *      frees a chain of Name_vector's
 *
 *      Parameters:
 *              ptr             Pointer to the first element in the chain
 *                              to be freed.
 *
 *      Global variables used:
 */
void
free_chain(Name_vector ptr)
{
        if (ptr != NULL) {
                if (ptr->next != NULL) {
                        free_chain(ptr->next);
                }
                free((char *) ptr);
        }
}

/*****************************************
 *
 *      String manipulation
 */

/*****************************************
 *
 *      Nameblock property handling
 */

/*****************************************
 *
 *      Error message handling
 */

/*
 *      fatal(format, args...)
 *
 *      Print a message and die
 *
 *      Parameters:
 *              format          printf type format string
 *              args            Arguments to match the format
 *
 *      Global variables used:
 *              fatal_in_progress Indicates if this is a recursive call
 *              parallel_process_cnt Do we need to wait for anything?
 *              report_pwd      Should we report the current path?
 */
/*VARARGS*/
void
fatal(const char *message, ...)
{
        va_list args;

        va_start(args, message);
        (void) fflush(stdout);
        (void) fprintf(stderr, gettext("make: Fatal error: "));
        (void) vfprintf(stderr, message, args);
        (void) fprintf(stderr, "\n");
        va_end(args);
        if (report_pwd) {
                (void) fprintf(stderr,
                               gettext("Current working directory %s\n"),
                               get_current_path());
        }
        (void) fflush(stderr);
        if (fatal_in_progress) {
                exit_status = 1;
                exit(1);
        }
        fatal_in_progress = true;
        /* Let all parallel children finish */
        if ((dmake_mode_type == parallel_mode) &&
            (parallel_process_cnt > 0)) {
                (void) fprintf(stderr,
                               gettext("Waiting for %d %s to finish\n"),
                               parallel_process_cnt,
                               parallel_process_cnt == 1 ?
                               gettext("job") : gettext("jobs"));
                (void) fflush(stderr);
        }

        while (parallel_process_cnt > 0) {
                await_parallel(true);
                finish_children(false);
        }

