

```

new/usr/src/tools/cw/Makefile
*****
2010 Tue Jan 15 20:51:36 2019
new/usr/src/tools/cw/Makefile
9899 cw(lonbld) should shadow more compilation
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2007 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 # Copyright 2018 Joyent, Inc.

27 PROG      = cw

29 MAN1ONBLDFILES= cw.lonbld

31 include ../Makefile.tools

33 # Bootstrap problem -- we have to build cw before we can use it
34 i386_CC=          $(SPRO_VROOT)/bin/cc
35 sparc_CC=          $(SPRO_VROOT)/bin/cc
36 $(__GNUC)i386_CC= $(GNUC_ROOT)/bin/gcc
37 $(__GNUC)sparc_CC= $(GNUC_ROOT)/bin/gcc

39 CFLAGS += $(CCVERBOSE)

41 # Override CFLAGS. This is needed only for bootstrap of cw.
42 $(__GNUC)CFLAGS=   -O -D_sun -Wall -Wno-unknown-pragmas -Werror \
43                         -std=gnu99 -nodefaultlibs
44 $(__SUNC)CFLAGS=    -xspace -Xa -xildoff -errtags=yes -errwarn=%all \
45                         -xc99=%all -W0,-xglobalstatic -v

48 $(__GNUC)LDLIBS += -lc
49 $(__GNUC)LDFLAGS=  $(MAPFILE.NES:%=-Wl,-M%)

51 $(ROOTONBLDMAN1ONBLDFILES) := FILEMODE=       644
52 CSTD=      $(CSTD_GNU99)
53 #endif /* ! codereview */

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

58 .KEEP_STATE:

60 all: $(PROG) $(MAN1ONBLDFILES)

```

```

1           new/usr/src/tools/cw/Makefile
2
3           62 install: all .WAIT $(ROOTONBLDMACHPROG) $(ROOTONBLDMAN1ONBLDFILES)
4           64 lint: lint_PROG
5           66 clean:
6           68 #
7           69 # Not run by default: bootstrap...
8           70 check:
9           71     $(ROOTONBLDBINMACH)/mandoc -Tlint -Wwarning $(MAN1ONBLDFILES)
10          73 include ../Makefile.targ

```

new/usr/src/tools/cw/cw.lonbld

```
*****
6241 Tue Jan 15 20:51:36 2019
new/usr/src/tools/cw/cw.lonbld
9899 cw(lonbld) should shadow more compilation
*****
1 .\""
2 .\" CDDL HEADER START
3 .\""
4 .\" The contents of this file are subject to the terms of the
5 .\" Common Development and Distribution License (the "License").
6 .\" You may not use this file except in compliance with the License.
7 .\""
8 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 .\" or http://www.opensolaris.org/os/licensing.
10 .\" See the License for the specific language governing permissions
11 .\" and limitations under the License.
12 .\""
13 .\" When distributing Covered Code, include this CDDL HEADER in each
14 .\" file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 .\" If applicable, add the following below this CDDL HEADER, with the
16 .\" fields enclosed by brackets "[]" replaced with your own identifying
17 .\" information: Portions Copyright [yyyy] [name of copyright owner]
18 .\""
19 .\" CDDL HEADER END
20 .\""
21 .\" Copyright 2010 Sun Microsystems, Inc. All rights reserved.
22 .\" Use is subject to license terms.
23 .\""
24 .\" Copyright 2018 Joyent, Inc.
25 .\""
26 .Dd September 4, 2018
27 .Dt CW LONBLD
28 .Os
29 .Sh NAME
30 .Nm cw
31 .Nd invoke one or more compilers with argument translation
32 .Sh SYNOPSIS
33 .Nm cw
34 .Op Fl C
35 .Op Fl -versions
36 .Op Fl -noecho
37 .Fl -primary Ar compiler
38 .Op Fl -shadow Ar compiler ...
39 .Fl -
40 .Ar compiler args ...
41 .Sh DESCRIPTION
42 .Nm cw
43 is a facility for invoking one or more compilers, providing translation from
44 Sun style arguments as appropriate.
45 This allows the use of arbitrary compilers without the need to alter large
46 numbers of makefiles.
47 A mode called shadow compilation invokes multiple compilers so that warnings
48 and errors may be obtained from all of them.
49 See
50 .Sx SHADOW COMPILATION
51 for details.
52 This version of cw supports compilers with both Sun Studio 12 and GCC-style
53 command lines.
54 .Sh ARGUMENTS
55 Both the
56 .Fl -primary
57 and
58 .Fl -shadow
59 parameters take a
60 .Em compiler specification .
61 This is a comma-separated list of the form
```

1

new/usr/src/tools/cw/cw.lonbld

```
62 .Ar name,executable,style
63 Where
64 .Ar name
65 is a name for the compiler,
66 .Ar executable
67 is the full path to the compiler executable, and
68 .Ar style
69 is the style of command-line options the compiler expects, either
70 .Em sun
71 or
72 .Em gnu.
73 .Bl -tag -width indent
74 .It Fl -primary Ar compiler
75 Specify the compiler to be used primarily (that which is used for link-editing
76 and pre-processing, and whos objects we deliver).
77 .It Fl -shadow Ar compiler
78 Specify a shadow compiler, which builds sources for the sake of checking code
79 quality and compatibility, but has its output discarded.
80 .It Fl -noecho
81 Do not echo the actual command line of any compilers invoked.
82 .It Fl -versions
83 Request from each configured primary and shadow compiler its version
84 information.
85 .It Fl C
86 The sources being compiled are C++. This is necessary as it affects the
87 translation of compiler arguments.
88 .It Fl -
89 Arguments intended for the compilers themselves must be separated from those
90 of
91 .Nm cw
92 by a
93 .Fl -
94 .It Fl _name=
95 .It Fl _style=
96 Parameters intended for the compiler be guarded with options of the form
97 .Fl _name=
98 and
99 .Fl _style=
100 Where
101 .Em name
102 and
103 .Em style
104 are those passed to
105 .Fl -primary
106 and
107 .Fl -shadow
108 this allows certain flags to be passed only to certain classes of compiler.
109 .Pp
110 For historical reasons, the
111 .Fl _style=
112 option is also translated such that a style of
113 .Em sun
114 may use the flag
115 .Fl _cc=
116 and a style of
117 .Em gnu
118 may use the flag
119 .Fl _gcc=
120 and when the
121 .Fl C
122 option is given and C++ is in use the style of
123 .Em sun
124 may use the flag
125 .Fl _CC=
126 and the style of
127 .Em gnu
```

2

```

128 may use the flag
129 .Fl _g++=
130 .El
131 .Sh SHADOW COMPILATION
132 If
133 .Fl -shadow
134 compilers are specified
135 .Nm cw
136 will invoke each shadow compiler, with the outputs modified (as well as any
137 translation for compiler style) as follows:
138 .Bl -enum
139 .It
140 If
141 .Nm cw
142 is invoked to link-edit without compilation (the input files are all objects),
143 the shadow compiler is not invoked.
140 If neither of
141 .Fl c '
142 .Fl S
143 appears in the argument list (that is, linking is attempted or only the
144 pre-processor is invoked), the shadow compilers will not be invoked.
144 .It
145 If the
146 .Fl o Ar filename
147 option was provided, with or without a separating space, it will be replaced with
148 .Fl o Ar tempfile
149 .It
150 If the option
151 .Fl o
152 was not provided,
153 .Fl o Ar tempfile
154 will be added to the end of the argument list used to invoke
155 the shadow compilers.
156 .El
157 When shadow compilation is in effect,
158 .Nm cw
159 writes to standard error each compiler's standard error output following its
160 argument list.
161 Messages from the compilers will not be interleaved.
162 If
163 .Nm cw
164 is used to invoke the preprocessor and no output location is specified,
165 .Nm cw
166 will write to standard output the primary compiler's standard output.
167 .Pp
168 Because the Sun compilers write intermediate objects to fixed
169 filenames in the current directory when instructed to compile and
170 link multiple source files via a single command line, it would be
171 unsafe to invoke more than one compiler in this fashion.
172 Therefore
173 .Nm cw
174 does not accept multiple source files unless the preprocessor is to be
175 invoked.
176 An attempt to invoke
177 .Nm cw
178 in this manner will result in an error.
179 .Sh ARGUMENT TRANSLATION
180 If the compiler to be invoked is a GNU-style C or C++ compiler, a set of
181 default flags is added to the beginning of the argument list, and the
182 remaining arguments are translated to their closest appropriate
183 semantic equivalents and passed in the same order as their
184 counterparts given to
185 .Nm cw .
186 See the comments at the head of
187 .Pa usr/src/tools/cw/cw.c
188 for a detailed list of translations.

