```
**********************************************************
    2010 Fri Dec 28 22:22:46 2018
new/usr/src/tools/cw/Makefile
Cleanup temp files without rm
**********************************************************
#
# 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 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright 2018 Joyent, Inc.

PROG    = cw

MAN1ONBLDFILES= cw.1onbld

include ../Makefile.tools

# Bootstrap problem -- we have to build cw before we can use it
i386_CC=                $(SPRO_VROOT)/bin/cc
sparc_CC=               $(SPRO_VROOT)/bin/cc
$(__GNUC)i386_CC=       $(GNUC_ROOT)/bin/gcc
$(__GNUC)sparc_CC=      $(GNUC_ROOT)/bin/gcc

CFLAGS += $(CCVERBOSE)

# Override CFLAGS.  This is needed only for bootstrap of cw.
$(__GNUC)CFLAGS=        -O -D__sun -Wall -Wno-unknown-pragmas -Werror \
                        -std=gnu99 -nodefaultlibs
$(__SUNC)CFLAGS=        -xspace -Xa -xildoff -errtags=yes -errwarn=%all \
                        -xc99=%all -W0,-xglobalstatic -v


$(__GNUC)LDLIBS +=      -lc
$(__GNUC)LDFLAGS=       $(MAPFILE.NES:%=-Wl,-M%)

$(ROOTONBLDMAN1ONBLDFILES) := FILEMODE=      644
CSTD=   $(CSTD_GNU99)
#endif /* ! codereview */

# Assume we don't have the install.bin available yet
INS.file=       $(RM) $@; $(CP) $< $(@D); $(CHMOD) $(FILEMODE) $@

.KEEP_STATE:

all: $(PROG) $(MAN1ONBLDFILES)
```

```
install: all .WAIT $(ROOTONBLDMACHPROG) $(ROOTONBLDMAN1ONBLDFILES)

lint: lint_PROG

clean:

#
# Not run by default: bootstrap...
check:
        $(ROOTONBLDBINMACH)/mandoc -Tlint -Wwarning $(MAN1ONBLDFILES)

include ../Makefile.targ
```

****
6241 Fri Dec 28 22:22:47 2018
new/usr/src/tools/cw/cw.1onbld
Revert "Revert most of "9899 cw(1onbld) should shadow more compilation""
This reverts commit 67deef8cbc83060db238a0f4ee252d1ba74641ef.
****

```
.\"
.\" 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 2010 Sun Microsystems, Inc.  All rights reserved.
.\" Use is subject to license terms.
.\"
.\" Copyright 2018 Joyent, Inc.
.\"
.Dd September 4, 2018
.Dt CW 1ONBLD
.Os
.Sh NAME
.Nm cw
.Nd invoke one or more compilers with argument translation
.Sh SYNOPSIS
.Nm cw
.Op Fl C
.Op Fl -versions
.Op Fl -noecho
.Fl -primary Ar compiler
.Op Fl -shadow Ar compiler ...
.Fl -
.Ar compiler args ...
.Sh DESCRIPTION
.Nm cw
is a facility for invoking one or more compilers, providing translation from
Sun style arguments as appropriate.
This allows the use of arbitrary compilers without the need to alter large
numbers of makefiles.
A mode called shadow compilation invokes multiple compilers so that warnings
and errors may be obtained from all of them.
See
.Sx SHADOW COMPILATION
for details.
This version of cw supports compilers with both Sun Studio 12 and GCC-style
command lines.
.Sh ARGUMENTS
Both the
.Fl -primary
and
.Fl -shadow
parameters take a
.Em compiler specification .
```

```
This is a comma-separated list of the form
.Ar name,executable,style
Where
.Ar name
is a name for the compiler,
.Ar executable
is the full path to the compiler executable, and
.Ar style
is the style of command-line options the compiler expects, either
.Em sun
or
.Em gnu .
.Bl -tag -width indent
.It Fl -primary Ar compiler
Specify the compiler to be used primarily (that which is used for link-editing
and pre-processing, and whos objects we deliver).
.It Fl -shadow Ar compiler
Specify a shadow compiler, which builds sources for the sake of checking code
quality and compatibility, but has its output discarded.
.It Fl -noecho
Do not echo the actual command line of any compilers invoked.
.It Fl -versions
Request from each configured primary and shadow compiler its version
information.
.It Fl C
The sources being compiled are C++.  This is necessary as it affects the
translation of compiler arguments.
.It Fl -
Arguments intended for the compilers themselves must be separated from those
of
.Nm cw
by a
.Fl - .
.It Fl _name=
.It Fl _style=
Parameters intended for the compiler be guarded with options of the form
.Fl _name=
and
.Fl _style=
Where
.Em name
and
.Em style
are those passed to
.Fl -primary
and
.Fl -shadow
this allows certain flags to be passed only to certain classes of compiler.
.Pp
For historical reasons, the
.Fl _style=
option is also translated such that a style of
.Em sun
may use the flag
.Fl _cc=
and a style of
.Em gnu
may use the flag
.Fl _gcc= ,
and when the
.Fl C
option is given and C++ is in use the style of
.Em sun
may use the flag
.Fl _CC=
and the style of
```

```
.Em gnu
may use the flag
.Fl _g++= .
.El
.Sh SHADOW COMPILATION
If
.Fl -shadow
compilers are specified
.Nm cw
will invoke each shadow compiler, with the outputs modified (as well as any
translation for compiler style) as follows:
.Bl -enum
.It
If
.Nm cw
is invoked to link-edit without compilation (the input files are all objects),
the shadow compiler is not invoked.
If neither of
.Fl c ,
.Fl S
appears in the argument list (that is, linking is attempted or only the
pre-processor is invoked), the shadow compilers will not be invoked.
.It
If the
.Fl o Ar filename
option was provided, with or without a separating space, it will be replaced wit
.Fl o Ar tempfile
.It
If the option
.Fl o
was not provided,
.Fl o Ar tempfile
will be added to the end of the argument list used to invoke
the shadow compilers.
.El
When shadow compilation is in effect,
.Nm cw
writes to standard error each compiler’s standard error output following its
argument list.
Messages from the compilers will not be interleaved.
If
.Nm cw
is used to invoke the preprocessor and no output location is specified,
.Nm cw
will write to standard output the primary compiler’s standard output.
.Pp
Because the Sun compilers write intermediate objects to fixed
filenames in the current directory when instructed to compile and
link multiple source files via a single command line, it would be
unsafe to invoke more than one compiler in this fashion.
Therefore
.Nm cw
does not accept multiple source files unless the preprocessor is to be
invoked.
An attempt to invoke
.Nm cw
in this manner will result in an error.
.Sh ARGUMENT TRANSLATION
If the compiler to be invoked is a GNU-style C or C++ compiler, a set of
default flags is added to the beginning of the argument list, and the
remaining arguments are translated to their closest appropriate
semantic equivalents and passed in the same order as their
counterparts given to
.Nm cw .
See the comments at the head of
.Pa usr/src/tools/cw/cw.c
```

```
for a detailed list of translations.
.Sh ENVIRONMENT
.Bl -tag -width indent
.It CW_SHADOW_SERIAL
If this variable is set in the environment, invoke the primary compiler, wait
for it to complete, then invoke the shadow compilers.
