```
**********************************************************
    5071 Mon Mar 30 10:49:09 2015
new/usr/src/pkg/manifests/system-test-libctest.mf
Incorporate rmustacc's review feedback.
**********************************************************
```

```
   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 #
  13 # Copyright (c) 2012 by Delphix. All rights reserved.
  14 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
  15 # Copyright 2015 Garrett D'Amore <garrett@damore.org>
  16 #

  18 set name=pkg.fmri value=pkg:/system/test/libctest@$(PKGVERS)
  19 set name=pkg.description value="C library Unit Tests"
  20 set name=pkg.summary value="C Library Unit Test Suite"
  21 set name=info.classification \
  22     value=org.opensolaris.category.2008:Development/System
  23 set name=variant.arch value=$(ARCH)
  24 dir path=opt/libc-tests
  25 dir path=opt/libc-tests/bin
  26 dir path=opt/libc-tests/cfg
  27 dir path=opt/libc-tests/cfg/symbols
  28 dir path=opt/libc-tests/runfiles
  29 dir path=opt/libc-tests/tests
  30 dir path=opt/libc-tests/tests
  31 dir path=opt/libc-tests/tests/symbols
  32 file path=opt/libc-tests/README mode=0444
  33 file path=opt/libc-tests/bin/libctest mode=0555
  34 file path=opt/libc-tests/cfg/README mode=0444
  35 file path=opt/libc-tests/cfg/compilation.cfg mode=0444
  36 file path=opt/libc-tests/cfg/symbols/README mode=0444
  37 file path=opt/libc-tests/cfg/symbols/ctype_h.cfg mode=0444
  38 file path=opt/libc-tests/cfg/symbols/dirent_h.cfg mode=0444
  39 file path=opt/libc-tests/cfg/symbols/fcntl_h.cfg mode=0444
  40 file path=opt/libc-tests/cfg/symbols/locale_h.cfg mode=0444
  41 file path=opt/libc-tests/cfg/symbols/math_h.cfg mode=0444
  42 file path=opt/libc-tests/cfg/symbols/netdb_h.cfg mode=0444
  43 file path=opt/libc-tests/cfg/symbols/pthread_h.cfg mode=0444
  44 file path=opt/libc-tests/cfg/symbols/signal_h.cfg mode=0444
  45 file path=opt/libc-tests/cfg/symbols/stdio_h.cfg mode=0444
  46 file path=opt/libc-tests/cfg/symbols/stdlib_h.cfg mode=0444
  47 file path=opt/libc-tests/cfg/symbols/strings_h.cfg mode=0444
  48 file path=opt/libc-tests/cfg/symbols/sys_stat_h.cfg mode=0444
  49 file path=opt/libc-tests/cfg/symbols/sys_time_h.cfg mode=0444
  50 file path=opt/libc-tests/cfg/symbols/sys_timeb_h.cfg mode=0444
  51 file path=opt/libc-tests/cfg/symbols/ucontext_h.cfg mode=0444
  52 file path=opt/libc-tests/cfg/symbols/unistd_h.cfg mode=0444
  53 file path=opt/libc-tests/cfg/symbols/wchar_h.cfg mode=0444
  54 file path=opt/libc-tests/cfg/symbols/wctype_h.cfg mode=0444
  55 file path=opt/libc-tests/runfiles/default.run mode=0444
  56 file path=opt/libc-tests/tests/fpround_test mode=0555
  57 file path=opt/libc-tests/tests/fpround_test.$(ARCH) mode=0555
  58 file path=opt/libc-tests/tests/fpround_test.$(ARCH64) mode=0555
  59 file path=opt/libc-tests/tests/newlocale_test mode=0555
  60 file path=opt/libc-tests/tests/newlocale_test.$(ARCH) mode=0555
  61 file path=opt/libc-tests/tests/newlocale_test.$(ARCH64) mode=0555
```



```
  62 file path=opt/libc-tests/tests/nl_langinfo_test mode=0555
  63 file path=opt/libc-tests/tests/nl_langinfo_test.$(ARCH) mode=0555
  64 file path=opt/libc-tests/tests/nl_langinfo_test.$(ARCH64) mode=0555
  65 file path=opt/libc-tests/tests/symbols/setup mode=0555
  66 file path=opt/libc-tests/tests/symbols/symbols_test.$(ARCH) mode=0555
  67 file path=opt/libc-tests/tests/symbols/symbols_test.$(ARCH64) mode=0555
  68 file path=opt/libc-tests/tests/wcsrtombs_test mode=0555
  69 file path=opt/libc-tests/tests/wcsrtombs_test.$(ARCH) mode=0555
  70 file path=opt/libc-tests/tests/wcsrtombs_test.$(ARCH64) mode=0555
  71 file path=opt/libc-tests/tests/wctype_test mode=0555
  72 file path=opt/libc-tests/tests/wctype_test.$(ARCH) mode=0555
  73 file path=opt/libc-tests/tests/wctype_test.$(ARCH64) mode=0555
  74 hardlink path=opt/libc-tests/tests/symbols/ctype_h target=setup
  75 hardlink path=opt/libc-tests/tests/symbols/dirent_h target=setup
  76 hardlink path=opt/libc-tests/tests/symbols/fcntl_h target=setup
  77 hardlink path=opt/libc-tests/tests/symbols/locale_h target=setup
  78 hardlink path=opt/libc-tests/tests/symbols/math_h target=setup
  79 hardlink path=opt/libc-tests/tests/symbols/netdb_h target=setup
  80 hardlink path=opt/libc-tests/tests/symbols/pthread_h target=setup
  81 hardlink path=opt/libc-tests/tests/symbols/signal_h target=setup
  82 hardlink path=opt/libc-tests/tests/symbols/stdio_h target=setup
  83 hardlink path=opt/libc-tests/tests/symbols/stdlib_h target=setup
  84 hardlink path=opt/libc-tests/tests/symbols/strings_h target=setup
  85 hardlink path=opt/libc-tests/tests/symbols/sys_stat_h target=setup
  86 hardlink path=opt/libc-tests/tests/symbols/sys_time_h target=setup
  87 hardlink path=opt/libc-tests/tests/symbols/sys_timeb_h target=setup
  88 hardlink path=opt/libc-tests/tests/symbols/ucontext_h target=setup
  89 hardlink path=opt/libc-tests/tests/symbols/unistd_h target=setup
  90 hardlink path=opt/libc-tests/tests/symbols/wchar_h target=setup
  91 hardlink path=opt/libc-tests/tests/symbols/wctype_h target=setup
  92 license lic_CDDL license=lic_CDDL
  93 depend fmri=locale/de type=require
  94 depend fmri=locale/en type=require
  95 depend fmri=locale/en-extra type=require
  96 depend fmri=locale/ja type=require
  97 depend fmri=locale/ru type=require
  98 depend fmri=system/test/testrunner type=require
```

```
**********************************************************
    1431 Mon Mar 30 10:49:09 2015
new/usr/src/test/libc-tests/cfg/Makefile
Incorporate rmustacc's review feedback.
**********************************************************
```