```

```

189 .Sh ENVIRONMENT
190 .Bl -tag -width indent
191 .It CW_SHADOW_SERIAL
192 If this variable is set in the environment, invoke the primary compiler, wait
193 for it to complete, then invoke the shadow compilers.
194 Normally the primary and shadow compilers are invoked in parallel.
195 .It CW_NO_EXEC
196 If this variable is set in the environment, write the usual output to
197 standard error but do not actually invoke any compiler.
198 This is useful for debugging the translation engine.
199 .El
200 .Sh EXIT STATUS
201 The following exit status values are returned:
202 .Bl -tag -width indent
203 .It 0
204 The primary compiler, and shadow compilers if invoked, all completed
205 successfully.
206 .It >0
207 A usage error occurred, or one or more compilers returned a nonzero
208 exit status.
209 .El
210 .Sh SEE ALSO
211 .Xr cc 1 ,
212 .Xr CC 1 ,
213 .Xr gcc 1
214 .Sh BUGS
215 The translations provided for gcc are not always exact and in some cases
216 reflect local policy rather than actual equivalence.
217 .Pp
218 Additional compiler types should be supported.
219 .Pp
220 The translation engine is hacky.

```

```
*****
43870 Tue Jan 15 20:51:37 2019
new/usr/src/tools/cw/cw.c
9899 cw(1onbld) should shadow more compilation
*****
```

```

2 /*
3 * CDDL HEADER START
4 *
5 * The contents of this file are subject to the terms of the
6 * Common Development and Distribution License (the "License").
7 * You may not use this file except in compliance with the License.
8 *
9 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */

23 /*
24 * Copyright 2018, Richard Lowe.
25 */
26 /*
27 * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
28 * Use is subject to license terms.
29 */
30 * Copyright 2019 Joyent, Inc.
31 */

33 /*
34 * Wrapper for the GNU C compiler to make it accept the Sun C compiler
35 * arguments where possible.
36 *
37 * Since the translation is inexact, this is something of a work-in-progress.
38 */
39 */

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

44 /*
45 * -#           Verbose mode
46 * -###        Show compiler commands built by driver, no compilation
47 * -A<name[(tokens)]> Preprocessor predicate assertion
48 * -B<[static|dynamic]> Specify dynamic or static binding
49 * -C           Prevent preprocessor from removing comments
50 * -c           Compile only - produce .o files, suppress linking
51 * -cg92        Alias for -xtarget=ss1000
52 * -D<name[=token]> Associate name with token as if by #define
53 * -d[y|n]      dynamic [-dy] or static [-dn] option to linker
54 * -E           Compile source through preprocessor only, output to stdout
55 * -erroff=<t> Suppress warnings specified by tags t(%none, %all, <tag list>)
56 * -errtags=<a> Display messages with tags a(no, yes)
57 * -errwarn=<t> Treats warnings specified by tags t(%none, %all, <tag list>)
58 *               as errors
59 * -fast         Optimize using a selection of options
60 * -fd           Report old-style function definitions and declarations
61 * -fnonstd     Initialize floating-point hardware to non-standard preferences

```

```

62 * -fns[=<yes|no>] Select non-standard floating point mode
63 * -fprecision=<p> Set FP rounding precision mode p(single, double, extended)
64 * -fround=<r> Select the IEEE rounding mode in effect at startup
65 * -fsimple[=<n>] Select floating-point optimization preferences <n>
66 * -fsingle   Use single-precision arithmetic (-Xt and -Xs modes only)
67 * -ftrap=<t> Select floating-point trapping mode in effect at startup
68 * -fstore    force floating pt. values to target precision on assignment
69 * -G          Build a dynamic shared library
70 * -g          Compile for debugging
71 * -H          Print path name of each file included during compilation
72 * -h <name>   Assign <name> to generated dynamic shared library
73 * -I<dir>    Add <dir> to preprocessor #include file search path
74 * -i          Passed to linker to ignore any LD_LIBRARY_PATH setting
75 * -keeptmp   Keep temporary files created during compilation
76 * -L<dir>    Pass to linker to add <dir> to the library search path
77 * -l<name>   Link with library lib<name>.a or lib<name>.so
78 * -mc         Remove duplicate strings from .comment section of output files
79 * -mr         Remove all strings from .comment section of output files
80 * -mr,"string" Remove all strings and append "string" to .comment section
81 * -mt         Specify options needed when compiling multi-threaded code
82 * -native     Find available processor, generate code accordingly
83 * -nofstore   Do not force floating pt. values to target precision
84 *             on assignment
85 * -norunpath  Do not build in a runtime path for shared libraries
86 * -O          Use default optimization level (-xO2 or -xO3. Check man page.)
87 * -o <outputfile> Set name of output file to <outputfile>
88 * -P          Compile source through preprocessor only, output to .i file
89 * -p          Compile for profiling with prof
90 * -Q[y|n]     Emit/don't emit identification info to output file
91 * -R<dir[:dir]> Build runtime search path list into executable
92 * -S          Compile and only generate assembly code (.s)
93 * -s          Strip symbol table from the executable file
94 * -t          Turn off duplicate symbol warnings when linking
95 * -U<name>   Delete initial definition of preprocessor symbol <name>
96 * -V          Report version number of each compilation phase
97 * -v          Do stricter semantic checking
98 * -W<c>,<arg> Pass <arg> to specified component <c> (a,l,m,p,o,2,h,i,u)
99 * -w          Suppress compiler warning messages
100 * -XA         Compile assuming ANSI C conformance, allow K & R extensions
101 *             (default mode)
102 * -xs         Compile assuming (pre-ANSI) K & R C style code
103 * -xt         Compile assuming K & R conformance, allow ANSI C
104 * -xarch=<a>  Specify target architecture instruction set
105 * -xbuiltin[=<b>] When profitable inline, or substitute intrinsic functions
106 *               for system functions, b=%all,%none
107 * -xCC         Accept C++ style comments
108 * -xchip=<c>  Specify the target processor for use by the optimizer
109 * -xcode=<c>  Generate different code for forming addresses
110 * -xcrossfile[=<n>] Enable optimization and inlining across source files,
111 *               n={0|1}
112 * -xe         Perform only syntax/semantic checking, no code generation
113 * -xF         Compile for later mapfile reordering or unused section
114 *               elimination
115 * -xhelp=<f>  Display on-line help information f(flags, readme, errors)
116 * -xildoff   Cancel -xildon
117 * -xildon    Enable use of the incremental linker, ild
118 * -xinline=[<a>, ..., <a>] Attempt inlining of specified user routines,
119 *               <a>=%auto,func,no%func
120 * -xlibmieee Force IEEE 754 return values for math routines in
121 *               exceptional cases
122 * -xlibmil   Inline selected libm math routines for optimization
123 * -xlic_lib=sunperf Link in the Sun supplied performance libraries
124 * -xlicinfo   Show license server information
125 * -xmaxopt=[off,1,2,3,4,5] maximum optimization level allowed on #pragma opt
126 * -xO<n>     Generate optimized code (n={1|2|3|4|5})
127 * -xP         Print prototypes for function definitions