Normally the primary and shadow compilers are invoked in parallel.
.It CW_NO_EXEC
f this variable is set in the environment, write the usual output to
standard error but do not actually invoke any compiler.
This is useful for debugging the translation engine.
.El
.Sh EXIT STATUS
The following exit status values are returned:
.Bl -tag -width indent
.It 0
The primary compiler, and shadow compilers if invoked, all completed
successfully.
.It >0
A usage error occurred, or one or more compilers returned a nonzero
exit status.
.El
.Sh SEE ALSO
.Xr cc 1 ,
.Xr CC 1 ,
.Xr gcc 1
.Sh BUGS
The translations provided for gcc are not always exact and in some cases
reflect local policy rather than actual equivalence.
.Pp
Additional compiler types should be supported.
.Pp
The translation engine is hacky.
```

****
   43270 Fri Dec 28 22:22:48 2018
new/usr/src/tools/cw/cw.c
Cleanup temp files without rm
Cleanup cw tempfiles properly
Revert "Revert most of "9899 cw(1onbld) should shadow more compilation""
This reverts commit 67deef8cbc83060db238a0f4ee252d1ba74641ef.
****

```
/*
 * 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 2018, Richard Lowe.
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Wrapper for the GNU C compiler to make it accept the Sun C compiler
 * arguments where possible.
 *
 * Since the translation is inexact, this is something of a work-in-progress.
 *
 */

/* If you modify this file, you must increment CW_VERSION */
#define CW_VERSION      "3.0"

/*
 * -#           Verbose mode
 * -###         Show compiler commands built by driver, no compilation
 * -A<name[(tokens)]>   Preprocessor predicate assertion
 * -B<[static|dynamic]> Specify dynamic or static binding
 * -C           Prevent preprocessor from removing comments
 * -c           Compile only - produce .o files, suppress linking
 * -cg92        Alias for -xtarget=ss1000
 * -D<name[=token]>     Associate name with token as if by #define
 * -d[y|n]      dynamic [-dy] or static [-dn] option to linker
 * -E           Compile source through preprocessor only, output to stdout
 * -erroff=<t>  Suppress warnings specified by tags t(%none, %all, <tag list>)
 * -errtags=<a> Display messages with tags a(no, yes)
 * -errwarn=<t> Treats warnings specified by tags t(%none, %all, <tag list>)
 *              as errors
 * -fast        Optimize using a selection of options
 * -fd          Report old-style function definitions and declarations
```

```
 * -fnonstd     Initialize floating-point hardware to non-standard preferences
 * -fns[=<yes|no>] Select non-standard floating point mode
 * -fprecision=<p> Set FP rounding precision mode p(single, double, extended)
 * -fround=<r>  Select the IEEE rounding mode in effect at startup
 * -fsimple[=<n>] Select floating-point optimization preferences <n>
 * -fsingle     Use single-precision arithmetic (-Xt and -Xs modes only)
 * -ftrap=<t>   Select floating-point trapping mode in effect at startup
 * -fstore      force floating pt. values to target precision on assignment
 * -G           Build a dynamic shared library
 * -g           Compile for debugging
 * -H           Print path name of each file included during compilation
 * -h <name>    Assign <name> to generated dynamic shared library
 * -I<dir>      Add <dir> to preprocessor #include file search path
 * -i           Passed to linker to ignore any LD_LIBRARY_PATH setting
 * -keeptmp     Keep temporary files created during compilation
 * -L<dir>      Pass to linker to add <dir> to the library search path
 * -l<name>     Link with library lib<name>.a or lib<name>.so
 * -mc          Remove duplicate strings from .comment section of output files
 * -mr          Remove all strings from .comment section of output files
 * -mr,"string" Remove all strings and append "string" to .comment section
 * -mt          Specify options needed when compiling multi-threaded code
 * -native      Find available processor, generate code accordingly
 * -nofstore    Do not force floating pt. values to target precision
 *              on assignment
 * -norunpath   Do not build in a runtime path for shared libraries
 * -O           Use default optimization level (-xO2 or -xO3. Check man page.)
 * -o <outputfile> Set name of output file to <outputfile>
 * -P           Compile source through preprocessor only, output to .i  file
 * -p           Compile for profiling with prof
 * -Q[y|n]      Emit/don’t emit identification info to output file
 * -R<dir[:dir]> Build runtime search path list into executable
 * -S           Compile and only generate assembly code (.s)
 * -s           Strip symbol table from the executable file
 * -t           Turn off duplicate symbol warnings when linking
 * -U<name>     Delete initial definition of preprocessor symbol <name>
 * -V           Report version number of each compilation phase
 * -v           Do stricter semantic checking
 * -W<c>,<arg>  Pass <arg> to specified component <c> (a,l,m,p,0,2,h,i,u)
 * -w           Suppress compiler warning messages
 * -Xa          Compile assuming ANSI C conformance, allow K & R extensions
 *              (default mode)
 * -Xs          Compile assuming (pre-ANSI) K & R C style code
 * -Xt          Compile assuming K & R conformance, allow ANSI C
 * -xarch=<a>   Specify target architecture instruction set
 * -xbuiltin[=<b>] When profitable inline, or substitute intrinisic functions
 *              for system functions, b={%all,%none}
 * -xCC         Accept C++ style comments
 * -xchip=<c>   Specify the target processor for use by the optimizer
 * -xcode=<c>   Generate different code for forming addresses
 * -xcrossfile[=<n>] Enable optimization and inlining across source files,
 *              n={0|1}
 * -xe          Perform only syntax/semantic checking, no code generation
 * -xF          Compile for later mapfile reordering or unused section
 *              elimination
 * -xhelp=<f>   Display on-line help information f(flags, readme, errors)
 * -xildoff     Cancel -xildon
 * -xildon      Enable use of the incremental linker, ild
 * -xinline=[<a>,...,<a>]  Attempt inlining of specified user routines,
 *              <a>={%auto,func,no%func}
 * -xlibmieee   Force IEEE 754 return values for math routines in
 *              exceptional cases
 * -xlibmil     Inline selected libm math routines for optimization
 * -xlic_lib=sunperf    Link in the Sun supplied performance libraries
 * -xlicinfo    Show license server information
 * -xmaxopt=[off,1,2,3,4,5] maximum optimization level allowed on #pragma opt
 * -xO<n>       Generate optimized code (n={1|2|3|4|5})
 * -xP          Print prototypes for function definitions
 * -xprofile=<p> Collect data for a profile or use a profile to optimize
 *              <p>={{collect,use}[:<path>],tcov}
 * -xregs=<r>   Control register allocation
 * -xs          Allow debugging without object (.o) files
 * -xsb         Compile for use with the WorkShop source browser
 * -xsbfast     Generate only WorkShop source browser info, no compilation
 * -xsfpconst   Represent unsuffixed floating point constants as single
 *              precision
 * -xspace      Do not do optimizations that increase code size
 * -xstrconst   Place string literals into read-only data segment