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

        exit_status = 1;
        exit(1);
}
```
______***unchanged_portion_omitted_***

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


#define MAXJOBS_ADJUST_RFE4694000

#ifdef MAXJOBS_ADJUST_RFE4694000

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

/*
 *      adjust_pmake_max_jobs (int pmake_max_jobs)
 *
 *      Parameters:
 *              pmake_max_jobs  - max jobs limit set by user
 *
 *      External functions used:
 *              sysconf()
 *              getloadavg()
 */
static int
adjust_pmake_max_jobs (int pmake_max_jobs)
{
        static int      ncpu = 0;
        double          loadavg[3];
        int             adjustment;
        int             adjusted_max_jobs;

        if (ncpu <= 0) {
                if ((ncpu = sysconf(_SC_NPROCESSORS_ONLN)) <= 0) {
                        ncpu = 1;
                }
        }
        if (getloadavg(loadavg, 3) != 3) return(pmake_max_jobs);
        adjustment = ((int)loadavg[LOADAVG_1MIN]);
        if (adjustment < 2) return(pmake_max_jobs);
        if (ncpu > 1) {
                adjustment = adjustment / ncpu;
        }
        adjusted_max_jobs = pmake_max_jobs - adjustment;
        if (adjusted_max_jobs < 1) adjusted_max_jobs = 1;
        return(adjusted_max_jobs);
}
______unchanged_portion_omitted_

#endif /* MAXJOBS_ADJUST_RFE4694000 */

/*
 *      distribute_process(char **commands, Property line)
 *
 *      Parameters:
 *              commands        argv vector of commands to execute
 *
 *      Return value:
 *                              The result of the execution
 *
 *      Static variables used:
 *              process_running Set to the pid of the process set running
```

```
 * #if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
 *              job_adjust_mode Current job adjust mode
 * #endif
 */
static Doname
distribute_process(char **commands, Property line)
{
        static unsigned file_number = 0;
        wchar_t         string[MAXPATHLEN];
        char            mbstring[MAXPATHLEN];
        int             filed;
        int             res;
        int             tmp_index;
        char            *tmp_index_str_ptr;

#if !defined (TEAMWARE_MAKE_CMN) || !defined (MAXJOBS_ADJUST_RFE4694000)
        while (parallel_process_cnt >= pmake_max_jobs) {
                await_parallel(false);
                finish_children(true);
        }
#else /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
        /* initialize adjust mode, if not initialized */
        if (job_adjust_mode == ADJUST_UNKNOWN) {
                job_adjust_init();
        }

        /* actions depend on adjust mode */
        switch (job_adjust_mode) {
        case ADJUST_M1:
                while (parallel_process_cnt >= adjust_pmake_max_jobs (pmake_max_
                        await_parallel(false);
                        finish_children(true);
                }
                break;
        case ADJUST_M2:
                if ((res = m2_acquire_job()) == 0) {
                        if (parallel_process_cnt > 0) {
                                await_parallel(false);
                                finish_children(true);

                                if ((res = m2_acquire_job()) == 0) {
                                        return build_serial;
                                }
                        } else {
                                return build_serial;
                        }
                }
                if (res < 0) {
                        /* job adjustment error */
                        job_adjust_error();

                        /* no adjustment */
                        while (parallel_process_cnt >= pmake_max_jobs) {
                                await_parallel(false);
                                finish_children(true);
                        }
                }
                break;
        default:
                while (parallel_process_cnt >= pmake_max_jobs) {
                        await_parallel(false);
                        finish_children(true);
                }
        }

#endif /* TEAMWARE_MAKE_CMN && MAXJOBS_ADJUST_RFE4694000 */
```

```
        setvar_envvar();
        /*
         * Tell the user what DMake is doing.
         */
        if (!silent && output_mode != txt2_mode) {
                /*
                 * Print local_host --> x job(s).
                 */
                (void) fprintf(stdout,
                               gettext("%s --> %d %s\n"),
                               local_host,
                               parallel_process_cnt + 1,
                               (parallel_process_cnt == 0) ? gettext("job") : ge

                /* Print command line(s). */
                tmp_index = 0;
                while (commands[tmp_index] != NULL) {
                    /* No @ char. */
                    /* XXX - need to add [2] when + prefix is added */
                    if ((commands[tmp_index][0] != (int) at_char) &&
                        (commands[tmp_index][1] != (int) at_char)) {
                        tmp_index_str_ptr = commands[tmp_index];
                        if (*tmp_index_str_ptr == (int) hyphen_char) {
                                tmp_index_str_ptr++;
                        }
                        (void) fprintf(stdout, "%s\n", tmp_index_str_ptr);
                    }
                    tmp_index++;
                }
                (void) fflush(stdout);
        }

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

        mktemp(mbstring);

        stdout_file = strdup(mbstring);
        stderr_file = NULL;

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

                mktemp(mbstring);

                stderr_file = strdup(mbstring);
        }

        process_running = run_rule_commands(local_host, commands);

        return build_running;
}
```
______***unchanged_portion_omitted_***

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

```
 *
 *      Parameters:
 *              waitflg         dwight
 */
void
await_parallel(Boolean waitflg)
{
        Boolean         nohang;
        pid_t           pid;
        int             status;
        Running         rp;
        int             waiterr;

        nohang = false;
        for ( ; ; ) {
                if (!nohang) {
                        (void) alarm((int) update_delay);
                }
                pid = waitpid((pid_t)-1,
                              &status,
                              nohang ? WNOHANG : 0);
                waiterr = errno;
                if (!nohang) {
                        (void) alarm(0);
                }
                if (pid <= 0) {
                        if (waiterr == EINTR) {
                                if (waitflg) {
                                        continue;
                                } else {
                                        return;
                                }
                        } else {
                                return;
                        }
                }
                for (rp = running_list;
                     (rp != NULL) && (rp->pid != pid);
                     rp = rp->next) {
                        ;
                }
                if (rp == NULL) {
                        fatal(gettext("Internal error: returned child pid not in
                } else {
                        rp->state = (WIFEXITED(status) && WEXITSTATUS(status) ==
                }
                nohang = true;
                parallel_process_cnt--;

#if defined (TEAMWARE_MAKE_CMN) && defined (MAXJOBS_ADJUST_RFE4694000)
                if (job_adjust_mode == ADJUST_M2) {
                        if (m2_release_job()) {
                                job_adjust_error();
                        }
                }
#endif
        }
}
```
______***unchanged_portion_omitted_***