```

```

128 * -xprofile=<p> Collect data for a profile or use a profile to optimize
129 * <p>=&{collect,use}[:<path>],tcov}
130 * -xregs=<r> Control register allocation
131 * -xs Allow debugging without object (.o) files
132 * -xsb Compile for use with the WorkShop source browser
133 * -xsbfast Generate only WorkShop source browser info, no compilation
134 * -xsfpcnst Represent unsuffixed floating point constants as single
135 * precision
136 * -xspace Do not do optimizations that increase code size
137 * -xstrconst Place string literals into read-only data segment
138 * -xtarget=<t> Specify target system for optimization
139 * -xtemp=<dir> Set directory for temporary files to <dir>
140 * -xtime Report the execution time for each compilation phase
141 * -xunroll=n Enable unrolling loops n times where possible
142 * -Y<c>,<dir> Specify <dir> for location of component <c> (a,l,m,p,o,h,i,u)
143 * -YA,<dir> Change default directory searched for components
144 * -YL,<dir> Change default directory searched for include files
145 * -VP,<dir> Change default directory for finding libraries files
146 * -YS,<dir> Change default directory for startup object files
147 */

149 /*
150 * Translation table:
151 */
152 /*
153 * -#
154 * -###
155 * -A<name[(tokens)]>
156 * -B<[static|dynamic]>
157 * -C
158 * -c
159 * -cg92
160 * -D<name[=token]>
161 * -dy or -dn
162 * -E
163 * -erroff=E_EMPTY_TRANSLATION_UNIT ignore
164 * -errtags=%all
165 * -errwarn=%all
166 * -fast
167 * -fd
168 * -fnonstd
169 * -fnfs[=<yes|no>]
170 * -fprecision=<p>
171 * -fround=<r>
172 * -fsimple[=<n>]
173 * -fsingle[=<n>]
174 * -ftrap=<t>
175 * -fstore
176 * -G
177 * -g
178 * -H
179 * -h <name>
180 * -I<dir>
181 * -i
182 * -keepptmp
183 * -l<dir>
184 * -l<name>
185 * -mc
186 * -mr
187 * -mr,"string"
188 * -mt
189 * -native
190 * -nofstore
191 * -nolib
192 * -norunpath
193 * -O

```

```

194 * -o <outputfile> pass-thru
195 * -P -E -o filename.i (or error)
196 * -p pass-thru
197 * -Q[y|n] error
198 * -R<dir[:dir]> pass-thru
199 * -S pass-thru
200 * -s -Wl,-s
201 * -t -Wl,-t
202 * -U<name> pass-thru
203 * -V --version
204 * -v -Wall
205 * -Wa,<arg> pass-thru
206 * -Wp,<arg> pass-thru except -xc99=<a>
207 * -Wl,<arg> pass-thru
208 * -W{m,0,2,h,i,u} error/ignore
209 * -xmodel=kernel -ffreestanding -mcmodel=kernel -mno-red-zone
210 * -Wu,-save_args -msave-args
211 * -w pass-thru
212 * -Xa -std=iso9899:199409 or -ansi
213 * -Xt error
214 * -Xs -traditional -std=c89
215 * -xarch=<a> table
216 * -xbuiltin[=<b>] -fbuiltin (-fno-builtins otherwise)
217 * -xCC ignore
218 * -xchip=<c> table
219 * -xcode=<c> table
220 * -xdebugformat=<format> ignore (always use dwarf-2 for gcc)
221 * -xcrossfile[=<n>] ignore
222 * -xe error
223 * -xF error
224 * -xhelp=<f> error
225 * -xildoff ignore
226 * -xildon ignore
227 * -xinline ignore
228 * -xlibmieee error
229 * -xlibmil error
230 * -xlic_lib=sunperf error
231 * -xmaxopt=[...] error
232 * -xO<n> -O<n>
233 * -xP error
234 * -xprofile=<p> error
235 * -xregs=<r> table
236 * -xs error
237 * -xsb error
238 * -xsbfast error
239 * -xsfpconst error
240 * -xspace ignore (-not -Os)
241 * -xstrconst ignore
242 * -xtarget=<t> table
243 * -xtemp=<dir> error
244 * -xtime error
245 * -xtransition -Wtransition
246 * -xunroll=n error
247 * -W0,-xdbggen=no%usedonly -fno-eliminate-unused-debug-symbols
248 * -Y<c>,<dir> -fno-eliminate-unused-debug-types
249 * -YA,<dir> error
250 * -YP,<dir> error
251 * -YI,<dir> -nostdinc -I<dir>
252 * -YS,<dir> error
253 * -/ error

256 #include <ctype.h>
257 #include <err.h>
258 #include <errno.h>
259 #include <fcntl.h>

```

```

260 #include <getopt.h>
261 #include <stdio.h>
262 #include <stdlib.h>
263 #include <string.h>
264 #include <unistd.h>
265 #include <dirent.h>
266 #endif /* ! codereview */

268 #include <sys/param.h>
269 #include <sys/stat.h>
270 #include <sys/types.h>
271 #include <sys/utsname.h>
272 #include <sys/wait.h>

274 #define CW_F_CXX      0x01
275 #define CW_F_SHADOW   0x02
276 #define CW_F_EXEC     0x04
277 #define CW_F_ECHO     0x08
278 #define CW_F_XLATE    0x10
279 #define CW_F_PROG    0x20

281 typedef enum cw_op {
282     CW_O_NONE = 0,
283     CW_O_PREPROCESS,
284     CW_O_COMPILE,
285     CW_O_LINK
286 } cw_op_t;

288 struct aelist {
289     struct ae {
290         struct ae *ae_next;
291         char *ae_arg;
292     } *ael_head, *ael_tail;
293     int ael_argc;
294 };

296 typedef enum {
297     GNU,
298     SUN,
299     SMATCH
300 } compiler_style_t;

302 typedef struct {
303     char *c_name;
304     char *c_path;
305     compiler_style_t c_style;
306 } cw_compiler_t;

308 typedef struct cw_ictx {
309     struct cw_ictx *i_next;
310     cw_compiler_t *i_compiler;
311     struct aelist *i_ae;
312     uint32_t i_flags;
313     int i_oldargc;
314     char **i_oldargv;
315     pid_t i_pid;
316     char *i_tmpdir;
317     char i_discard[MAXPATHLEN];
318     char *i_stderri;
319 } cw_ictx_t;
320 unchanged_portion_omitted_
321 */
322 * The compiler wants the output file to end in appropriate extension. If
323 * we're generating a name from whole cloth (path == NULL), we assume that
324 * extension to be .o, otherwise we match the extension of the caller.

```

```

565 */
566 static char *
567 discard_file_name(cw_ictx_t *ctx, const char *path)
568 {
569     char *ret, *ext;
570     char tmpl[] = "cwXXXXXX";
571
572     if (path == NULL) {
573         ext = ".o";
574     } else {
575         ext = strrchr(path, '.');
576     }
577
578     /*
579     * We need absolute control over where the temporary file goes, since
580     * we rely on it for cleanup so tmpnam(3C) and tmpnam(3C) are
581     * inappropriate (they use TMPDIR, preferentially).
582     *
583     * mkstemp(3C) doesn't actually help us, since the temporary file
584     * isn't used by us, only its name.
585     */
586     if (mktemp(tmpl) == NULL)
587         nomem();
588
589     (void) asprintf(&ret, "%s/%s", ctx->i_tmpdir, tmpl,
590                     (ext != NULL) ? ext : "");
591
592     if (ret == NULL)
593         nomem();
594
595     return (ret);
596 }

597 #endif /* ! codereview */
598 static void
600 do_gcc(cw_ictx_t *ctx)
601 {
602     int c;
603     int nolibc = 0;
604     int in_output = 0, seen_o = 0, c_files = 0;
605     cw_op_t op = CW_O_LINK;
606     char *model = NULL;
607     char *nameflag;
608     int mflag = 0;
609
610     if (ctx->i_flags & CW_F_PROG) {
611         newae(ctx->i_ae, "--version");
612         return;
613     }
614
615     newae(ctx->i_ae, "-fident");
616     newae(ctx->i_ae, "-finline");
617     newae(ctx->i_ae, "-fno-inline-functions");
618     newae(ctx->i_ae, "-fno-builtin");
619     newae(ctx->i_ae, "-fno-asm");
620     newae(ctx->i_ae, "-fdiagnostics-show-option");
621     newae(ctx->i_ae, "-nodefaultlibs");
622
623 #if defined(__sparc)
624     /*
625     * The SPARC ldd and std instructions require 8-byte alignment of
626     * their address operand.  gcc correctly uses them only when the
627     * ABI requires 8-byte alignment; unfortunately we have a number of
628     * pieces of buggy code that doesn't conform to the ABI.  This
629     * flag makes gcc work more like Studio with -xmemalign=4.
630     */