 * -xtarget=<t> Specify target system for optimization
 * -xtemp=<dir> Set directory for temporary files to <dir>
 * -xtime       Report the execution time for each compilation phase
 * -xunroll=n   Enable unrolling loops n times where possible
 * -Y<c>,<dir>  Specify <dir> for location of component <c> (a,l,m,p,0,h,i,u)
 * -YA,<dir>    Change default directory searched for components
 * -YI,<dir>    Change default directory searched for include files
 * -YP,<dir>    Change default directory for finding libraries files
 * -YS,<dir>    Change default directory for startup object files
 */

/*
 * Translation table:
 */
/*
 * -#                           -v
 * -###                         error
 * -A<name[(tokens)]>           pass-thru
 * -B<[static|dynamic]>         pass-thru (syntax error for anything else)
 * -C                           pass-thru
 * -c                           pass-thru
 * -cg92                        -m32 -mcpu=v8 -mtune=supersparc (SPARC only)
 * -D<name[=token]>             pass-thru
 * -dy or -dn                   -Wl,-dy or -Wl,-dn
 * -E                           pass-thru
 * -erroff=E_EMPTY_TRANSLATION_UNIT ignore
 * -errtags=%all                -Wall
 * -errwarn=%all                -Werror else -Wno-error
 * -fast                        error
 * -fd                          error
 * -fnonstd                     error
 * -fns[=<yes|no>]              error
 * -fprecision=<p>              error
 * -fround=<r>                  error
 * -fsimple[=<n>]               error
 * -fsingle[=<n>]               error
 * -ftrap=<t>                   error
 * -fstore                      error
 * -G                           pass-thru
 * -g                           pass-thru
 * -H                           pass-thru
 * -h <name>                    pass-thru
 * -I<dir>                      pass-thru
 * -i                           pass-thru
 * -keeptmp                     -save-temps
 * -L<dir>                      pass-thru
 * -l<name>                     pass-thru
 * -mc                          error
 * -mr                          error
 * -mr,"string"                 error
 * -mt                          -D_REENTRANT
 * -native                      error
 * -nofstore                    error
 * -nolib                       -nodefaultlibs
 * -norunpath                   ignore
```

```
 * -O                           -O1 (Check the man page to be certain)
 * -o <outputfile>              pass-thru
 * -P                           -E -o filename.i (or error)
 * -p                           pass-thru
 * -Q[y|n]                      error
 * -R<dir[:dir]>                pass-thru
 * -S                           pass-thru
 * -s                           -Wl,-s
 * -t                           -Wl,-t
 * -U<name>                     pass-thru
 * -V                           --version
 * -v                           -Wall
 * -Wa,<arg>                    pass-thru
 * -Wp,<arg>                    pass-thru except -xc99=<a>
 * -Wl,<arg>                    pass-thru
 * -W{m,0,2,h,i,u>              error/ignore
 * -xmodel=kernel               -ffreestanding -mcmodel=kernel -mno-red-zone
 * -Wu,-save_args               -msave-args
 * -w                           pass-thru
 * -Xa                          -std=iso9899:199409 or -ansi
 * -Xt                          error
 * -Xs                          -traditional -std=c89
 * -xarch=<a>                   table
 * -xbuiltin[=<b>]              -fbuiltin (-fno-builtin otherwise)
 * -xCC                         ignore
 * -xchip=<c>                   table
 * -xcode=<c>                   table
 * -xdebugformat=<format>       ignore (always use dwarf-2 for gcc)
 * -xcrossfile[=<n>]            ignore
 * -xe                          error
 * -xF                          error
 * -xhelp=<f>                   error
 * -xildoff                     ignore
 * -xildon                      ignore
 * -xinline                     ignore
 * -xlibmieee                   error
 * -xlibmil                     error
 * -xlic_lib=sunperf            error
 * -xmaxopt=[...]               error
 * -xO<n>                       -O<n>
 * -xP                          error
 * -xprofile=<p>                error
 * -xregs=<r>                   table
 * -xs                          error
 * -xsb                         error
 * -xsbfast                     error
 * -xsfpconst                   error
 * -xspace                      ignore (-not -Os)
 * -xstrconst                   ignore
 * -xtarget=<t>                 table
 * -xtemp=<dir>                 error
 * -xtime                       error
 * -xtransition                 -Wtransition
 * -xunroll=n                   error
 * -W0,-xdbggen=no%usedonly     -fno-eliminate-unused-debug-symbols
 *                              -fno-eliminate-unused-debug-types
 * -Y<c>,<dir>                  error
 * -YA,<dir>                    error
 * -YI,<dir>                    -nostdinc -I<dir>
 * -YP,<dir>                    error
 * -YS,<dir>                    error
 */

#include <ctype.h>
#include <err.h>
#include <errno.h>
```

```
#include <fcntl.h>
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#endif /* ! codereview */

#include <sys/param.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/wait.h>

#define CW_F_CXX        0x01
#define CW_F_SHADOW     0x02
#define CW_F_EXEC       0x04
#define CW_F_ECHO       0x08
#define CW_F_XLATE      0x10
#define CW_F_PROG       0x20

typedef enum cw_op {
        CW_O_NONE = 0,
        CW_O_PREPROCESS,
        CW_O_COMPILE,
        CW_O_LINK
} cw_op_t;

struct aelist {
        struct ae {
                struct ae *ae_next;
                char *ae_arg;
        } *ael_head, *ael_tail;
        int ael_argc;
};

typedef enum {
        GNU,
        SUN
} compiler_style_t;

typedef struct {
        char *c_name;
        char *c_path;
        compiler_style_t c_style;
} cw_compiler_t;

typedef struct cw_ictx {
        struct cw_ictx  *i_next;
        cw_compiler_t   *i_compiler;
        struct aelist   *i_ae;
        uint32_t        i_flags;
        int             i_oldargc;
        char            **i_oldargv;
        pid_t           i_pid;
        char            *i_tmpdir;
        char            i_discard[MAXPATHLEN];
        char            *i_stderr;
} cw_ictx_t;
```
______***unchanged_portion_omitted_***

```
/*
 * The compiler wants the output file to end in appropriate extension.  If
 * we're generating a name from whole cloth (path == NULL), we assume that
 * extension to be .o, otherwise we match the extension of the caller.
```

```
 */
static char *
discard_file_name(cw_ictx_t *ctx, const char *path)
{
        char *ret, *ext;
        char tmpl[] = "cwXXXXXX";

        if (path == NULL) {
                ext = ".o";
        } else {
                ext = strrchr(path, '.');
        }

        /*
         * We need absolute control over where the temporary file goes, since
         * we rely on it for cleanup so tempnam(3C) and tmpnam(3C) are
         * inappropriate (they use TMPDIR, preferentially).
         *
         * mkstemp(3C) doesn't actually help us, since the temporary file
         * isn't used by us, only its name.