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

#
# Copyright 2015 Garrett D'Amore <garrett@damore.org>
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
# Copyright (c) 2012 by Delphix. All rights reserved.
#

include $(SRC)/Makefile.master

CFGS = README \
        compilation.cfg \
        symbols/README \
        symbols/ctype_h.cfg \
        symbols/dirent_h.cfg \
        symbols/fcntl_h.cfg \
        symbols/locale_h.cfg \
        symbols/math_h.cfg \
        symbols/netdb_h.cfg \
        symbols/pthread_h.cfg \
        symbols/signal_h.cfg \
        symbols/stdio_h.cfg \
        symbols/stdlib_h.cfg \
        symbols/strings_h.cfg \
        symbols/sys_stat_h.cfg \
        symbols/sys_time_h.cfg \
        symbols/sys_timeb_h.cfg \
        symbols/ucontext_h.cfg \
        symbols/unistd_h.cfg \
        symbols/wchar_h.cfg \
        symbols/wctype_h.cfg

ROOTOPTPKG = $(ROOT)/opt/libc-tests
ROOTOPTPKGCFG = $(ROOT)/opt/libc-tests/cfg
ROOTOPTPKGDIRS = $(ROOTOPTPKG) \
        $(ROOTOPTPKGCFG) \
        $(ROOTOPTPKGCFG)/symbols

FILES = $(CFGS:%=$(ROOTOPTPKGCFG)/%)
$(FILES) := FILEMODE = 0444

all: $(CFGS)

install: $(ROOTOPTPKG) $(ROOTOPTPKGCFG) $(FILES)

clean lint clobber:

$(ROOTOPTPKGDIRS):
        $(INS.dir)

$(ROOTOPTPKGCFG)/%: % $(ROOTOPTPKGDIRS)
        $(INS.file)
```

```
**********************************************************
    3377 Mon Mar 30 10:49:09 2015
new/usr/src/test/libc-tests/cfg/README
Incorporate rmustacc's review feedback.
**********************************************************
```

```
   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 #
  13 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  14 #

  16 The configuration files in this directory are structured as lines,
  17 where each line is made up of fields, separated by "|" characters,
  18 possibly surrounded by whitespace.

  20 New lines preceeded by backslashes are ignored, allowing for a continuation
  21 of lines, in the usual UNIX way.

  23 A line beginning with a hashmark is a comment, and is ignored, as are lines
  23 A line beginning with a hashmark is comment, and is ignore, as are lines
  24 consisting solely of whitespace.

  26 The first field is always the "keyword", which determines the meaning and
  27 presence of any other fields.

  29 These files are parsed using the test_load_config() function.  This
  30 function has the following prototype:

  32         int test_load_config(test_t, const char *, ...);

  34 The variable arguments are the keywords and handling functions.  These
  35 must be supplied in pairs and the list is terminated with a NULL, like this:

  37         test_config_load(t, "myfile.cfg", "mykeyword", keywordcb, NULL);

  39 The test_config_load function will search for the named file (provided it
  40 is not an absolute path) in a few locations:

  42         * relative to the current directory, exactly as specified
  43         * relative to $STF_SUITE/cfg/   (if $STF_SUITE is defined)
  44         * relative to ../../cfg/        (if $STF_SUITE is undefined)
  45         * relative to cfg/

  47 The handling functions (keywordcb in the example above) have the following
  48 typedef:

  50         typedef int (*test_cfg_func_t)(char **fields, int nfields, char **err);

  52 so for example, keywordcb should be declared thusly:

  54         int keywordcb(char **fields, int nfields, char **err);

  56 These functions are called each time a paired keyword is seen in the file.
  57 "fields" is an array of fields, pre-split with surrounding whitespace removed,
  58 and contains "nfields" items.  Internal whitespace is unaffected.

  60 The function should return 0 on successful handling, or -1 on failure.  In
```



```
  61 the event of failure, it should record an error string in "err" using
  62 asprintf() or strdup(). ("err" should be unmodified otherwise.)

  64 This parser is rather simplistic, and it lacks support for embedding "|"
  65 fields in lines, and also doesn't support escaping, so you can't add "\"
  66 at the end of a line (if you need that, leave some trailing whitespace).

  68 There are also some internal limits on the length of lines (1K), and on the
  69 number of fields (20).  As this is only used for these test suites, this
  70 should not be a significant limitation.

  72 Please see ../tests/symbols/symbols_test.c for an example of correct usage.

  74 Aside:

  76   Astute readers may ask why invent a new configuration file, and why use
  77   position based parsing instead of name value pairs.  These files are
  78   optimized for specific needs, and intended to support relatively dense
  79   information in a format that is easy for humans to work with.  JSON or XML
  80   or even YAML could have served, but the overhead of a syntax was more than
  81   we wanted to introduce.  Test suites are free to use other formats if they
  82   choose, but this simple format has the advantage of being built-in and
  83   easy to use.
```