```

```

631     newae(ctx->i_ae, "-mno-integer-lld-std");
632 #endif
633
634     /*
635      * This is needed because 'u' is defined
636      * under a conditional on 'sun'. Should
637      * probably just remove the conditional,
638      * or make it be dependent on '_sun'.
639      *
640      * -Dunix is also missing in enhanced ANSI mode
641      */
642     newae(ctx->i_ae, "-D_sun");
643
644     if (asprintf(&nameflag, "-%s=", ctx->i_compiler->c_name) == -1)
645         nomem();
646
647     /*
648      * Walk the argument list, translating as we go ..
649      */
650     while (--ctx->i_oldargc > 0) {
651         char *arg = *++ctx->i_oldargv;
652         size_t arglen = strlen(arg);
653
654         if (*arg == '-') {
655             arglen--;
656         } else {
657             /*
658              * Discard inline files that gcc doesn't grok
659              */
660             if (!in_output && arglen > 3 &&
661                 strcmp(arg + arglen - 3, ".il") == 0)
662                 continue;
663
664             if (!in_output && arglen > 2 &&
665                 arg[arglen - 2] == '.' &&
666                 (arg[arglen - 1] == 's' || arg[arglen - 1] == 'S' ||
667                  arg[arglen - 1] == 'c' || arg[arglen - 1] == 'i'))
668                 c_files++;
669
670             /*
671              * Otherwise, filenames and partial arguments
672              * are passed through for gcc to chew on. However,
673              * output is always discarded for the secondary
674              * compiler.
675              */
676             if ((ctx->i_flags & CW_F_SHADOW) && in_output) {
677                 newae(ctx->i_ae, discard_file_name(ctx, arg));
678             } else {
679                 if ((ctx->i_flags & CW_F_SHADOW) && in_output)
680                     newae(ctx->i_ae, ctx->i_discard);
681                 else
682                     newae(ctx->i_ae, arg);
683             }
684 #endif /* ! codereview */
685             in_output = 0;
686             continue;
687         }
688
689         if (ctx->i_flags & CW_F_CXX) {
690             if (strncmp(arg, "-_g++=", 6) == 0) {
691                 newae(ctx->i_ae, strchr(arg, '=') + 1);
692                 continue;
693             }
694             if (strncmp(arg, "-compat=", 8) == 0) {
695                 /* discard -compat=4 and -compat=5 */
696                 continue;
697             }
698         }
699     }
700
701     continue;
702 }
703
704     if (strcmp(arg, "-Qoption") == 0) {
705         /*
706          * discard -Qoption and its two arguments */
707         if (ctx->i_oldargc < 3)
708             error(arg);
709         ctx->i_oldargc -= 2;
710         ctx->i_oldargv += 2;
711         continue;
712     }
713     if (strcmp(arg, "-xwe") == 0) {
714         /*
715          * turn warnings into errors */
716         newae(ctx->i_ae, "-Werror");
717         continue;
718     }
719     if (strcmp(arg, "-norunpath") == 0) {
720         /*
721          * gcc has no corresponding option */
722         continue;
723     }
724     if (strcmp(arg, "-nolib") == 0) {
725         /*
726          * -nodefaultlibs is on by default */
727         nolibc = 1;
728         continue;
729     }
730     #if defined(__sparc)
731     if (strcmp(arg, "-cg92") == 0) {
732         mflag |= xlate_xtb(ctx->i_ae, "v8");
733         xlate(ctx->i_ae, "super", xchip_tbl);
734         continue;
735     }
736     #endif /* __sparc */
737
738     switch ((c = arg[1])) {
739     case '_':
740         if ((strncmp(arg, nameflag, strlen(nameflag)) == 0) ||
741             (strncmp(arg, "-_gcc=", 6) == 0) ||
742             (strncmp(arg, "-_gnu=", 6) == 0)) {
743             newae(ctx->i_ae, strchr(arg, '=') + 1);
744         }
745         break;
746     case '#':
747         if (arglen == 1) {
748             newae(ctx->i_ae, "-v");
749             break;
750         }
751         error(arg);
752         break;
753     case 'f':
754         if ((strcmp(arg, "-fpic") == 0) ||
755             (strcmp(arg, "-fPIC") == 0)) {
756             newae(ctx->i_ae, arg);
757             break;
758         }
759         error(arg);
760         break;
761     case 'g':
762         newae(ctx->i_ae, "-gdwarf-2");
763         break;
764     case 'E':
765         if (arglen == 1) {
766             newae(ctx->i_ae, "-xc");
767             newae(ctx->i_ae, arg);
768             op = CW_O_PREPROCESS;
769             nolibc = 1;
770             break;
771         }
772     }
773 }
```

```

694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
267
```

```

760             error(arg);
761             break;
762         case 'c':
763             if (arglen == 1) {
764                 op = CW_O_COMPILE;
765                 nolibc = 1;
766             }
767             /* FALLTHROUGH */
768         case 'C':
769         case 'H':
770         case 'p':
771             if (arglen == 1) {
772                 newae(ctx->i_ae, arg);
773                 break;
774             }
775             error(arg);
776             break;
777         case 'A':
778         case 'h':
779         case 'I':
780         case 'i':
781         case 'L':
782         case 'l':
783         case 'R':
784         case 'U':
785         case 'u':
786         case 'w':
787             newae(ctx->i_ae, arg);
788             break;
789         case 'o':
790             seen_o = 1;
791             if (arglen == 1) {
792                 in_output = 1;
793                 newae(ctx->i_ae, arg);
794             } else if (ctx->i_flags & CW_F_SHADOW) {
795                 newae(ctx->i_ae, "-o");
796                 newae(ctx->i_ae, discard_file_name(ctx, arg));
797                 newae(ctx->i_ae, ctx->i_discard);
798             } else {
799                 newae(ctx->i_ae, arg);
800             }
801             break;
802         case 'D':
803             newae(ctx->i_ae, arg);
804             /*
805             * XXX Clearly a hack ... do we need _KADB too?
806             */
807             if (strcmp(arg, "-D_KERNEL") == 0 || 
808                 strcmp(arg, "-D_BOOT") == 0)
809                 newae(ctx->i_ae, "-ffreestanding");
810             break;
811         case 'd':
812             if (arglen == 2) {
813                 if (strcmp(arg, "-dy") == 0) {
814                     newae(ctx->i_ae, "-Wl,-dy");
815                     break;
816                 }
817                 if (strcmp(arg, "-dn") == 0) {
818                     newae(ctx->i_ae, "-Wl,-dn");
819                     break;
820                 }
821             }
822             if (strcmp(arg, "-dalign") == 0) {
823                 /*
824                  * -dalign forces alignment in some cases;

```

```

825             * gcc does not need any flag to do this.
826             */
827             break;
828         }
829         error(arg);
830         break;
831     case 'e':
832         if (strcmp(arg, "-erroff=E_EMPTY_TRANSLATION_UNIT") == 0) {
833             /*
834             * Accept but ignore this -- gcc doesn't
835             * seem to complain about empty translation
836             * units
837             */
838             break;
839         }
840         /* XX64 -- ignore all -erroff= options, for now */
841         if (strncmp(arg, "-erroff=", 8) == 0)
842             break;
843         if (strcmp(arg, "-errtags=yes") == 0) {
844             warnings(ctx->i_ae);
845             break;
846         }
847         if (strcmp(arg, "-errwarn=%all") == 0) {
848             newae(ctx->i_ae, "-Werror");
849             break;
850         }
851         error(arg);
852         break;
853     case 'G':
854         newae(ctx->i_ae, "-shared");
855         nolibc = 1;
856         break;
857     case 'k':
858         if (strcmp(arg, "-keeptmp") == 0) {
859             newae(ctx->i_ae, "-save-temp");
860             break;
861         }
862         error(arg);
863         break;
864     case 'm':
865         if (strcmp(arg, "-mt") == 0) {
866             newae(ctx->i_ae, "-D_REENTRANT");
867             break;
868         }
869         if (strcmp(arg, "-m64") == 0) {
870             newae(ctx->i_ae, "-m64");
871             newae(ctx->i_ae, "-mtune=opteron");
872             #if defined(__x86)
873             #endif
874             mflag |= M64;
875             break;
876         }
877         if (strcmp(arg, "-m32") == 0) {
878             newae(ctx->i_ae, "-m32");
879             mflag |= M32;
880             break;
881         }
882         error(arg);
883         break;
884     case 'B': /* linker options */
885     case 'M':
886     case 'z':
887         {
888             char *opt;
889             size_t len;