         */
        if (mktemp(tmpl) == NULL)
                nomem();

        (void) asprintf(&ret, "%s/%s%s", ctx->i_tmpdir, tmpl,
            (ext != NULL) ? ext : "");

        if (ret == NULL)
                nomem();

        return (ret);
}

#endif /* ! codereview */
static void
do_gcc(cw_ictx_t *ctx)
{
        int c;
        int nolibc = 0;
        int in_output = 0, seen_o = 0, c_files = 0;
        cw_op_t op = CW_O_LINK;
        char *model = NULL;
        char *nameflag;
        int mflag = 0;

        if (ctx->i_flags & CW_F_PROG) {
                newae(ctx->i_ae, "--version");
                return;
        }

        newae(ctx->i_ae, "-fident");
        newae(ctx->i_ae, "-finline");
        newae(ctx->i_ae, "-fno-inline-functions");
        newae(ctx->i_ae, "-fno-builtin");
        newae(ctx->i_ae, "-fno-asm");
        newae(ctx->i_ae, "-fdiagnostics-show-option");
        newae(ctx->i_ae, "-nodefaultlibs");

#if defined(__sparc)
        /*
         * The SPARC ldd and std instructions require 8-byte alignment of
         * their address operand.  gcc correctly uses them only when the
         * ABI requires 8-byte alignment; unfortunately we have a number of
         * pieces of buggy code that doesn't conform to the ABI.  This
         * flag makes gcc work more like Studio with -xmemalign=4.
         */
        newae(ctx->i_ae, "-mno-integer-ldd-std");
#endif

        /*
         * This is needed because 'u' is defined
         * under a conditional on 'sun'.  Should
         * probably just remove the conditional,
         * or make it be dependent on '__sun'.
         *
         * -Dunix is also missing in enhanced ANSI mode
         */
        newae(ctx->i_ae, "-D__sun");

        if (asprintf(&nameflag, "-_%s=", ctx->i_compiler->c_name) == -1)
                nomem();

        /*
         * Walk the argument list, translating as we go ..
         */
        while (--ctx->i_oldargc > 0) {
                char *arg = *++ctx->i_oldargv;
                size_t arglen = strlen(arg);

                if (*arg == '-') {
                        arglen--;
                } else {
                        /*
                         * Discard inline files that gcc doesn't grok
                         */
                        if (!in_output && arglen > 3 &&
                            strcmp(arg + arglen - 3, ".il") == 0)
                                continue;

                        if (!in_output && arglen > 2 &&
                            arg[arglen - 2] == '.' &&
                            (arg[arglen - 1] == 'S' || arg[arglen - 1] == 's' ||
                            arg[arglen - 1] == 'c' || arg[arglen - 1] == 'i'))
                                c_files++;

                        /*
                         * Otherwise, filenames and partial arguments
                         * are passed through for gcc to chew on.  However,
                         * output is always discarded for the secondary
                         * compiler.
                         */
                        if ((ctx->i_flags & CW_F_SHADOW) && in_output) {
                                newae(ctx->i_ae, discard_file_name(ctx, arg));
                        } else {
                        if ((ctx->i_flags & CW_F_SHADOW) && in_output)
                                newae(ctx->i_ae, ctx->i_discard);
                        else
                                newae(ctx->i_ae, arg);
                        }
#endif /* ! codereview */
                        in_output = 0;
                        continue;
                }

                if (ctx->i_flags & CW_F_CXX) {
                        if (strncmp(arg, "-_g++=", 6) == 0) {
                                newae(ctx->i_ae, strchr(arg, '=') + 1);
                                continue;
                        }
                        if (strncmp(arg, "-compat=", 8) == 0) {
                                /* discard -compat=4 and -compat=5 */
                                continue;
```

```
                        }
                        if (strcmp(arg, "-Qoption") == 0) {
                                /* discard -Qoption and its two arguments */
                                if (ctx->i_oldargc < 3)
                                        error(arg);
                                ctx->i_oldargc -= 2;
                                ctx->i_oldargv += 2;
                                continue;
                        }
                        if (strcmp(arg, "-xwe") == 0) {
                                /* turn warnings into errors */
                                newae(ctx->i_ae, "-Werror");
                                continue;
                        }
                        if (strcmp(arg, "-norunpath") == 0) {
                                /* gcc has no corresponding option */
                                continue;
                        }
                        if (strcmp(arg, "-nolib") == 0) {
                                /* -nodefaultlibs is on by default */
                                nolibc = 1;
                                continue;
                        }
#if defined(__sparc)
                        if (strcmp(arg, "-cg92") == 0) {
                                mflag |= xlate_xtb(ctx->i_ae, "v8");
                                xlate(ctx->i_ae, "super", xchip_tbl);
                                continue;
                        }
#endif  /* __sparc */
                }

                switch ((c = arg[1])) {
                case '_':
                        if ((strncmp(arg, nameflag, strlen(nameflag)) == 0) ||
                            (strncmp(arg, "-_gcc=", 6) == 0) ||
                            (strncmp(arg, "-_gnu=", 6) == 0)) {
                                newae(ctx->i_ae, strchr(arg, '=') + 1);
                        }
                        break;
                case '#':
                        if (arglen == 1) {
                                newae(ctx->i_ae, "-v");
                                break;
                        }
                        error(arg);
                        break;
                case 'f':
                        if ((strcmp(arg, "-fpic") == 0) ||
                            (strcmp(arg, "-fPIC") == 0)) {
                                newae(ctx->i_ae, arg);
                                break;
                        }
                        error(arg);
                        break;
                case 'g':
                        newae(ctx->i_ae, "-gdwarf-2");
                        break;
                case 'E':
                        if (arglen == 1) {
                                newae(ctx->i_ae, "-xc");
                                newae(ctx->i_ae, arg);
                                op = CW_O_PREPROCESS;
                                nolibc = 1;
                                break;
                        }
                        error(arg);
                        break;
                case ’c’:
                case ’S’:
                        if (arglen == 1) {
                                op = CW_O_COMPILE;
                                nolibc = 1;
                        }
                        /* FALLTHROUGH */
                case ’C’:
                case ’H’:
                case ’p’:
                        if (arglen == 1) {
                                newae(ctx->i_ae, arg);
                                break;
                        }
                        error(arg);
                        break;
                case ’A’:
                case ’h’:
                case ’I’:
                case ’i’:
                case ’L’:
                case ’l’:
                case ’R’:
                case ’U’:
                case ’u’:
                case ’w’:
                        newae(ctx->i_ae, arg);
                        break;
                case ’o’:
                        seen_o = 1;
                        if (arglen == 1) {
                                in_output = 1;
                                newae(ctx->i_ae, arg);
                        } else if (ctx->i_flags & CW_F_SHADOW) {
                                newae(ctx->i_ae, "-o");
                                newae(ctx->i_ae, discard_file_name(ctx, arg));
                                newae(ctx->i_ae, ctx->i_discard);
                        } else {
                                newae(ctx->i_ae, arg);
                        }
                        break;
                case ’D’:
                        newae(ctx->i_ae, arg);
                        /*
                         * XXX  Clearly a hack ... do we need _KADB too?
                         */
                        if (strcmp(arg, "-D_KERNEL") == 0 ||
                            strcmp(arg, "-D_BOOT") == 0)
                                newae(ctx->i_ae, "-ffreestanding");
                        break;
                case ’d’:
                        if (arglen == 2) {
                                if (strcmp(arg, "-dy") == 0) {
                                        newae(ctx->i_ae, "-Wl,-dy");
                                        break;
                                }
                                if (strcmp(arg, "-dn") == 0) {
                                        newae(ctx->i_ae, "-Wl,-dn");
                                        break;
                                }
                        }
                        if (strcmp(arg, "-dalign") == 0) {
                                /*
                                 * -dalign forces alignment in some cases;
```

```
                                 * gcc does not need any flag to do this.