```
**********************************************************
    2850 Mon Mar 30 10:49:09 2015
new/usr/src/test/libc-tests/cfg/compilation.cfg
Incorporate rmustacc's review feedback.
**********************************************************
```

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

#
# Copyright 2015 Garrett D'Amore <garrett@damore.org>
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

#
# Compilation environments.
#
# Each compilation environment is declared using the keyword "env", like
# this:
#
# env | <name> | <std> | <defs>
#
# <name> is just a symbolic name for environment.
# <std>  indicates either c89 or c99, i.e. which C standard to compile
#        under.  This infuences choice of compiler and switches.
# <defs> is a list of CPP style -D or -U flags to define C preprocessor
#        symbols.
#
# Groups of compilation environments can be named, using the "env_group"
# keyword (this can also be used to create aliases):
#
# env_group | <name> | <envs>
#
# <name> is a name for the group or alias
# <envs> is a whitespace separated list of previously declared environments
#        or environment groups (or aliases).
#

env | XPG3              | c89 | -D_XOPEN_SOURCE
env | XPG4              | c89 | -D_XOPEN_SOURCE -D_XOPEN_VERSION=4
env | SUSv1             | c89 | -D_XOPEN_SOURCE -D_XOPEN_SOURCE_EXTENDED=1
env | SUSv2             | c89 | -D_XOPEN_SOURCE=500
env | SUSv3             | c99 | -D_XOPEN_SOURCE=600
env | SUSv4             | c99 | -D_XOPEN_SOURCE=700
env | POSIX-1990        | c89 | -D_POSIX_SOURCE
env | POSIX-1992        | c89 | -D_POSIX_SOURCE -D_POSIX_C_SOURCE=2
env | POSIX-1993        | c89 | -D_POSIX_C_SOURCE=199309L
env | POSIX-1995        | c89 | -D_POSIX_C_SOURCE=199506L
env | POSIX-2001        | c99 | -D_POSIX_C_SOURCE=200112L
env | POSIX-2008        | c99 | -D_POSIX_C_SOURCE=200809L
env | C90               | c89 |
env | C99               | c99 |

#
# These are ordered from less inclusive (most recent) to most inclusive.
# This allows for us to "include" by reference.
#
env_group | POSIX-2008+ | POSIX-2008
env_group | POSIX-2001+ | POSIX-2008+ POSIX-2001
```



```
env_group | POSIX-1995+ | POSIX-2001+ POSIX-1995
env_group | POSIX-1993+ | POSIX-1995+ POSIX-1993
env_group | POSIX-1992+ | POSIX-1993+ POSIX-1992
env_group | POSIX-1990+ | POSIX-1992+ POSIX-1990
env_group | POSIX+      | POSIX-1990+
env_group | SUSv4+      | SUSv4 POSIX-2008+
env_group | SUSv3+      | SUSv3 SUSv4+ POSIX-2001+
env_group | SUSv2+      | SUSv2 SUSv3+
env_group | SUSv1+      | SUSv1 SUSv2+
env_group | SUS+        | SUSv1+
env_group | XPG4+       | XPG4 SUSv1+
env_group | XPG3+       | XPG3 XPG4+
env_group | C99+        | C99 POSIX-2001+ SUSv3+
env_group | C+          | C90 C99 POSIX+ SUS+
env_group | ALL         | C+

#
# Aliases.
#
env_group | XPG4v2      | SUSv1
env_group | XPG4v2+     | SUSv1+
env_group | XPG5        | SUSv2
env_group | XPG5+       | SUSv2+
env_group | XPG6        | SUSv3
env_group | XPG6+       | SUSv3+
env_group | XPG7        | SUSv4
env_group | XPG7+       | SUSv4+
```

```
**********************************************************
    3049 Mon Mar 30 10:49:09 2015
new/usr/src/test/libc-tests/cfg/symbols/README
Incorporate rmustacc's review feedback.
**********************************************************
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2015 Garrett D'Amore <garrett@damore.org>
#

The configuration files in this directory are structured using the
syntax defined in the ../README file.  They make use of the compilation
environments declared in ../compilation.cfg, and are processed by the
symbols test.

We have organized the files by header file, that is the tests for symbols
declared in a header file (e.g. <unistd.h> appear in a file based on that
header file's name (e.g. unistd_h.cfg.)  This is purely for convenience.

Within these various declarations, we have the following field types:

<envs>    This is a list of compilation environments where the symbol
          should be legal.  To indicate that the symbol must not be legal
          an environment group can be prefixed with "-".  For example,
          "SUS -SUSv4+" indicates a symbol that is legal in all SUS
          environments up to SUSv3, and was removed in SUSv4 and subsequent
          versions of SUS.  As you can see, we can list multiple environments
          or environment groups, and we can add or remove to previous groups
          with subsequent ones.

<name>    This is a symbol name.  It follows the rules for C symbol names.

<header>  This is a header file, for example, unistd.h.  Conventionally,
          the header files used should match the file where the test is
          declared.

<type>    This is a C type.  Function types can be declared without their
          names, e.g. "void (*)(int)".  Structures (e.g. "struct stat") and
          pointer types (e.g. "pthead_t *") are legal as well.

Here are the types of declarations in these files:

type | <name> | <header> | <envs>

    Tests for a C type with <name>.  The test verifies that a variable with
    this type can be declared when the <header> is included.

value | <name> | <type> | <header> | <envs>

    Tests for a value named <name>, of type <type>.  The test attempts to
    assign the given value to a scratch variable declared with the given
    type.  The value can be a macro or other C symbol.

func | <name> | <type> | <type> [; <type> ]... | <header> | <envs>

    Tests whether a function <name>, returning the first <type>, and
```



```
    taking arguments of following <type> values, is declared.  Note that
    the argument types are separated by semicolons.  For varargs style
    functions, leave out the ... part.  For function declarations
    that have no declared arguments, either void can specified, or
    the type list can be omitted.

Examples:

    type | size_t | sys/types.h | ALL
    value | NULL | void * | stdlib.h | ALL
    func strnlen | int | const char *; int | string.h | ALL
```

```
**********************************************************
    7523 Mon Mar 30 10:49:09 2015
new/usr/src/test/libc-tests/tests/common/test_common.c
Incorporate rmustacc's review feedback.
**********************************************************
```

```
   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 /*
  13  * Copyright 2015 Garrett D'Amore <garrett@damore.org>
  13  * Copyright 2014 Garrett D'Amore <garrett@damore.org>
  14  */

  16 /*
  17  * Common handling for test programs.
  18  */

  20 #include <stdio.h>
  21 #include <stdlib.h>
  22 #include <stdarg.h>
  23 #include <string.h>
  24 #include <errno.h>
  25 #include <pthread.h>
  26 #include <ctype.h>
  27 #include <unistd.h>
  28 #include <sys/param.h>
  29 #include "test_common.h"

  31 static int debug = 0;