```

```

891         char *s;
892
893         if (arglen == 1) {
894             opt = *++ctx->i_oldargv;
895             if (opt == NULL || *opt == '\0')
896                 error(arg);
897             ctx->i_oldargc--;
898         } else {
899             opt = arg + 2;
900         }
901         len = strlen(opt) + 7;
902         if ((s = malloc(len)) == NULL)
903             nomem();
904         (void) sprintf(s, len, "-Wl,-%c%s", c, opt);
905         newae(ctx->i_ae, s);
906         free(s);
907     }
908     break;
909 case 'O':
910     if (arglen == 1) {
911         newae(ctx->i_ae, "-O");
912         break;
913     }
914     error(arg);
915     break;
916 case 'P':
917     /*
918      * We could do '-E -o filename.i', but that's hard,
919      * and we don't need it for the case that's triggering
920      * this addition. We'll require the user to specify
921      * -o in the Makefile. If they don't they'll find out
922      * in a hurry.
923      */
924     newae(ctx->i_ae, "-E");
925     op = CW_O_PREPROCESS;
926     nolibc = 1;
927     break;
928 case 's':
929     if (arglen == 1) {
930         newae(ctx->i_ae, "-Wl,-s");
931         break;
932     }
933     error(arg);
934     break;
935 case 't':
936     if (arglen == 1) {
937         newae(ctx->i_ae, "-Wl,-t");
938         break;
939     }
940     error(arg);
941     break;
942 case 'V':
943     if (arglen == 1) {
944         ctx->i_flags &= ~CW_F_ECHO;
945         newae(ctx->i_ae, "--version");
946         break;
947     }
948     error(arg);
949     break;
950 case 'v':
951     if (arglen == 1) {
952         warnings(ctx->i_ae);
953         break;
954     }
955     error(arg);
956     break;

```

```

957         case 'W':
958             if (strcmp(arg, "-Wp,-xc99", 9) == 0) {
959                 /*
960                  * gcc's preprocessor will accept c99
961                  * regardless, so accept and ignore.
962                  */
963                 break;
964             }
965             if (strcmp(arg, "-Wa,", 4) == 0 ||
966                 strcmp(arg, "-Wp,", 4) == 0 ||
967                 strcmp(arg, "-Wl,", 4) == 0) {
968                 newae(ctx->i_ae, arg);
969                 break;
970             }
971             if (strcmp(arg, "-W0,-noglobal") == 0 ||
972                 strcmp(arg, "-W0,-xglobalstatic") == 0) {
973                 /*
974                  * gcc doesn't prefix local symbols
975                  * in debug mode, so this is not needed.
976                  */
977                 break;
978             }
979             if (strcmp(arg, "-W0,-Lt") == 0) {
980                 /*
981                  * Generate tests at the top of loops.
982                  * There is no direct gcc equivalent, ignore.
983                  */
984                 break;
985             }
986             if (strcmp(arg, "-W0,-xdbggen=no%usedonly") == 0) {
987                 newae(ctx->i_ae,
988                       "-fno-eliminate-unused-debug-symbols");
989                 newae(ctx->i_ae,
990                       "-fno-eliminate-unused-debug-types");
991                 break;
992             }
993             if (strcmp(arg, "-W2,-xwrap_int") == 0) {
994                 /*
995                  * Use the legacy behaviour (pre-SS11)
996                  * for integer wrapping.
997                  * gcc does not need this.
998                  */
999                 break;
1000            }
1001            if (strcmp(arg, "-Wd,-xsafe=unboundsym") == 0) {
1002                 /*
1003                  * Prevents optimizing away checks for
1004                  * unbound weak symbol addresses. gcc does
1005                  * not do this, so it's not needed.
1006                  */
1007                 break;
1008            }
1009            if (strcmp(arg, "-Wc,-xcode=", 11) == 0) {
1010                 xlate(ctx->i_ae, arg + 11, xcode_tbl);
1011                 break;
1012            }
1013            if (strcmp(arg, "-Wc,-Qiselect", 13) == 0) {
1014                 /*
1015                  * Prevents insertion of register symbols.
1016                  * gcc doesn't do this, so ignore it.
1017                  */
1018                 break;
1019            }
1020            if (strcmp(arg, "-Wc,-Qassembler-ounrefsym=0") == 0) {
1021                 /*
1022                  * Prevents optimizing away of static variables.
1023                  */
1024                 break;
1025            }

```

```

1023                         * gcc does not do this, so it's not needed.
1024                         */
1025                         break;
1026 }
1027 #if defined(__x86)
1028     if (strcmp(arg, "-Wu,-save_args") == 0) {
1029         newae(ctx->i_ae, "-msave-args");
1030         break;
1031     }
1032 #endif /* __x86 */
1033     error(arg);
1034     break;
1035 case 'X':
1036     if (strcmp(arg, "-Xa") == 0 ||
1037         strcmp(arg, "-Xt") == 0) {
1038         break;
1039     }
1040     if (strcmp(arg, "-Xs") == 0) {
1041         Xsmode(ctx->i_ae);
1042         break;
1043     }
1044     error(arg);
1045     break;
1046 case 'x':
1047     if (arglen == 1)
1048         error(arg);
1049     switch (arg[2]) {
1050     case 'a':
1051         if (strncmp(arg, "-xarch=", 7) == 0) {
1052             mflag |= xlate_xtb(ctx->i_ae, arg + 7);
1053             break;
1054         }
1055         error(arg);
1056         break;
1057     case 'b':
1058         if (strncmp(arg, "-xbuiltin=", 10) == 0) {
1059             if (strcmp(arg + 10, "%all"))
1060                 newae(ctx->i_ae, "-fbuiltin");
1061             break;
1062         }
1063         error(arg);
1064         break;
1065     case 'C':
1066         /* Accept C++ style comments -- ignore */
1067         if (strcmp(arg, "-xCC") == 0)
1068             break;
1069         error(arg);
1070         break;
1071     case 'c':
1072         if (strncmp(arg, "-xc99=%all", 10) == 0) {
1073             newae(ctx->i_ae, "-std=gnu99");
1074             break;
1075         }
1076         if (strncmp(arg, "-xc99=%none", 11) == 0) {
1077             newae(ctx->i_ae, "-std=gnu89");
1078             break;
1079         }
1080         if (strncmp(arg, "-xchip=", 7) == 0) {
1081             xlate(ctx->i_ae, arg + 7, xchip_tbl);
1082             break;
1083         }
1084         if (strncmp(arg, "-xcode=", 7) == 0) {
1085             xlate(ctx->i_ae, arg + 7, xcode_tbl);
1086             break;
1087         }
1088         if (strncmp(arg, "-xcrossfile", 11) == 0)