                                 */
                                break;
                        }
                        error(arg);
                        break;
                case ’e’:
                        if (strcmp(arg,
                            "-erroff=E_EMPTY_TRANSLATION_UNIT") == 0) {
                                /*
                                 * Accept but ignore this -- gcc doesn’t
                                 * seem to complain about empty translation
                                 * units
                                 */
                                break;
                        }
                        /* XX64 -- ignore all -erroff= options, for now */
                        if (strncmp(arg, "-erroff=", 8) == 0)
                                break;
                        if (strcmp(arg, "-errtags=yes") == 0) {
                                warnings(ctx->i_ae);
                                break;
                        }
                        if (strcmp(arg, "-errwarn=%all") == 0) {
                                newae(ctx->i_ae, "-Werror");
                                break;
                        }
                        error(arg);
                        break;
                case ’G’:
                        newae(ctx->i_ae, "-shared");
                        nolibc = 1;
                        break;
                case ’k’:
                        if (strcmp(arg, "-keeptmp") == 0) {
                                newae(ctx->i_ae, "-save-temps");
                                break;
                        }
                        error(arg);
                        break;
                case ’m’:
                        if (strcmp(arg, "-mt") == 0) {
                                newae(ctx->i_ae, "-D_REENTRANT");
                                break;
                        }
                        if (strcmp(arg, "-m64") == 0) {
                                newae(ctx->i_ae, "-m64");
#if defined(__x86)
                                newae(ctx->i_ae, "-mtune=opteron");
#endif
                                mflag |= M64;
                                break;
                        }
                        if (strcmp(arg, "-m32") == 0) {
                                newae(ctx->i_ae, "-m32");
                                mflag |= M32;
                                break;
                        }
                        error(arg);
                        break;
                case ’B’:       /* linker options */
                case ’M’:
                case ’z’:
                        {
                                char *opt;
                                size_t len;
                                char *s;

                                if (arglen == 1) {
                                        opt = *++ctx->i_oldargv;
                                        if (opt == NULL || *opt == '\0')
                                                error(arg);
                                        ctx->i_oldargc--;
                                } else {
                                        opt = arg + 2;
                                }
                                len = strlen(opt) + 7;
                                if ((s = malloc(len)) == NULL)
                                        nomem();
                                (void) snprintf(s, len, "-Wl,-%c%s", c, opt);
                                newae(ctx->i_ae, s);
                                free(s);
                        }
                        break;
                case 'O':
                        if (arglen == 1) {
                                newae(ctx->i_ae, "-O");
                                break;
                        }
                        error(arg);
                        break;
                case 'P':
                        /*
                         * We could do '-E -o filename.i', but that's hard,
                         * and we don't need it for the case that's triggering
                         * this addition.  We'll require the user to specify
                         * -o in the Makefile.  If they don't they'll find out
                         * in a hurry.
                         */
                        newae(ctx->i_ae, "-E");
                        op = CW_O_PREPROCESS;
                        nolibc = 1;
                        break;
                case 's':
                        if (arglen == 1) {
                                newae(ctx->i_ae, "-Wl,-s");
                                break;
                        }
                        error(arg);
                        break;
                case 't':
                        if (arglen == 1) {
                                newae(ctx->i_ae, "-Wl,-t");
                                break;
                        }
                        error(arg);
                        break;
                case 'V':
                        if (arglen == 1) {
                                ctx->i_flags &= ~CW_F_ECHO;
                                newae(ctx->i_ae, "--version");
                                break;
                        }
                        error(arg);
                        break;
                case 'v':
                        if (arglen == 1) {
                                warnings(ctx->i_ae);
                                break;
                        }
                        error(arg);
                        break;
```

```
                case 'W':
                        if (strncmp(arg, "-Wp,-xc99", 9) == 0) {
                                /*
                                 * gcc's preprocessor will accept c99
                                 * regardless, so accept and ignore.
                                 */
                                break;
                        }
                        if (strncmp(arg, "-Wa,", 4) == 0 ||
                            strncmp(arg, "-Wp,", 4) == 0 ||
                            strncmp(arg, "-Wl,", 4) == 0) {
                                newae(ctx->i_ae, arg);
                                break;
                        }
                        if (strcmp(arg, "-W0,-noglobal") == 0 ||
                            strcmp(arg, "-W0,-xglobalstatic") == 0) {
                                /*
                                 * gcc doesn't prefix local symbols
                                 * in debug mode, so this is not needed.
                                 */
                                break;
                        }
                        if (strcmp(arg, "-W0,-Lt") == 0) {
                                /*
                                 * Generate tests at the top of loops.
                                 * There is no direct gcc equivalent, ignore.
                                 */
                                break;
                        }
                        if (strcmp(arg, "-W0,-xdbggen=no%usedonly") == 0) {
                                newae(ctx->i_ae,
                                    "-fno-eliminate-unused-debug-symbols");
                                newae(ctx->i_ae,
                                    "-fno-eliminate-unused-debug-types");
                                break;
                        }
                        if (strcmp(arg, "-W2,-xwrap_int") == 0) {
                                /*
                                 * Use the legacy behaviour (pre-SS11)
                                 * for integer wrapping.
                                 * gcc does not need this.
                                 */
                                break;
                        }
                        if (strcmp(arg, "-Wd,-xsafe=unboundsym") == 0) {
                                /*
                                 * Prevents optimizing away checks for
                                 * unbound weak symbol addresses.  gcc does
                                 * not do this, so it's not needed.
                                 */
                                break;
                        }
                        if (strncmp(arg, "-Wc,-xcode=", 11) == 0) {
                                xlate(ctx->i_ae, arg + 11, xcode_tbl);
                                break;
                        }
                        if (strncmp(arg, "-Wc,-Qiselect", 13) == 0) {
                                /*
                                 * Prevents insertion of register symbols.
                                 * gcc doesn't do this, so ignore it.
                                 */
                                break;
                        }
                        if (strcmp(arg, "-Wc,-Qassembler-ounrefsym=0") == 0) {
                                /*
                                 * Prevents optimizing away of static variables.
                                 * gcc does not do this, so it’s not needed.