  32 static int force = 0;
  33 static pthread_mutex_t lk;

  35 static int passes;
  36 static int tests;

  38 struct test {
  39         char            *name;
  40         int             ntids;
  41         pthread_t       *tids;
  42         int             fails;
  43         void            *arg;
  44         void            (*func)(test_t t, void *);
  45 };
_____unchanged_portion_omitted_

 175 void
 176 test_debugf(test_t t, const char *format, ...)
 177 {
 178         va_list args;

 180         if (!debug)
 181                 return;

 183         (void) pthread_mutex_lock(&lk);
 184         if (t) {
 185                 (void) printf("TEST DEBUG %s: ", t->name);
 186         } else {
 187                 (void) printf("TEST DEBUG: ");
```



```
 188         }
 189         va_start(args, format);
 190         (void) vprintf(format, args);
 191         va_end(args);
 192         (void) printf("\n");
 193         (void) fflush(stdout);
 194         (void) pthread_mutex_unlock(&lk);
 195 }
_____unchanged_portion_omitted_

 260 #define MAXCB           20
 261 #define MAXFIELD        20

 263 int
 264 test_load_config(test_t t, const char *fname, ...)
 265 {
 266         va_list         va;
 267         const char      *keyws[MAXCB];
 268         test_cfg_func_t callbs[MAXCB];
 269         char            *fields[MAXFIELD];
 270         int             nfields;

 272         FILE            *cfg;
 273         char            line[1024];
 274         char            buf[1024];
 275         int             done;
 276         char            *ptr;
 277         char            *tok;
 278         char            *err;
 279         int             lineno;
 280         int             rv;
 281         int             found;
 282         char            path[MAXPATHLEN];
 283         int             i;

 285         va_start(va, fname);
 286         for (i = 0; i < MAXCB; i++) {
 284         for (int i = 0; i < MAXCB; i++) {
 287                 keyws[i] = (const char *)va_arg(va, const char *);
 288                 if (keyws[i] == NULL)
 289                         break;
 290                 callbs[i] = (test_cfg_func_t)va_arg(va, test_cfg_func_t);
 291         }
 292         va_end(va);
 293         if (i == MAXCB) {
 294                 test_debugf(t, "too many arguments to function >= %d", MAXCB);
 295         }

 297         found = 0;

 299         if (access(fname, F_OK) == 0) {
 300                 found++;
 301         }
 302         if (!found && fname[0] != '/') {
 303                 char *stf = getenv("STF_SUITE");
 304                 if (stf == NULL) {
 305                         stf = "../..";
 306                 }
 307                 (void) snprintf(path, sizeof (path), "%s/cfg/%s", stf, fname);
 308                 if (access(path, F_OK) == 0) {
 309                         fname = path;
 310                         found++;
 311                 } else {
 312                         (void) snprintf(path, sizeof (path), "cfg/%s", fname);
 313                         if (access(path, F_OK) == 0) {
 314                                 fname = path;
```

```
315                                 found++;
316                         }
317                 }
318         }

320         if ((cfg = fopen(fname, "r")) ==  NULL) {
321                 test_failed(t, "open(%s): %s", fname, strerror(errno));
322                 return (-1);
323         }

325         line[0] = 0;
326         done = 0;
327         lineno = 0;

329         while (!done) {

331                 lineno++;

333                 if (fgets(buf, sizeof (buf), cfg) == NULL) {
334                         done++;
335                 } else {
336                         (void) strtok(buf, "\n");
337                         if ((*buf != 0) && (buf[strlen(buf)-1] == '\\')) {
338                                 /*
339                                  * Continuation.  This isn't quite right,
340                                  * as it doesn't allow for a "\" at the
341                                  * end of line (no escaping).
342                                  */
343                                 buf[strlen(buf)-1] = 0;
344                                 (void) strlcat(line, buf, sizeof (line));
345                                 continue;
346                         }
347                         (void) strlcat(line, buf, sizeof (line));
348                 }

350                 /* got a line */
351                 ptr = line;
352                 test_trim(&ptr);

354                 /* skip comments and empty lines */
355                 if (ptr[0] == 0 || ptr[0] == '#') {
356                         line[0] = 0;
357                         continue;
358                 }

360                 tok = strsep(&ptr, "|");
361                 if (tok == NULL) {
362                         break;
363                 }
364                 test_trim(&tok);

366                 for (nfields = 0; nfields < MAXFIELD; nfields++) {
367                         fields[nfields] = strsep(&ptr, "|");
368                         if (fields[nfields] == NULL) {
369                                 break;
370                         }
371                         test_trim(&fields[nfields]);
372                 }

374                 found = 0;
375                 rv = 0;

377                 for (int i = 0; keyws[i] != NULL; i++) {
378                         if (strcmp(tok, keyws[i]) == 0) {
379                                 found++;
380                                 err = NULL;
```



```
381                                 rv = callbs[i](fields, nfields, &err);
382                         }
383                 }
384                 if (!found) {
385                         rv = -1;
386                         err = NULL;
387                         (void) asprintf(&err, "unknown keyword %s", tok);
388                 }
389                 if (rv != 0) {
390                         if (err) {
391                                 test_failed(t, "%s:%d: %s", fname,
392                                     lineno, err);
393                                 free(err);
394                         } else {
395                                 test_failed(t, "%s:%d: unknown error",
396                                     fname, lineno);
397                         }
398                         (void) fclose(cfg);
399                         return (rv);
400                 }

402                 line[0] = 0;
403         }
404         (void) fclose(cfg);
405         return (0);
406 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   17808 Mon Mar 30 10:49:09 2015
new/usr/src/test/libc-tests/tests/symbols/symbols_test.c
Incorporate rmustacc's review feedback.
**********************************************************
```