```

```

1089                         break;
1090                         error(arg);
1091                         break;
1092 }
1093 if (strcmp(arg, "-xdebugformat=", 14) == 0)
1094     break;
1095     error(arg);
1096     break;
1097 case 'F':
1098     /*
1099      * Compile for mapfile reordering, or unused
1100      * section elimination, syntax can be -xF or
1101      * more complex, like -xF=%all -- ignore.
1102     */
1103     if (strncmp(arg, "-xF", 3) == 0)
1104         break;
1105     error(arg);
1106     break;
1107 case 'i':
1108     if (strncmp(arg, "-xinline", 8) == 0)
1109         /*
1110          * No inlining; ignore */
1111         break;
1112     if (strcmp(arg, "-xildon") == 0 ||
1113         strcmp(arg, "-xiloff") == 0)
1114         /*
1115          * No incremental linking; ignore */
1116         break;
1117     error(arg);
1118     break;
1119 #if defined(__x86)
1120
1121
1122
1123
1124
1125
1126
1127
1128 #endif /* __x86 */
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2100
2101
2101
2102
2102
2103
2103
2104
2104
2105
2105
2106
2106
2107
2107
2108
2108
2109
2109
2110
2110
2111
2111
2112
2112
2113
2113
2114
2114
2115
2115
2116
2116
2117
2117
2118
2118
2119
2119
2120
2120
2121
2121
2122
2122
2123
2123
2124
2124
2125
2125
2126
2126
2127
2127
2128
2128
2129
2129
2130
2130
2131
2131
2132
2132
2133
2133
2134
2134
2135
2135
2136
2136
2137
2137
2138
2138
2139
2139
2140
2140
2141
2141
2142
2142
2143
2143
2144
2144
2145
2145
2146
2146
2147
2147
2148
2148
2149
2149
2150
2150
2151
2151
2152
2152
2153
2153
2154
2154
2155
2155
2156
2156
2157
2157
2158
2158
2159
2159
2160
2160
2161
2161
2162
2162
2163
2163
2164
2164
2165
2165
2166
2166
2167
2167
2168
2168
2169
2169
2170
2170
2171
2171
2172
2172
2173
2173
2174
2174
2175
2175
2176
2176
2177
2177
2178
2178
2179
2179
2180
2180
2181
2181
2182
2182
2183
2183
2184
2184
2185
2185
2186
2186
2187
2187
2188
2188
2189
2189
2190
2190
2191
2191
2192
2192
2193
2193
2194
2194
2195
2195
2196
2196
2197
2197
2198
2198
2199
2199
2200
2200
2201
2201
2202
2202
2203
2203
2204
2204
2205
2205
2206
2206
2207
2207
2208
2208
2209
2209
2210
2210
2211
2211
2212
2212
2213
2213
2214
2214
2215
2215
2216
2216
2217
2217
2218
2218
2219
2219
2220
2220
2221
2221
2222
2222
2223
2223
2224
2224
2225
2225
2226
2226
2227
2227
2228
2228
2229
2229
2230
2230
2231
2231
2232
2232
2233
2233
2234
2234
2235
2235
2236
2236
2237
2237
2238
2238
2239
2239
2240
2240
2241
2241
2242
2242
2243
2243
2244
2244
2245
2245
2246
2246
2247
2247
2248
2248
2249
2249
2250
2250
2251
2251
2252
2252
2253
2253
2254
2254
2255
2255
2256
2256
2257
2257
2258
2258
2259
2259
2260
2260
2261
2261
2262
2262
2263
2263
2264
2264
2265
2265
2266
2266
2267
2267
2268
2268
2269
2269
2270
2270
2271
2271
2272
2272
2273
2273
2274
2274
2275
2275
2276
2276
2277
2277
2278
2278
2279
2279
2280
2280
2281
2281
2282
2282
2283
2283
2284
2284
2285
2285
2286
2286
2287
2287
2288
2288
2289
2289
2290
2290
2291
2291
2292
2292
2293
2293
2294
2294
2295
2295
2296
2296
2297
2297
2298
2298
2299
2299
2300
2300
2301
2301
2302
2302
2303
2303
2304
2304
2305
2305
2306
2306
2307
2307
2308
2308
2309
2309
2310
2310
2311
2311
2312
2312
2313
2313
2314
2314
2315
2315
2316
2316
2317
2317
2318
2318
2319
2319
2320
2320
2321
2321
2322
2322
2323
2323
2324
2324
2325
2325
2326
2326
2327
2327
2328
2328
2329
2329
2330
2330
2331
2331
2332
2332
2333
2333
2334
2334
2335
2335
2336
2336
2337
2337
2338
2338
2339
2339
2340
2340
2341
2341
2342
2342
2343
2343
2344
2344
2345
2345
2346
2346
2347
2347
2348
2348
2349
2349
2350
2350
2351
2351
2352
2352
2353
2353
2354
2354
2355
2355
2356
2356
2357
2357
2358
2358
2359
2359
2360
2360
2361
2361
2362
2362
2363
2363
2364
2364
2365
2365
2366
2366
2367
2367
2368
2368
2369
2369
2370
2370
2371
2371
2372
2372
2373
2373
2374
2374
2375
2375
2376
2376
2377
2377
2378
2378
2379
2379
2380
2380
2381
2381
2382
2382
2383
2383
2384
2384
2385
2385
2386
2386
2387
2387
2388
2388
2389
2389
2390
2390
2391
2391
2392
2392
2393
2393
2394
2394
2395
2395
2396
2396
2397
2397
2398
2398
2399
2399
2400
2400
2401
2401
2402
2402
2403
2403
2404
2404
2405
2405
2406
2406
2407
2407
2408
2408
2409
2409
2410
2410
2411
2411
2412
2412
2413
2413
2414
2414
2415
2415
2416
2416
2417
2417
2418
2418
2419
2419
2420
2420
2421
2421
2422
2422
2423
2423
2424
2424
2425
2425
2426
2426
2427
2427
2428
2428
2429
2429
2430
2430
2431
2431
2432
2432
2433
2433
2434
2434
2435
2435
2436
2436
2437
2437
2438
2438
2439
2439
2440
2440
2441
2441
2442
2442
2443
2443
2444
2444
2445
2445
2446
2446
2447
2447
2448
2448
2449
2449
2450
2450
2451
2451
2452
2452
2453
2453
2454
2454
2455
2455
2456
2456
2457
2457
2458
2458
2459
2459
2460
2460
2461
2461
2462
2462
2463
2463
2464
2464
2465
2465
2466
2466
2467
2467
2468
2468
2469
2469
2470
2470
2471
2471
2472
2472
2473
2473
2474
2474
2475
2475
2476
2476
2477
2477
2478
2478
2479
2479
2480
2480
2481
2481
2482
2482
2483
2483
2484
2484
2485
2485
2486
2486
2487
2487
2488
2488
2489
2489
2490
2490
2491
2491
2492
2492
2493
2493
2494
2494
2495
2495
2496
2496
2497
2497
2498
2498
2499
2499
2500
2500
2501
2501
2502
2502
2503
2503
2504
2504
2505
2505
2506
2506
2507
2507
2508
2508
2509
2509
2510
2510
2511
2511
2512
2512
2513
2513
2514
2514
2515
2515
2516
2516
2517
2517
2518
2518
2519
2519
2520
2520
2521
2521
2522
2522
2523
2523
2524
2524
2525
2525
2526
2526
2527
2527
2528
2528
2529
2529
2530
2530
2531
2531
2532
2532
2533
2533
2534
2534
2535
2535
2536
2536
2537
2537
2538
2538
2539
2539
2540
2540
2541
2541
2542
2542
2543
2543
2544
2544
2545
2545
2546
2546
2547
2547
2548
2548
2549
2549
2550
2550
2551
2551
2552
2552
2553
2553
2554
2554
2555
2555
2556
2556
2557
2557
2558
2558
2559
2559
2560
2560
2561
2561
2562
2562
2563
2563
2564
2564
2565
2565
2566
2566
2567
2567
2568
2568
2569
2569
2570
2570
2571
2571
2572
2572
2573
2573
2574
2574
2575
2575
2576
2576
2577
2577
2578
2578
2579
2579
2580
2580
2581
2581
2582
2582
2583
2583
2584
2584
2585
2585
2586
2586
2587
2587
2588
2588
2589
2589
2590
2590
2591
2591
2592
2592
2593
2593
2594
2594
2595
2595
2596
2596
2597
2597
2598
2598
2599
2599
2600
2600
2601
2601
2602
2602
2603
2603
2604
2604
2605
2605
2606
2606
2607
2607
2608
2608
2609
2609
2610
2610
2611
2611
2612
2612
2613