                                 */
                                break;
                        }
#if defined(__x86)
                        if (strcmp(arg, "-Wu,-save_args") == 0) {
                                newae(ctx->i_ae, "-msave-args");
                                break;
                        }
#endif  /* __x86 */
                        error(arg);
                        break;
                case ’X’:
                        if (strcmp(arg, "-Xa") == 0 ||
                            strcmp(arg, "-Xt") == 0) {
                                break;
                        }
                        if (strcmp(arg, "-Xs") == 0) {
                                Xsmode(ctx->i_ae);
                                break;
                        }
                        error(arg);
                        break;
                case ’x’:
                        if (arglen == 1)
                                error(arg);
                        switch (arg[2]) {
                        case ’a’:
                                if (strncmp(arg, "-xarch=", 7) == 0) {
                                        mflag |= xlate_xtb(ctx->i_ae, arg + 7);
                                        break;
                                }
                                error(arg);
                                break;
                        case ’b’:
                                if (strncmp(arg, "-xbuiltin=", 10) == 0) {
                                        if (strcmp(arg + 10, "%all"))
                                                newae(ctx->i_ae, "-fbuiltin");
                                        break;
                                }
                                error(arg);
                                break;
                        case ’C’:
                                /* Accept C++ style comments -- ignore */
                                if (strcmp(arg, "-xCC") == 0)
                                        break;
                                error(arg);
                                break;
                        case ’c’:
                                if (strncmp(arg, "-xc99=%all", 10) == 0) {
                                        newae(ctx->i_ae, "-std=gnu99");
                                        break;
                                }
                                if (strncmp(arg, "-xc99=%none", 11) == 0) {
                                        newae(ctx->i_ae, "-std=gnu89");
                                        break;
                                }
                                if (strncmp(arg, "-xchip=", 7) == 0) {
                                        xlate(ctx->i_ae, arg + 7, xchip_tbl);
                                        break;
                                }
                                if (strncmp(arg, "-xcode=", 7) == 0) {
                                        xlate(ctx->i_ae, arg + 7, xcode_tbl);
                                        break;
                                }
                                if (strncmp(arg, "-xcrossfile", 11) == 0)
```

```
                                        break;
                                error(arg);
                                break;
                        case ’d’:
                                if (strncmp(arg, "-xdebugformat=", 14) == 0)
                                        break;
                                error(arg);
                                break;
                        case ’F’:
                                /*
                                 * Compile for mapfile reordering, or unused
                                 * section elimination, syntax can be -xF or
                                 * more complex, like -xF=%all -- ignore.
                                 */
                                if (strncmp(arg, "-xF", 3) == 0)
                                        break;
                                error(arg);
                                break;
                        case ’i’:
                                if (strncmp(arg, "-xinline", 8) == 0)
                                        /* No inlining; ignore */
                                        break;
                                if (strcmp(arg, "-xildon") == 0 ||
                                    strcmp(arg, "-xildoff") == 0)
                                        /* No incremental linking; ignore */
                                        break;
                                error(arg);
                                break;
#if defined(__x86)
                        case ’m’:
                                if (strcmp(arg, "-xmodel=kernel") == 0) {
                                        newae(ctx->i_ae, "-ffreestanding");
                                        newae(ctx->i_ae, "-mno-red-zone");
                                        model = "-mcmodel=kernel";
                                        nolibc = 1;
                                        break;
                                }
                                error(arg);
                                break;
#endif  /* __x86 */
                        case ’O’:
                                if (strncmp(arg, "-xO", 3) == 0) {
                                        size_t len = strlen(arg);
                                        char *s = NULL;
                                        int c = *(arg + 3);
                                        int level;

                                        if (len != 4 || !isdigit(c))
                                                error(arg);

                                        level = atoi(arg + 3);
                                        if (level > 5)
                                                error(arg);
                                        if (level >= 2) {
                                                /*
                                                 * For gcc-3.4.x at -O2 we
                                                 * need to disable optimizations
                                                 * that break ON.
                                                 */
                                                optim_disable(ctx->i_ae, level);
                                                /*
                                                 * limit -xO3 to -O2 as well.
                                                 */
                                                level = 2;
                                        }
                                        if (asprintf(&s, "-O%d", level) == -1)
```

```
                                                nomem();
                                        newae(ctx->i_ae, s);
                                        free(s);
                                        break;
                                }
                                error(arg);
                                break;
                        case 'r':
                                if (strncmp(arg, "-xregs=", 7) == 0) {
                                        xlate(ctx->i_ae, arg + 7, xregs_tbl);
                                        break;
                                }
                                error(arg);
                                break;
                        case 's':
                                if (strcmp(arg, "-xs") == 0 ||
                                    strcmp(arg, "-xspace") == 0 ||
                                    strcmp(arg, "-xstrconst") == 0)
                                        break;
                                error(arg);
                                break;
                        case 't':
                                if (strncmp(arg, "-xtarget=", 9) == 0) {
                                        xlate(ctx->i_ae, arg + 9, xtarget_tbl);
                                        break;
                                }
                                error(arg);
                                break;
                        case 'e':
                        case 'h':
                        case 'l':
                        default:
                                error(arg);
                                break;
                        }
                        break;
                case 'Y':
                        if (arglen == 1) {
                                if ((arg = *++ctx->i_oldargv) == NULL ||
                                    *arg == '\0')
                                        error("-Y");
                                ctx->i_oldargc--;
                                arglen = strlen(arg + 1);
                        } else {
                                arg += 2;
                        }
                        /* Just ignore -YS,... for now */
                        if (strncmp(arg, "S,", 2) == 0)
                                break;
                        if (strncmp(arg, "l,", 2) == 0) {
                                char *s = strdup(arg);
                                s[0] = '-';
                                s[1] = 'B';
                                newae(ctx->i_ae, s);
                                free(s);
                                break;
                        }
                        if (strncmp(arg, "I,", 2) == 0) {
                                char *s = strdup(arg);
                                s[0] = '-';
                                s[1] = 'I';
                                newae(ctx->i_ae, "-nostdinc");
                                newae(ctx->i_ae, s);
                                free(s);
                                break;
                        }
```

```
                                error(arg);
                                break;
                        case 'Q':
                                /*
                                 * We could map -Qy into -Wl,-Qy etc.
                                 */
                        default:
                                error(arg);
                                break;
                        }
                }

                free(nameflag);

                /*
                 * When compiling multiple source files in a single invocation some
                 * compilers output objects into the current directory with
                 * predictable and conventional names.
                 *
                 * We prevent any attempt to compile multiple files at once so that
                 * any such objects created by a shadow can't escape into a later
                 * link-edit.
                 */
                if (c_files > 1 && op != CW_O_PREPROCESS) {
 950            if (c_files > 1 && (ctx->i_flags & CW_F_SHADOW) &&
 951                op != CW_O_PREPROCESS) {
                        errx(2, "multiple source files are "
                            "allowed only with -E or -P");
                }

                /*
                 * Make sure that we do not have any unintended interactions between
                 * the xarch options passed in and the version of the Studio compiler
                 * used.
                 */
                if ((mflag & (SS11|SS12)) == (SS11|SS12)) {
                        errx(2,
                            "Conflicting \"-xarch=\" flags (both Studio 11 and 12)\n");
                }

                switch (mflag) {
                case 0:
                        /* FALLTHROUGH */
                case M32:
#if defined(__sparc)
                        /*
                         * Only -m32 is defined and so put in the missing xarch
                         * translation.
                         */
                        newae(ctx->i_ae, "-mcpu=v8");
                        newae(ctx->i_ae, "-mno-v8plus");
#endif
                        break;
                case M64:
#if defined(__sparc)
                        /*
                         * Only -m64 is defined and so put in the missing xarch
                         * translation.