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

/*
 * Copyright 2015 Garrett D'Amore <garrett@damore.org>
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 */

/*
 * This program tests symbol visibility using the /usr/bin/c89 and
 * /usr/bin/c99 programs.
 *
 * See symbols_defs.c for the actual list of symbols tested.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <err.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <note.h>
#include <sys/wait.h>
#include "test_common.h"

char *dname;
char *cfile;
char *ofile;
char *lfile;
char *efile;

const char *sym = NULL;

static int good_count = 0;
static int fail_count = 0;
static int full_count = 0;
static int extra_debug = 0;
static char *compilation = "compilation.cfg";

#if defined(_LP64)
#define MFLAG "-m64"
#elif defined(_ILP32)
#define MFLAG "-m32"
#endif

const char *compilers[] = {
        "cc",
        "gcc",
        "/opt/SUNWspro/bin/cc",
        "/opt/gcc/4.4.4/bin/gcc",
        "/opt/sunstudio12.1/bin/cc",
```



```
        "/opt/sfw/bin/gcc",
        "/usr/local/bin/gcc",
        NULL
};

const char *puname[] = {
        "",
        "/usr/bin/puname -S "
};

char *compiler = NULL;
const char *c89flags = NULL;
const char *c99flags = NULL;

#define MAXENV  64      /* maximum number of environments (bitmask width) */
/* ====== BEGIN ======== */

#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdint.h>

#define MAXENV  64      /* bits */
#define MAXHDR  10      /* maximum # headers to require to access symbol */
#define MAXARG  20      /* maximum # of arguments */

#define WS      " \t"

static int next_env = 0;

struct compile_env {
        char            *name;
        char            *lang;
        char            *defs;
        int             index;
};
```

_____*unchanged_portion_omitted_*

```
struct env_group *env_groups = NULL;

struct sym_test *sym_tests = NULL;
struct sym_test **sym_insert = &sym_tests;

static char *
mystrdup(const char *s)
{
        char *r;
        if ((r = strdup(s)) == NULL) {
                perror("strdup");
                exit(1);
        }
        return (r);
}

static void *
myzalloc(size_t sz)
{
        void *buf;
        if ((buf = calloc(1, sz)) == NULL) {
                perror("calloc");
                exit(1);
        }
        return (buf);
}
```

```
132 static void
133 myasprintf(char **buf, const char *fmt, ...)
134 {
135         int rv;
136         va_list va;
137         va_start(va, fmt);
138         rv = vasprintf(buf, fmt, va);
139         va_end(va);
140         if (rv < 0) {
141                 perror("vasprintf");
142                 exit(1);
143         }
144 }

146 static void
147 append_sym_test(struct sym_test *st)
148 {
149         *sym_insert = st;
150         sym_insert = &st->next;
151 }
```

*unchanged_portion_omitted_*

```
227 static int
228 do_env(char **fields, int nfields, char **err)
229 {
230         char *name;
231         char *lang;
232         char *defs;

234         if (nfields != 3) {
235                 myasprintf(err, "number of fields (%d) != 3", nfields);
214                 (void) asprintf(err, "number of fields (%d) != 3", nfields);
236                 return (-1);
237         }

239         if (next_env >= MAXENV) {
240                 myasprintf(err, "too many environments");
219                 (void) asprintf(err, "too many environments");
241                 return (-1);
242         }

244         name = fields[0];
245         lang = fields[1];
246         defs = fields[2];

248         compile_env[next_env].name = mystrdup(name);
249         compile_env[next_env].lang = mystrdup(lang);
250         compile_env[next_env].defs = mystrdup(defs);
227         compile_env[next_env].name = strdup(name);
228         compile_env[next_env].lang = strdup(lang);
229         compile_env[next_env].defs = strdup(defs);
251         compile_env[next_env].index = next_env;
252         next_env++;
253         return (0);
254 }

256 static int
257 do_env_group(char **fields, int nfields, char **err)
258 {
259         char *name;
260         char *list;
261         struct env_group *eg;
262         uint64_t mask;
263         char *item;
```



```
265         if (nfields != 2) {
266                 myasprintf(err, "number of fields (%d) != 2", nfields);
245                 (void) asprintf(err, "number of fields (%d) != 2", nfields);
267                 return (-1);
268         }

270         name = fields[0];
271         list = fields[1];
272         mask = 0;

274         if (expand_env(list, &mask, &item) < 0) {
275                 myasprintf(err, "reference to undefined env %s", item);
254                 (void) asprintf(err, "reference to undefined env %s", item);
276                 return (-1);
277         }

279         eg = myzalloc(sizeof (*eg));
280         eg->name = mystrdup(name);
258         eg = calloc(1, sizeof (*eg));
259         eg->name = strdup(name);
281         eg->mask = mask;
282         eg->next = env_groups;
283         env_groups = eg;
284         return (0);
285 }

287 static char *progbuf = NULL;
288 size_t proglen = 0;
289 size_t progsiz = 0;

291 static void
292 addprogch(char c)
293 {
294         while (progsiz <= (proglen + 1)) {
295                 progbuf = realloc(progbuf, progsiz + 4096);
296                 if (progbuf == NULL) {
297                         perror("realloc");
298                         exit(1);
299                 }
300                 progsiz += 1024;
301         }
302         progbuf[proglen++] = c;
303         progbuf[proglen] = 0;
304 }

306 static void
307 addprogstr(char *s)
308 {
309         while (*s != NULL) {
310                 addprogch(*s);
311                 s++;
312         }
313 }

315 static void
316 addprogfmt(const char *fmt, ...)
317 {
318         va_list va;
319         char *buf = NULL;
320         va_start(va, fmt);
321         if (vasprintf(&buf, fmt, va) < 0) {
322                 perror("vasprintf");
323                 exit(1);
324         }
325         va_end(va);
326         addprogstr(buf);
```

```
327         free(buf);
328 }

330 static void
331 mkprog(struct sym_test *st)
332 {
269         static char buf[2048];
333         char *s;
271         char *prog = buf;

335         proglen = 0;
273         *prog = 0;

275 #define ADDSTR(p, str)  (void) strcpy(p, str); p += strlen(p)
276 #define ADDFMT(p, ...)  \
277         (void) snprintf(p, sizeof (buf) - (p-buf), __VA_ARGS__); \
278         p += strlen(p)
279 #define ADDCHR(p, c)    *p++ = c; *p = 0

337         for (int i = 0; i < MAXHDR && st->hdrs[i] != NULL; i++) {
338                 addprogfmt("#include <%s>\n", st->hdrs[i]);
282                 ADDFMT(prog, "#include <%s>\n", st->hdrs[i]);
339         }

341         for (s = st->rtype; *s; s++) {
342                 addprogch(*s);
286                 ADDCHR(prog, *s);
343                 if (*s == '(') {
344                         s++;
345                         addprogch(*s);
289                         ADDCHR(prog, *s);
346                         s++;
347                         break;
348                 }
349         }
350         addprogch(' ');
294         ADDCHR(prog, ' ');

352         /* for function pointers, s is closing suffix, otherwise empty */

354         switch (st->type) {
355         case SYM_TYPE:
356                 addprogstr("test_type;");
300                 ADDFMT(prog, "test_type;", st->rtype);
357                 break;