```

```

1155             nomem();
1156             newae(ctx->i_ae, s);
1157             free(s);
1158             break;
1159         }
1160         error(arg);
1161         break;
1162     case 'r':
1163         if (strncmp(arg, "-xregs=", 7) == 0) {
1164             xlate(ctx->i_ae, arg + 7, xregs_tbl);
1165             break;
1166         }
1167         error(arg);
1168         break;
1169     case 's':
1170         if (strcmp(arg, "-xs") == 0 ||
1171             strcmp(arg, "-xspace") == 0 ||
1172             strcmp(arg, "-xstrconst") == 0)
1173             break;
1174         error(arg);
1175         break;
1176     case 't':
1177         if (strncmp(arg, "-xtarget=", 9) == 0) {
1178             xlate(ctx->i_ae, arg + 9, xtargtbl);
1179             break;
1180         }
1181         error(arg);
1182         break;
1183     case 'e':
1184     case 'h':
1185     case 'l':
1186     default:
1187         error(arg);
1188         break;
1189     }
1190     break;
1191 case 'Y':
1192     if (arglen == 1) {
1193         if ((arg = *++ctx->i_oldargv) == NULL ||
1194             *arg == '\0')
1195             error("-Y");
1196         ctx->i_oldargc--;
1197         arglen = strlen(arg + 1);
1198     } else {
1199         arg += 2;
1200     }
1201     /* Just ignore -YS,... for now */
1202     if (strncmp(arg, "S,", 2) == 0)
1203         break;
1204     if (strncmp(arg, "l,", 2) == 0) {
1205         char *s = strdup(arg);
1206         s[0] = '-';
1207         s[1] = 'B';
1208         newae(ctx->i_ae, s);
1209         free(s);
1210         break;
1211     }
1212     if (strncmp(arg, "I,", 2) == 0) {
1213         char *s = strdup(arg);
1214         s[0] = '-';
1215         s[1] = 'I';
1216         newae(ctx->i_ae, "-nostdinc");
1217         newae(ctx->i_ae, s);
1218         free(s);
1219         break;
1220     }

```

```

1221             error(arg);
1222             break;
1223         case 'Q':
1224             /*
1225              * We could map -Qy into -Wl,-Qy etc.
1226              */
1227         default:
1228             error(arg);
1229             break;
1230     }
1231 }
1233 free(nameflag);

1235 /*
1236  * When compiling multiple source files in a single invocation some
1237  * compilers output objects into the current directory with
1238  * predictable and conventional names.
1239  *
1240  * We prevent any attempt to compile multiple files at once so that
1241  * any such objects created by a shadow can't escape into a later
1242  * link-edit.
1243 */
1244 if (c_files > 1 && op != CW_O_PREPROCESS) {
1245 if (c_files > 1 && (ctx->i_flags & CW_F_SHADOW) &&
1246     op != CW_O_PREPROCESS) {
1247     errx(2, "multiple source files are "
1248          "allowed only with -E or -P");
1249 }
1250 /*
1251  * Make sure that we do not have any unintended interactions between
1252  * the xarch options passed in and the version of the Studio compiler
1253  * used.
1254 */
1255 if ((mflag & (SS11|SS12)) == (SS11|SS12)) {
1256     errx(2,
1257          "Conflicting \"-xarch=\" flags (both Studio 11 and 12)\n");
1258 }
1259 switch (mflag) {
1260 case 0:
1261     /* FALLTHROUGH */
1262 case M32:
1263 #if defined(__sparc)
1264     /*
1265      * Only -m32 is defined and so put in the missing xarch
1266      * translation.
1267      */
1268     newae(ctx->i_ae, "-mcpu=v8");
1269     newae(ctx->i_ae, "-mno-v8plus");
1270 #endif
1271     break;
1272 case M64:
1273 #if defined(__sparc)
1274     /*
1275      * Only -m64 is defined and so put in the missing xarch
1276      * translation.
1277      */
1278     newae(ctx->i_ae, "-mcpu=v9");
1279 #endif
1280     break;
1281 case SS12:
1282 #if defined(__sparc)
1283     /*
1284      * no -m32/-m64 flag used - this is an error for sparc builds */
1285     (void) fprintf(stderr, "No -m32/-m64 flag defined\n");
1286 
```

```

1285         exit(2);
1286 #endif
1287         break;
1288     case SS11:
1289         /* FALLTHROUGH */
1290     case (SS11|M32):
1291     case (SS11|M64):
1292         break;
1293     case (SS12|M32):
1294 #if defined(__sparc)
1295         /*
1296          * Need to add in further 32 bit options because with SS12
1297          * the xarch=sparcvis option can be applied to 32 or 64
1298          * bit, and so the translation table (xtbl) cannot handle
1299          * that.
1300         */
1301         newae(ctx->i_ae, "-mv8plus");
1302 #endif
1303         break;
1304     case (SS12|M64):
1305         break;
1306     default:
1307         (void) fprintf(stderr,
1308             "Incompatible -xarch= and/or -m32/-m64 options used.\n");
1309         exit(2);
1310     }
1311
1312     if (ctx->i_flags & CW_F_SHADOW) {
1313         if (op == CW_O_PREPROCESS)
1314             exit(0);
1315         else if (op == CW_O_LINK && c_files == 0)
1316             if ((op == CW_O_LINK || op == CW_O_PREPROCESS) &&
1317                 (ctx->i_flags & CW_F_SHADOW))
1318                 exit(0);
1319     }
1320
1321     if (model != NULL)
1322         newae(ctx->i_ae, model);
1323     if (!nolibc)
1324         newae(ctx->i_ae, "-lc");
1325     if (!seen_o && (ctx->i_flags & CW_F_SHADOW)) {
1326         newae(ctx->i_ae, "-o");
1327         newae(ctx->i_ae, discard_file_name(ctx, NULL));
1328     }
1329
1330     unchanged_portion_omitted
1331
1332 static void
1333 do_cc(cw_ictx_t *ctx)
1334 {
1335     int in_output = 0, seen_o = 0, c_files = 0;
1336     int in_output = 0, seen_o = 0;
1337     cw_op_t op = CW_O_LINK;
1338     char *nameflag;
1339
1340     if (ctx->i_flags & CW_F_PROG) {
1341         newae(ctx->i_ae, "-V");
1342         return;
1343     }
1344
1345     if (asprintf(&nameflag, "-%s=", ctx->i_compiler->c_name) == -1)
1346         nomem();
1347
1348     while (--ctx->i_oldargc > 0) {

```

```

1372         char *arg = *++ctx->i_oldargv;
1373         size_t arglen = strlen(arg);
1374 #endif /* ! codereview */
1375
1376         if (strncmp(arg, "-CC=", 5) == 0) {
1377             newae(ctx->i_ae, strchr(arg, '=')+1);
1378             continue;
1379         }
1380
1381         if (*arg != '-') {
1382             if (!in_output && arglen > 2 &&
1383                 arg[arglen-2] == ',' &&
1384                 (arg[arglen-1] == 's' || arg[arglen-1] == 'S' ||
1385                  arg[arglen-1] == 'c' || arg[arglen-1] == 'I' ||
1386                  c_files++));
1387
1388 #endif /* ! codereview */
1389         if (in_output == 0 || !(ctx->i_flags & CW_F_SHADOW)) {
1390             newae(ctx->i_ae, arg);
1391         } else {
1392             in_output = 0;
1393             newae(ctx->i_ae, discard_file_name(ctx, arg));
1394             newae(ctx->i_ae, ctx->i_discard);
1395         }
1396         continue;
1397     switch (*arg + 1) {
1398     case '_':
1399         if ((strncmp(arg, nameflag, strlen(nameflag)) == 0) ||
1400             (strncmp(arg, "-cc=", 5) == 0) ||
1401             (strncmp(arg, "-sun=", 6) == 0)) {
1402             newae(ctx->i_ae, strchr(arg, '=')+1);
1403         }
1404         break;
1405
1406     case 'V':
1407         ctx->i_flags &= ~CW_F_ECHO;
1408         newae(ctx->i_ae, arg);
1409         break;
1410
1411     case 'o':
1412         seen_o = 1;
1413         if (strlen(arg) == 2) {
1414             in_output = 1;
1415             newae(ctx->i_ae, arg);
1416         } else if (ctx->i_flags & CW_F_SHADOW) {
1417             newae(ctx->i_ae, "-o");
1418             newae(ctx->i_ae, discard_file_name(ctx, arg));
1419             newae(ctx->i_ae, ctx->i_discard);
1420         } else {
1421             newae(ctx->i_ae, arg);
1422         }
1423         break;
1424
1425     case 'c':
1426     case 'S':
1427         if (strlen(arg) == 2)
1428             op = CW_O_COMPILE;
1429         newae(ctx->i_ae, arg);
1430         break;
1431
1432     case 'E':
1433     case 'P':
1434         if (strlen(arg) == 2)
1435             op = CW_O_PREPROCESS;
1436         /*FALLTHROUGH*/
1437     default:
1438         newae(ctx->i_ae, arg);
1439     }