                         */
                        newae(ctx->i_ae, "-mcpu=v9");
#endif
                        break;
                case SS12:
#if defined(__sparc)
                        /* no -m32/-m64 flag used - this is an error for sparc builds */
                        (void) fprintf(stderr, "No -m32/-m64 flag defined\n");
                exit(2);
#endif
                break;
        case SS11:
                /* FALLTHROUGH */
        case (SS11|M32):
        case (SS11|M64):
                break;
        case (SS12|M32):
#if defined(__sparc)
                /*
                 * Need to add in further 32 bit options because with SS12
                 * the xarch=sparcvis option can be applied to 32 or 64
                 * bit, and so the translatation table (xtbl) cannot handle
                 * that.
                 */
                newae(ctx->i_ae, "-mv8plus");
#endif
                break;
        case (SS12|M64):
                break;
        default:
                (void) fprintf(stderr,
                    "Incompatible -xarch= and/or -m32/-m64 options used.\n");
                exit(2);
        }

        if (ctx->i_flags & CW_F_SHADOW) {
                if (op == CW_O_PREPROCESS)
                        exit(0);
                else if (op == CW_O_LINK && c_files == 0)
        if ((op == CW_O_LINK || op == CW_O_PREPROCESS) &&
            (ctx->i_flags & CW_F_SHADOW))
                        exit(0);
        }
#endif /* ! codereview */

        if (model != NULL)
                newae(ctx->i_ae, model);
        if (!nolibc)
                newae(ctx->i_ae, "-lc");
        if (!seen_o && (ctx->i_flags & CW_F_SHADOW)) {
                newae(ctx->i_ae, "-o");
                newae(ctx->i_ae, discard_file_name(ctx, NULL));
                newae(ctx->i_ae, ctx->i_discard);
        }
}

static void
do_cc(cw_ictx_t *ctx)
{
        int in_output = 0, seen_o = 0, c_files = 0;
        int in_output = 0, seen_o = 0;
        cw_op_t op = CW_O_LINK;
        char *nameflag;

        if (ctx->i_flags & CW_F_PROG) {
                newae(ctx->i_ae, "-V");
                return;
        }

        if (asprintf(&nameflag, "-_%s=", ctx->i_compiler->c_name) == -1)
                nomem();

        while (--ctx->i_oldargc > 0) {
                char *arg = *++ctx->i_oldargv;
```

```
                size_t arglen = strlen(arg);
#endif /* ! codereview */

                if (strncmp(arg, "-_CC=", 5) == 0) {
                        newae(ctx->i_ae, strchr(arg, '=') + 1);
                        continue;
                }

                if (*arg != '-') {
                        if (!in_output && arglen > 2 &&
                            arg[arglen - 2] == '.' &&
                            (arg[arglen - 1] == 'S' || arg[arglen - 1] == 's' ||
                            arg[arglen - 1] == 'c' || arg[arglen - 1] == 'i'))
                                c_files++;

#endif /* ! codereview */
                        if (in_output == 0 || !(ctx->i_flags & CW_F_SHADOW)) {
                                newae(ctx->i_ae, arg);
                        } else {
                                in_output = 0;
                                newae(ctx->i_ae, discard_file_name(ctx, arg));
                                newae(ctx->i_ae, ctx->i_discard);
                        }
                        continue;
                }
                switch (*(arg + 1)) {
                case '_':
                        if ((strncmp(arg, nameflag, strlen(nameflag)) == 0) ||
                            (strncmp(arg, "-_cc=", 5) == 0) ||
                            (strncmp(arg, "-_sun=", 6) == 0)) {
                                newae(ctx->i_ae, strchr(arg, '=') + 1);
                        }
                        break;

                case 'V':
                        ctx->i_flags &= ~CW_F_ECHO;
                        newae(ctx->i_ae, arg);
                        break;
                case 'o':
                        seen_o = 1;
                        if (strlen(arg) == 2) {
                                in_output = 1;
                                newae(ctx->i_ae, arg);
                        } else if (ctx->i_flags & CW_F_SHADOW) {
                                newae(ctx->i_ae, "-o");
                                newae(ctx->i_ae, discard_file_name(ctx, arg));
                                newae(ctx->i_ae, ctx->i_discard);
                        } else {
                                newae(ctx->i_ae, arg);
                        }
                        break;
                case 'c':
                case 'S':
                        if (strlen(arg) == 2)
                                op = CW_O_COMPILE;
                        newae(ctx->i_ae, arg);
                        break;
                case 'E':
                case 'P':
                        if (strlen(arg) == 2)
                                op = CW_O_PREPROCESS;
                /*FALLTHROUGH*/
                default:
                        newae(ctx->i_ae, arg);
                }
        }
        free(nameflag);

        /* See the comment on this same code in do_gcc() */
        if (c_files > 1 && op != CW_O_PREPROCESS) {
                errx(2, "multiple source files are "
                    "allowed only with -E or -P");
        }

        if (ctx->i_flags & CW_F_SHADOW) {
                if (op == CW_O_PREPROCESS)
        if ((op == CW_O_LINK || op == CW_O_PREPROCESS) &&
            (ctx->i_flags & CW_F_SHADOW))
                        exit(0);
                else if (op == CW_O_LINK && c_files == 0)
                        exit(0);
        }
#endif /* ! codereview */

        if (!seen_o && (ctx->i_flags & CW_F_SHADOW)) {
                newae(ctx->i_ae, "-o");
                newae(ctx->i_ae, discard_file_name(ctx, NULL));
                newae(ctx->i_ae, ctx->i_discard);
        }
}
______unchanged_portion_omitted_

static int
reap(cw_ictx_t *ctx)
{
        int status, ret = 0;
        char buf[1024];
        struct stat s;

        /*
         * Only wait for one specific child.
         */
        if (ctx->i_pid <= 0)
                return (-1);

        do {
                if (waitpid(ctx->i_pid, &status, 0) < 0) {
                        warn("cannot reap child");
                        return (-1);
                }
                if (status != 0) {
                        if (WIFSIGNALED(status)) {
                                ret = -WTERMSIG(status);
                                break;
                        } else if (WIFEXITED(status)) {
                                ret = WEXITSTATUS(status);
                                break;
                        }
                }
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));

        (void) unlink(ctx->i_discard);

        if (stat(ctx->i_stderr, &s) < 0) {
                warn("stat failed on child cleanup");
                return (-1);
        }
        if (s.st_size != 0) {
                FILE *f;

                if ((f = fopen(ctx->i_stderr, "r")) != NULL) {
```

```
                        while (fgets(buf, sizeof (buf), f))
                                (void) fprintf(stderr, "%s", buf);
                        (void) fflush(stderr);
                        (void) fclose(f);
                }
        }
        (void) unlink(ctx->i_stderr);
        free(ctx->i_stderr);

        /*
         * cc returns an error code when given -V; we want that to succeed.
         */
        if (ctx->i_flags & CW_F_PROG)
                return (0);

        return (ret);
}

static int
exec_ctx(cw_ictx_t *ctx, int block)
{
        if ((ctx->i_stderr = tempnam(ctx->i_tmpdir, "cw")) == NULL) {
        char *file;

        /*
         * To avoid offending cc's sensibilities, the name of its output
         * file must end in '.o'.