359         case SYM_VALUE:
360                 addprogfmt("test_value%s;\n", s);       /* s usually empty */
361                 addprogstr("void\ntest_func(void)\n{\n");
362                 addprogfmt("\ttest_value = %s;\n}", st->name);
304                 ADDFMT(prog, "test_value%s;\n", s);     /* s usually empty */
305                 ADDSTR(prog, "void\ntest_func(void)\n{\n");
306                 ADDFMT(prog, "\ttest_value = %s;\n}",
307                     st->name);
363                 break;

365         case SYM_FUNC:
366                 addprogstr("\ntest_func(");
311                 ADDSTR(prog, "\ntest_func(");
367                 for (int i = 0; st->atypes[i] != NULL && i < MAXARG; i++) {
368                         int didname = 0;
369                         if (i > 0) {
370                                 addprogstr(", ");
315                                 ADDSTR(prog, ", ");
371                         }
372                         if (strcmp(st->atypes[i], "void") == 0) {
```



```
373                                 didname = 1;
374                         }
375                         if (strcmp(st->atypes[i], "") == 0) {
376                                 didname = 1;
377                                 addprogstr("void");
322                                 ADDSTR(prog, "void");
378                         }

380                         /* print the argument list */
381                         for (char *a = st->atypes[i]; *a; a++) {
382                                 if (*a == '(' && a[1] == '*' && !didname) {
383                                         addprogfmt("(*a%d", i);
328                                         ADDFMT(prog, "(*a%d", i);
384                                         didname = 1;
385                                         a++;
386                                 } else if (*a == '[' && !didname) {
387                                         addprogfmt("a%d[", i);
332                                         ADDFMT(prog, "a%d[", i);
388                                         didname = 1;
389                                 } else {
390                                         addprogch(*a);
335                                         ADDCHR(prog, *a);
391                                 }
392                         }
393                         if (!didname) {
394                                 addprogfmt(" a%d", i);
339                                 ADDFMT(prog, " a%d", i);
395                         }
396                 }

398                 if (st->atypes[0] == NULL) {
399                         addprogstr("void");
344                         ADDSTR(prog, "void");
400                 }

402                 /* close argument list, and closing ")" for func ptrs */
403                 addprogfmt(")%s\n{\n\t", s);    /* NB: s is normally empty */
348                 ADDFMT(prog, ")%s\n{\n\t", s);  /* NB: s is normally empty */

405                 if (strcmp(st->rtype, "") != 0 &&
406                     strcmp(st->rtype, "void") != 0) {
407                         addprogstr("return ");
352                         ADDSTR(prog, "return ");
408                 }

410                 /* add the function call */
411                 addprogfmt("%s(", st->name);
356                 ADDFMT(prog, "%s(", st->name);
412                 for (int i = 0; st->atypes[i] != NULL && i < MAXARG; i++) {
413                         if (strcmp(st->atypes[i], "") != 0 &&
414                             strcmp(st->atypes[i], "void") != 0) {
415                                 addprogfmt("%sa%d", i > 0 ? ", " : "", i);
360                                 ADDFMT(prog, "%sa%d", i > 0 ? ", " : "", i);
416                         }
417                 }

419                 addprogstr(");\n}");
364                 ADDSTR(prog, ");\n}");
420                 break;
421         }

423         addprogch('\n');
368         ADDCHR(prog, '\n');

425         st->prog = progbuf;
370         st->prog = strdup(buf);
```

```
426 }

428 static int
429 add_envs(struct sym_test *st, char *envs, char **err)
430 {
431         char *item;
432         if (expand_env_list(envs, &st->test_mask, &st->need_mask, &item) < 0) {
433                 myasprintf(err, "bad env action %s", item);
378                 (void) asprintf(err, "bad env action %s", item);
434                 return (-1);
435         }
436         return (0);
437 }

439 static int
440 add_headers(struct sym_test *st, char *hdrs, char **err)
441 {
442         int i = 0;

444         for (char *h = strsep(&hdrs, ";"); h != NULL; h = strsep(&hdrs, ";")) {
445                 if (i >= MAXHDR) {
446                         myasprintf(err, "too many headers");
391                         (void) asprintf(err, "too many headers");
447                         return (-1);
448                 }
449                 test_trim(&h);
450                 st->hdrs[i++] = mystrdup(h);
395                 st->hdrs[i++] = strdup(h);
451         }

453         return (0);
454 }

456 static int
457 add_arg_types(struct sym_test *st, char *atype, char **err)
458 {
459         int i = 0;
460         char *a;
461         for (a = strsep(&atype, ";"); a != NULL; a = strsep(&atype, ";")) {
462                 if (i >= MAXARG) {
463                         myasprintf(err, "too many arguments");
408                         (void) asprintf(err, "too many arguments");
464                         return (-1);
465                 }
466                 test_trim(&a);
467                 st->atypes[i++] = mystrdup(a);
412                 st->atypes[i++] = strdup(a);
468         }

470         return (0);
471 }

473 static int
474 do_type(char **fields, int nfields, char **err)
475 {
476         char *decl;
477         char *hdrs;
478         char *envs;
479         struct sym_test *st;

481         if (nfields != 3) {
482                 myasprintf(err, "number of fields (%d) != 3", nfields);
427                 (void) asprintf(err, "number of fields (%d) != 3", nfields);
483                 return (-1);
484         }
485         decl = fields[0];
```



```
486         hdrs = fields[1];
487         envs = fields[2];

489         st = myzalloc(sizeof (*st));
434         st = calloc(1, sizeof (*st));
490         st->type = SYM_TYPE;
491         st->name = mystrdup(decl);
492         st->rtype = mystrdup(decl);
436         st->name = strdup(decl);
437         st->rtype = strdup(decl);

494         if ((add_envs(st, envs, err) < 0) ||
495             (add_headers(st, hdrs, err) < 0)) {
496                 return (-1);
497         }
498         append_sym_test(st);

500         return (0);
501 }

503 static int
504 do_value(char **fields, int nfields, char **err)
505 {
506         char *name;
507         char *type;
508         char *hdrs;
509         char *envs;
510         struct sym_test *st;

512         if (nfields != 4) {
513                 myasprintf(err, "number of fields (%d) != 4", nfields);
458                 (void) asprintf(err, "number of fields (%d) != 4", nfields);
514                 return (-1);
515         }
516         name = fields[0];
517         type = fields[1];
518         hdrs = fields[2];
519         envs = fields[3];

