****
20138 Wed May 20 12:14:47 2015
new/usr/src/cmd/make/bin/misc.cc
make: remove a bunch of unused and mis-licensed code from history
****
______unchanged_portion_omitted_

```
/*
 *      Copyright (c) 1987-1992 Sun Microsystems, Inc.  All Rights Reserved.
 *      Sun considers its source code as an unpublished, proprietary
 *      trade secret, and it is available only under strict license
 *      provisions.  This copyright notice is placed here only to protect
 *      Sun in the event the source is deemed a published work.  Dissassembly,
 *      decompilation, or other means of reducing the object code to human
 *      readable form is prohibited by the license agreement under which
 *      this code is provided to the user or company in possession of this
 *      copy.
 *      RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the
 *      Government is subject to restrictions as set forth in subparagraph
 *      (c)(1)(ii) of the Rights in Technical Data and Computer Software
 *      clause at DFARS 52.227-7013 and in similar clauses in the FAR and
 *      NASA FAR Supplement.
 *
 * 1.3 91/09/30
 */


/* Some includes are commented because of the includes at the beginning */
/* #include <signal.h> */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/param.h>
/* #include <string.h> */
#include <unistd.h>
#include <stdlib.h>
/* #include <stdio.h> */
/* #include <avo/find_dir.h> */
/* #ifndef TEAMWARE_MAKE_CMN
#include <avo/find_dir.h>
#endif */

/* Routines to find the base directory name from which the various components
 * -executables, *crt* libraries etc will be accessed
 */

/* This routine checks to see if a given filename is an executable or not.
   Logically similar to the csh statement : if  ( -x $i && ! -d $i )
 */

static int
check_if_exec(char *file)
{
        struct stat stb;
        if (stat(file, &stb) < 0) {
                return ( -1);
        }
        if (S_ISDIR(stb.st_mode)) {
                return (-1);
        }
        if (!(stb.st_mode & S_IEXEC)) {
                return ( -1);
        }
        return (0);
}

/* resolve - check for specified file in specified directory
 *      sets up dir, following symlinks.
```

```
 *      returns zero for success, or
 *      -1 for error (with errno set properly)
 */
static int
resolve (const char *indir,     /* search directory */
         const char *cmd,       /* search for name */
         char   *dir,   /* directory buffer */
         char   **run)  /* resultion name ptr ptr */
{
    char               *p;
    int                 rv = -1;
    int                 sll;
    char                symlink[MAXPATHLEN + 1];

    do {
        errno = ENAMETOOLONG;
        if ((strlen (indir) + strlen (cmd) + 2) > (size_t) MAXPATHLEN)
            break;

        sprintf(dir, "%s/%s", indir, cmd);
        if (check_if_exec(dir) != 0)  /* check if dir is an executable */
        {
                break;          /* Not an executable program */
        }

        /* follow symbolic links */
        while ((sll = readlink (dir, symlink, MAXPATHLEN)) >= 0) {
            symlink[sll] = 0;
            if (*symlink == '/')
                strcpy (dir, symlink);
            else
                sprintf (strrchr (dir, '/'), "/%s", symlink);
        }
        if (errno != EINVAL)
            break;

        p = strrchr (dir, '/');
        *p++ = 0;
        if (run)                /* user wants resolution name */
            *run = p;
        rv = 0;                 /* complete, with success! */

    } while (0);

    return rv;
}

/*
 *find_run_directory - find executable file in PATH
 *
 * PARAMETERS:
 *      cmd     filename as typed by user (argv[0])
 *      cwd     buffer from which is read the working directory
 *               if first character is '/' or into which is
 *               copied working directory name otherwise
 *      dir     buffer into which is copied program's directory
 *      pgm     where to return pointer to tail of cmd (may be NULL
 *               if not wanted)
 *      run     where to return pointer to tail of final resolved
 *               name ( dir/run is the program) (may be NULL
 *               if not wanted)
 *      path    user's path from environment
 *
 * Note: run and pgm will agree except when symbolic links have
 *      renamed files
 *
 * RETURNS:
 *      returns zero for success,
 *      -1 for error (with errno set properly).
 *
 * EXAMPLE:
 *      find_run_directory (argv[0], ".", &charray1, (char **) 0, (char **) 0,
 *                          getenv(NOGETTEXT("PATH")));
 */
extern int
find_run_directory (char        *cmd,
                    char        *cwd,
                    char        *dir,
                    char        **pgm,
                    char        **run,
                    char        *path)
{
    int                 rv = 0;
    char                *f, *s;
    int                 i;
    char                tmp_path[MAXPATHLEN];

    if (!cmd || !*cmd || !cwd || !dir) {
        errno = EINVAL;         /* stupid arguments! */
        return -1;
    }

    if (*cwd != '/')
        if (!(getcwd (cwd, MAXPATHLEN)))
            return -1;          /* can not get working directory */

    f = strrchr (cmd, '/');
    if (pgm)                    /* user wants program name */
        *pgm = f ? f + 1 : cmd;

    /* get program directory */
    rv = -1;
    if (*cmd == '/')    /* absname given */
        rv = resolve ("", cmd + 1, dir, run);
    else if (f)         /* relname given */
        rv = resolve (cwd, cmd, dir, run);
    else {  /* from searchpath */
            if (!path || !*path) {      /* if missing or null path */
                    tmp_path[0] = '.';  /* assume sanity */
                    tmp_path[1] = '\0';
            } else {
                    strcpy(tmp_path, path);
            }
        f = tmp_path;
        rv = -1;
        errno = ENOENT; /* errno gets this if path empty */
        while (*f && (rv < 0)) {
            s = f;
            while (*f && (*f != ':'))
                ++f;
            if (*f)
                *f++ = 0;
            if (*s == '/')
                rv = resolve (s, cmd, dir, run);
            else {
                char                abuf[MAXPATHLEN];

                sprintf (abuf, "%s/%s", cwd, s);
                rv = resolve (abuf, cmd, dir, run);
            }
        }
    }
```

```

    /* Remove any trailing /. */
    i = strlen(dir);
    if ( dir[i-2] == '/' && dir[i-1] == '.') {
            dir[i-2] = '\0';
    }

    return rv;
}
```