         */
        if ((file = tempnam(NULL, ".cw")) == NULL) {
                nomem();
                return (-1);
        }
        (void) strlcpy(ctx->i_discard, file, MAXPATHLEN);
        (void) strlcat(ctx->i_discard, ".o", MAXPATHLEN);
        free(file);

        if ((ctx->i_stderr = tempnam(NULL, ".cw")) == NULL) {
                nomem();
                return (-1);
        }

        if ((ctx->i_pid = fork()) == 0) {
                int fd;

                (void) fclose(stderr);
                if ((fd = open(ctx->i_stderr, O_WRONLY | O_CREAT | O_EXCL,
                    0666)) < 0) {
                        err(1, "open failed for standard error");
                }
                if (dup2(fd, 2) < 0) {
                        err(1, "dup2 failed for standard error");
                }
                if (fd != 2)
                        (void) close(fd);
                if (freopen("/dev/fd/2", "w", stderr) == NULL) {
                        err(1, "freopen failed for /dev/fd/2");
                }

                prepctx(ctx);
                exit(invoke(ctx));
        }

        if (ctx->i_pid < 0) {
                err(1, "fork failed");
        }
        if (block)
                return (reap(ctx));

        return (0);
}
```

*______unchanged_portion_omitted_*

```
static void
cleanup(cw_ictx_t *ctx)
{
        DIR *dirp;
        struct dirent *dp;
        char buf[MAXPATHLEN];

        if ((dirp = opendir(ctx->i_tmpdir)) == NULL) {
                if (errno != ENOENT) {
                        err(1, "couldn't open temp directory: %s",
                            ctx->i_tmpdir);
                } else {
                        return;
                }
        }

        errno = 0;
        while ((dp = readdir(dirp)) != NULL) {
                (void) snprintf(buf, MAXPATHLEN, "%s/%s", ctx->i_tmpdir,
                    dp->d_name);

                if (strncmp(dp->d_name, ".", strlen(dp->d_name)) == 0 ||
                    strncmp(dp->d_name, "..", strlen(dp->d_name)) == 0)
                        continue;

                if (unlink(buf) == -1)
                        err(1, "failed to unlink temp file: %s", dp->d_name);
                errno = 0;
        }

        if (errno != 0) {
                err(1, "failed to read temporary directory: %s",
                    ctx->i_tmpdir);
        }

        (void) closedir(dirp);
        if (rmdir(ctx->i_tmpdir) != 0) {
                err(1, "failed to unlink temporary directory: %s",
                    ctx->i_tmpdir);
        }
}

#endif /* ! codereview */
int
main(int argc, char **argv)
{
        int ch;
        cw_compiler_t primary = { NULL, NULL, 0 };
        cw_compiler_t shadows[10];
        int nshadows = 0;
        int ret = 0;
        boolean_t do_serial = B_FALSE;
        boolean_t do_exec = B_FALSE;
        boolean_t vflg = B_FALSE;
        boolean_t Cflg = B_FALSE;
        boolean_t cflg = B_FALSE;
        boolean_t nflg = B_FALSE;
        char *tmpdir;
#endif /* ! codereview */
```

```
        cw_ictx_t *main_ctx;

        static struct option longopts[] = {
                { "compiler", no_argument, NULL, 'c' },
                { "noecho", no_argument, NULL, 'n' },
                { "primary", required_argument, NULL, 'p' },
                { "shadow", required_argument, NULL, 's' },
                { "versions", no_argument, NULL, 'v' },
                { NULL, 0, NULL, 0 },
        };


        if ((main_ctx = newictx()) == NULL)
                nomem();

        while ((ch = getopt_long(argc, argv, "C", longopts, NULL)) != -1) {
                switch (ch) {
                case 'c':
                        cflg = B_TRUE;
                        break;
                case 'C':
                        Cflg = B_TRUE;
                        break;
                case 'n':
                        nflg = B_TRUE;
                        break;
                case 'p':
                        if (primary.c_path != NULL) {
                                warnx("Only one primary compiler may "
                                    "be specified");
                                usage();
                        }

                        parse_compiler(optarg, &primary);
                        break;
                case 's':
                        if (nshadows >= 10)
                                errx(1, "May only use 10 shadows at "
                                    "the moment");
                        parse_compiler(optarg, &shadows[nshadows]);
                        nshadows++;
                        break;
                case 'v':
                        vflg = B_TRUE;
                        break;
                default:
                        (void) fprintf(stderr, "Did you forget '--'?\n");
                        usage();
                }
        }

        if (primary.c_path == NULL) {
                warnx("A primary compiler must be specified");
                usage();
        }

        do_serial = (getenv("CW_SHADOW_SERIAL") == NULL) ? B_FALSE : B_TRUE;
        do_exec = (getenv("CW_NO_EXEC") == NULL) ? B_TRUE : B_FALSE;

        /* Leave room for argv[0] */
        argc -= (optind - 1);
        argv += (optind - 1);

        main_ctx->i_oldargc = argc;
        main_ctx->i_oldargv = argv;
        main_ctx->i_flags = CW_F_XLATE;
        if (nflg == 0)
                main_ctx->i_flags |= CW_F_ECHO;
        if (do_exec)
                main_ctx->i_flags |= CW_F_EXEC;
        if (Cflg)
                main_ctx->i_flags |= CW_F_CXX;
        main_ctx->i_compiler = &primary;

        if (cflg) {
                (void) fputs(primary.c_path, stdout);
        }

        if (vflg) {
                (void) printf("cw version %s\n", CW_VERSION);
                (void) fflush(stdout);
                main_ctx->i_flags &= ~CW_F_ECHO;
                main_ctx->i_flags |= CW_F_PROG | CW_F_EXEC;
                do_serial = 1;
        }

        tmpdir = getenv("TMPDIR");
        if (tmpdir == NULL)
                tmpdir = "/tmp";

        if (asprintf(&main_ctx->i_tmpdir, "%s/cw.XXXXXX", tmpdir) == -1)
                nomem();

        if ((main_ctx->i_tmpdir = mkdtemp(main_ctx->i_tmpdir)) == NULL)
                errx(1, "failed to create temporary directory");

#endif /* ! codereview */
        ret |= exec_ctx(main_ctx, do_serial);

        for (int i = 0; i < nshadows; i++) {
                int r;
                cw_ictx_t *shadow_ctx;

                if ((shadow_ctx = newictx()) == NULL)
                        nomem();

                (void) memcpy(shadow_ctx, main_ctx, sizeof (cw_ictx_t));
                memcpy(shadow_ctx, main_ctx, sizeof (cw_ictx_t));

                shadow_ctx->i_flags |= CW_F_SHADOW;
                shadow_ctx->i_compiler = &shadows[i];

                r = exec_ctx(shadow_ctx, do_serial);
                if (r == 0) {
                        shadow_ctx->i_next = main_ctx->i_next;
                        main_ctx->i_next = shadow_ctx;
                }
                ret |= r;
        }

        if (!do_serial) {
                cw_ictx_t *next = main_ctx;
                while (next != NULL) {
                        cw_ictx_t *toreap = next;
                        next = next->i_next;
                        ret |= reap(toreap);
                }
        }

        cleanup(main_ctx);
#endif /* ! codereview */
```

```
        return (ret);
}
```