521         st = myzalloc(sizeof (*st));
466         st = calloc(1, sizeof (*st));
522         st->type = SYM_VALUE;
523         st->name = mystrdup(name);
524         st->rtype = mystrdup(type);
468         st->name = strdup(name);
469         st->rtype = strdup(type);

526         if ((add_envs(st, envs, err) < 0) ||
527             (add_headers(st, hdrs, err) < 0)) {
528                 return (-1);
529         }
530         append_sym_test(st);

532         return (0);
533 }

535 static int
536 do_func(char **fields, int nfields, char **err)
537 {
538         char *name;
539         char *rtype;
540         char *atype;
541         char *hdrs;
542         char *envs;
543         struct sym_test *st;
```

```
545         if (nfields != 5) {
546                 myasprintf(err, "number of fields (%d) != 5", nfields);
491                 (void) asprintf(err, "number of fields (%d) != 5", nfields);
547                 return (-1);
548         }
549         name = fields[0];
550         rtype = fields[1];
551         atype = fields[2];
552         hdrs = fields[3];
553         envs = fields[4];

555         st = myzalloc(sizeof (*st));
500         st = calloc(1, sizeof (*st));
556         st->type = SYM_FUNC;
557         st->name = mystrdup(name);
558         st->rtype = mystrdup(rtype);
502         st->name = strdup(name);
503         st->rtype = strdup(rtype);

560         if ((add_envs(st, envs, err) < 0) ||
561             (add_headers(st, hdrs, err) < 0) ||
562             (add_arg_types(st, atype, err) < 0)) {
563                 return (-1);
564         }
565         append_sym_test(st);

567         return (0);
568 }
_____unchanged_portion_omitted_

591 /*
592  * Iterate through tests.  Pass in NULL for cenv to begin the iteration. For
593  * subsequent iterations, use the return value from the previous iteration.
594  * Returns NULL when there are no more environments.
537  * Iterate through tests.  Pass NULL for cenv first time, and previous result
538  * the next.  Returns NULL when no more environments.
595  */
596 struct compile_env *
597 sym_test_env(struct sym_test *st, struct compile_env *cenv, int *need)
598 {
599         int i = cenv ? cenv->index + 1: 0;
600         uint64_t b = 1ULL << i;

602         while ((i < MAXENV) && (b != 0)) {
603                 cenv = &compile_env[i];
604                 if (b & st->test_mask) {
605                         *need = (st->need_mask & b) ? 1 : 0;
606                         return (cenv);
607                 }
608                 b <<= 1;
609                 i++;
610         }
611         return (NULL);
612 }
_____unchanged_portion_omitted_

632 static void
633 show_file(test_t t, const char *path)
634 {
635         FILE *f;
636         char *buf = NULL;
637         size_t cap = 0;
638         int line = 1;

640         f = fopen(path, "r");
641         if (f == NULL) {
```



```
642                 test_debugf(t, "fopen(%s): %s", path, strerror(errno));
643                 return;
644         }

646         test_debugf(t, "----->> begin (%s) <<------", path);
647         while (getline(&buf, &cap, f) >= 0) {
648                 (void) strtok(buf, "\r\n");
649                 test_debugf(t, "%d: %s", line, buf);
650                 line++;
651         }
652         test_debugf(t, "----->> end (%s) <<------", path);
653         (void) fclose(f);
654 }

656 static void
657 cleanup(void)
658 {
659         if (ofile != NULL) {
660                 (void) unlink(ofile);
661                 free(ofile);
662                 ofile = NULL;
663         }
664         if (lfile != NULL) {
665                 (void) unlink(lfile);
666                 free(lfile);
667                 lfile = NULL;
668         }
669         if (cfile != NULL) {
670                 (void) unlink(cfile);
671                 free(cfile);
672                 cfile = NULL;
673         }
674         if (efile != NULL) {
675                 (void) unlink(efile);
676                 free(efile);
677                 efile = NULL;
678         }
679         if (dname) {
680                 (void) rmdir(dname);
681                 free(dname);
682                 dname = NULL;
683         }
684 }

686 static int
687 mkworkdir(void)
688 {
689         char b[32];
690         char *d;

692         cleanup();

694         (void) strlcpy(b, "/tmp/symbols_testXXXXXX", sizeof (b));
695         if ((d = mkdtemp(b)) == NULL) {
696                 perror("mkdtemp");
697                 return (-1);
698         }
699         dname = mystrdup(d);
700         myasprintf(&cfile, "%s/compile_test.c", d);
701         myasprintf(&ofile, "%s/compile_test.o", d);
702         myasprintf(&lfile, "%s/compile_test.log", d);
703         myasprintf(&efile, "%s/compile_test.exe", d);
634         dname = strdup(d);
635         (void) asprintf(&cfile, "%s/compile_test.c", d);
636         (void) asprintf(&ofile, "%s/compile_test.o", d);
637         (void) asprintf(&lfile, "%s/compile_test.log", d);
```

```
704         return (0);
705 }

707 void
708 find_compiler(void)
709 {
710         test_t t;
711         int i;
712         FILE *cf;

714         t = test_start("finding compiler");

716         if ((cf = fopen(cfile, "w+")) == NULL) {
717                 test_failed(t, "Unable to open %s for write: %s", cfile,
718                     strerror(errno));
719                 return;
720         }
721         (void) fprintf(cf, "#include <stdio.h>\n");
722         (void) fprintf(cf, "int main(int argc, char **argv) {\n");
723         (void) fprintf(cf, "#if defined(__SUNPRO_C)\n");
724         (void) fprintf(cf, "exit(51);\n");
725         (void) fprintf(cf, "#elif defined(__GNUC__)\n");
726         (void) fprintf(cf, "exit(52);\n");
727         (void) fprintf(cf, "#else\n");
728         (void) fprintf(cf, "exit(99)\n");
729         (void) fprintf(cf, "#endif\n}\n");
730         (void) fclose(cf);

732         for (i = 0; compilers[i] != NULL; i++) {
664         for (i = 0; compilers[i/2] != NULL; i++) {
733                 char cmd[256];
734                 int rv;

736                 (void) snprintf(cmd, sizeof (cmd),
737                     "%s %s %s -o %s >/dev/null 2>&1",
738                     compilers[i], MFLAG, cfile, efile);
669                     "%s%s %s %s -o %s >/dev/null 2>&1",
670                     puname[i%2], compilers[i/2], MFLAG, cfile, ofile);
739                 test_debugf(t, "trying %s", cmd);
740                 rv = system(cmd);

742                 test_debugf(t, "result: %d", rv);

744                 if ((rv < 0) || !WIFEXITED(rv) || WEXITSTATUS(rv) != 0)
745                         continue;

747                 rv = system(efile);
679                 rv = system(ofile);
748                 if (rv >= 0 && WIFEXITED(rv)) {
749                         rv = WEXITSTATUS(rv);
750                 } else {
751                         rv = -1;
752                 }

754                 switch (rv) {
755                 case 51:        /* STUDIO */
756                         test_debugf(t, "Found Studio C");
757                         c89flags = "-Xc -errwarn=%all -v -xc99=%none " MFLAG;
758                         c99flags = "-Xc -errwarn=%all -v -xc99=%all " MFLAG;
759                         if (extra_debug) {
760                                 test_debugf(t, "c89flags: %s", c89flags);
761                                 test_debugf(t, "c99flags: %s", c99flags);
762                         }
763                         test_passed(t);
764                         break;
765                 case 52:        /* GCC */
```



```
766                         test_debugf(t, "Found GNU C");
767                         c89flags = "-Wall -Werror -std=c89 " MFLAG;
768                         c99flags = "-Wall -Werror -std=c99 " MFLAG;
769                         if (extra_debug) {
770                                 test_debugf(t, "c89flags: %s", c89flags);
771                                 test_debugf(t, "c99flags: %s", c99flags);
772                         }
773                         test_passed(t);
774                         break;
775                 default:
776                         continue;
777                 }
778                 myasprintf(&compiler, "%s", compilers[i]);
710                 (void) asprintf(&compiler,
711                     "%s%s", puname[i%2], compilers[i/2]);
779                 test_debugf(t, "compiler: %s", compiler);
780                 return;
781         }
782         test_failed(t, "No compiler found.");
783 }

785 int
786 do_compile(test_t t, struct sym_test *st, struct compile_env *cenv, int need)
787 {
788         char *cmd;
789         FILE *logf;
790         FILE *dotc;
791         const char *prog;

793         full_count++;

795         if ((dotc = fopen(cfile, "w+")) == NULL) {
796                 test_failed(t, "fopen(%s): %s", cfile, strerror(errno));
797                 return (-1);
798         }
799         prog = sym_test_prog(st);
800         if (fwrite(prog, 1, strlen(prog), dotc) < strlen(prog)) {
801                 test_failed(t, "fwrite: %s", strerror(errno));
802                 (void) fclose(dotc);
803                 return (-1);
804         }
805         if (fclose(dotc) < 0) {
806                 test_failed(t, "fclose: %s", strerror(errno));
807                 return (-1);
808         }

810         (void) unlink(ofile);

812         myasprintf(&cmd, "%s %s %s -c %s -o %s >>%s 2>&1",
745         if (asprintf(&cmd, "%s %s %s -c %s -o %s >>%s 2>&1",
813             compiler, strcmp(env_lang(cenv), "c99") == 0 ? c99flags : c89flags,
814             env_defs(cenv), cfile, ofile, lfile);
747             env_defs(cenv), cfile, ofile, lfile) < 0) {
748                 test_failed(t, "asprintf: %s", strerror(errno));
749                 return (-1);
750         }

816         if (extra_debug) {
817                 test_debugf(t, "command: %s", cmd);
818         }


820         if ((logf = fopen(lfile, "w+")) == NULL) {
821                 test_failed(t, "fopen: %s", strerror(errno));
822                 return (-1);
823         }
```

```
824         (void) fprintf(logf, "====================\n");
825         (void) fprintf(logf, "PROGRAM:\n%s\n", sym_test_prog(st));
826         (void) fprintf(logf, "COMMAND: %s\n", cmd);
827         (void) fprintf(logf, "EXPECT: %s\n", need ? "OK" : "FAIL");
828         (void) fclose(logf);

830         switch (system(cmd)) {
831         case -1:
832                 test_failed(t, "error compiling in %s: %s", env_name(cenv),
833                     strerror(errno));
834                 return (-1);
835         case 0:
836                 if (!need) {
767         if (system(cmd) != 0) {
768                 if (need) {
837                         fail_count++;
838                         show_file(t, lfile);
839                         test_failed(t, "symbol visible in %s", env_name(cenv));
771                         test_failed(t, "error compiling in %s", env_name(cenv));
840                         return (-1);
841                 }
842                 break;
843         default:
844                 if (need) {
774         } else {
775                 if (!need) {
845                         fail_count++;
846                         show_file(t, lfile);
847                         test_failed(t, "error compiling in %s", env_name(cenv));
778                         test_failed(t, "symbol visible in %s", env_name(cenv));
848                         return (-1);
849                 }
850                 break;
851         }
852         good_count++;
853         return (0);
854 }
_____unchanged_portion_omitted_

885 int
886 main(int argc, char **argv)
887 {
888         int optc;
889         int optC = 0;

891         while ((optc = getopt(argc, argv, "DdfCs:c:")) != EOF) {
892                 switch (optc) {
893                 case 'd':
894                         test_set_debug();
895                         break;
896                 case 'f':
897                         test_set_force();
898                         break;
899                 case 'D':
900                         test_set_debug();
901                         extra_debug++;
902                         break;
903                 case 'c':
904                         compilation = optarg;
905                         break;
906                 case 'C':
907                         optC++;
908                         break;
909                 case 's':
910                         sym = optarg;
911                         break;
```



```
912                 default:
913                         (void) fprintf(stderr, "Usage: %s [-df]\n", argv[0]);
914                         exit(1);
915                 }
916         }

918         if (test_load_config(NULL, compilation,
919             "env", do_env, "env_group", do_env_group, NULL) < 0) {
920                 exit(1);
921         }

923         while (optind < argc) {
924                 if (test_load_config(NULL, argv[optind++],
925                     "type", do_type,
926                     "value", do_value,
927                     "func", do_func,
928                     NULL) < 0) {
929                         exit(1);
930                 }
931         }

933         if (atexit(cleanup) != 0) {
934                 perror("atexit");
935                 exit(1);
936         }
863         (void) atexit(cleanup);

938         if (mkworkdir() < 0) {
939                 perror("mkdir");
940                 exit(1);
941         }

943         find_compiler();
944         if (!optC)
945                 test_compile();

947         exit(0);
948 }
_____unchanged_portion_omitted_
```