```

```

1436     }
1438     free(nameflag);
1439
1440     /* See the comment on this same code in do_gcc() */
1441     if (c_files > 1 && op != CW_O_PREPROCESS) {
1442         errx(2, "multiple source files are "
1443             "allowed only with -E or -P");
1444     }
1445
1446     if (ctx->i_flags & CW_F_SHADOW) {
1447         if (op == CW_O_PREPROCESS)
1118         if ((op == CW_O_LINK || op == CW_O_PREPROCESS) &&
1119             (ctx->i_flags & CW_F_SHADOW))
1448             exit(0);
1449         else if (op == CW_O_LINK && c_files == 0)
1450             exit(0);
1451     }
1452 #endif /* ! codereview */
1453
1454     if (!seen_o && (ctx->i_flags & CW_F_SHADOW)) {
1455         newae(ctx->i_ae, "-o");
1456         newae(ctx->i_ae, discard_file_name(ctx, NULL));
1121         newae(ctx->i_ae, ctx->i_discard);
1457     }
1458 }
_____unchanged_portion_omitted_
1538 static int
1539 reap(cw_ictx_t *ctx)
1540 {
1541     int status, ret = 0;
1542     char buf[1024];
1543     struct stat s;
1544
1545     /*
1546      * Only wait for one specific child.
1547      */
1548     if (ctx->i_pid <= 0)
1549         return (-1);
1550
1551     do {
1552         if (waitpid(ctx->i_pid, &status, 0) < 0) {
1553             warn("cannot reap child");
1554             return (-1);
1555         }
1556         if (status != 0) {
1557             if (WIFSIGNALED(status)) {
1558                 ret = -WTERMSIG(status);
1559                 break;
1560             } else if (WIFEXITED(status)) {
1561                 ret = WEXITSTATUS(status);
1562                 break;
1563             }
1564         }
1565     } while (!WIFEXITED(status) && !WIFSIGNALED(status));
1232
1566     (void) unlink(ctx->i_discard);
1567
1568     if (stat(ctx->i_stderr, &s) < 0) {
1569         warn("stat failed on child cleanup");
1570         return (-1);
1571     }
1572     if (s.st_size != 0) {
1573         FILE *f;

```

```

1574         if ((f = fopen(ctx->i_stderr, "r")) != NULL) {
1575             while (fgets(buf, sizeof (buf), f))
1576                 (void) fprintf(stderr, "%s", buf);
1577             (void) fflush(stderr);
1578             (void) fclose(f);
1579         }
1580     }
1581     (void) unlink(ctx->i_stderr);
1582     free(ctx->i_stderr);
1583
1584     /*
1585      * cc returns an error code when given -V; we want that to succeed.
1586      */
1587     if (ctx->i_flags & CW_F_PROG)
1588         return (0);
1589
1590     return (ret);
1591 }
1592
1593 static int
1594 exec_ctx(cw_ictx_t *ctx, int block)
1595 {
1596     if ((ctx->i_stderr = tempnam(ctx->i_tmpdir, "cw")) == NULL) {
1263         char *file;
1264
1265         /*
1266          * To avoid offending cc's sensibilities, the name of its output
1267          * file must end in '.o'.
1268          */
1269         if ((file = tempnam(NULL, ".cw")) == NULL) {
1270             nomem();
1271             return (-1);
1272         }
1273         (void) strlcpy(ctx->i_discard, file, MAXPATHLEN);
1274         (void) strlcat(ctx->i_discard, ".o", MAXPATHLEN);
1275         free(file);
1276
1277         if ((ctx->i_stderr = tempnam(NULL, ".cw")) == NULL) {
1278             nomem();
1279             return (-1);
1280         }
1281
1282         if ((ctx->i_pid = fork()) == 0) {
1283             int fd;
1284
1285             (void) fclose(stderr);
1286             if ((fd = open(ctx->i_stderr, O_WRONLY | O_CREAT | O_EXCL,
1287                           0666)) < 0) {
1288                 err(1, "open failed for standard error");
1289             }
1290             if (dup2(fd, 2) < 0) {
1291                 err(1, "dup2 failed for standard error");
1292             }
1293             if (fd != 2)
1294                 (void) close(fd);
1295             if (freopen("/dev/fd/2", "w", stderr) == NULL) {
1296                 err(1, "freopen failed for /dev/fd/2");
1297             }
1298
1299             prepctx(ctx);
1300             exit(invoker(ctx));
1301         }
1302
1303         if (ctx->i_pid < 0) {
1304             err(1, "fork failed");
1305         }
1306
1307         if (ctx->i_stderr == NULL) {
1308             err(1, "exec failed");
1309         }
1310
1311         if (ctx->i_stderr != NULL) {
1312             (void) fclose(ctx->i_stderr);
1313             free(ctx->i_stderr);
1314         }
1315
1316         if (ctx->i_stderr == NULL) {
1317             err(1, "exec failed");
1318         }
1319
1320         if (ctx->i_stderr != NULL) {
1321             (void) fclose(ctx->i_stderr);
1322             free(ctx->i_stderr);
1323         }
1324
1325         if (ctx->i_stderr == NULL) {
1326             err(1, "exec failed");
1327         }
1328
1329         if (ctx->i_stderr != NULL) {
1330             (void) fclose(ctx->i_stderr);
1331             free(ctx->i_stderr);
1332         }
1333
1334         if (ctx->i_stderr == NULL) {
1335             err(1, "exec failed");
1336         }
1337
1338         if (ctx->i_stderr != NULL) {
1339             (void) fclose(ctx->i_stderr);
1340             free(ctx->i_stderr);
1341         }
1342
1343         if (ctx->i_stderr == NULL) {
1344             err(1, "exec failed");
1345         }
1346
1347         if (ctx->i_stderr != NULL) {
1348             (void) fclose(ctx->i_stderr);
1349             free(ctx->i_stderr);
1350         }
1351
1352         if (ctx->i_stderr == NULL) {
1353             err(1, "exec failed");
1354         }
1355
1356         if (ctx->i_stderr != NULL) {
1357             (void) fclose(ctx->i_stderr);
1358             free(ctx->i_stderr);
1359         }
1360
1361         if (ctx->i_stderr == NULL) {
1362             err(1, "exec failed");
1363         }
1364
1365         if (ctx->i_stderr != NULL) {
1366             (void) fclose(ctx->i_stderr);
1367             free(ctx->i_stderr);
1368         }
1369
1370         if (ctx->i_stderr == NULL) {
1371             err(1, "exec failed");
1372         }
1373
1374         if (ctx->i_stderr != NULL) {
1375             (void) fclose(ctx->i_stderr);
1376             free(ctx->i_stderr);
1377         }
1378
1379         if (ctx->i_stderr == NULL) {
1380             err(1, "exec failed");
1381         }
1382
1383         if (ctx->i_stderr != NULL) {
1384             (void) fclose(ctx->i_stderr);
1385             free(ctx->i_stderr);
1386         }
1387
1388         if (ctx->i_stderr == NULL) {
1389             err(1, "exec failed");
1390         }
1391
1392         if (ctx->i_stderr != NULL) {
1393             (void) fclose(ctx->i_stderr);
1394             free(ctx->i_stderr);
1395         }
1396
1397         if (ctx->i_stderr == NULL) {
1398             err(1, "exec failed");
1399         }
1400
1401         if (ctx->i_stderr != NULL) {
1402             (void) fclose(ctx->i_stderr);
1403             free(ctx->i_stderr);
1404         }
1405
1406         if (ctx->i_stderr == NULL) {
1407             err(1, "exec failed");
1408         }
1409
1410         if (ctx->i_stderr != NULL) {
1411             (void) fclose(ctx->i_stderr);
1412             free(ctx->i_stderr);
1413         }
1414
1415         if (ctx->i_stderr == NULL) {
1416             err(1, "exec failed");
1417         }
1418
1419         if (ctx->i_stderr != NULL) {
1420             (void) fclose(ctx->i_stderr);
1421             free(ctx->i_stderr);
1422         }
1423
1424         if (ctx->i_stderr == NULL) {
1425             err(1, "exec failed");
1426         }
1427
1428         if (ctx->i_stderr != NULL) {
1429             (void) fclose(ctx->i_stderr);
1430             free(ctx->i_stderr);
1431         }
1432
1433         if (ctx->i_stderr == NULL) {
1434             err(1, "exec failed");
1435         }
1436
1437         if (ctx->i_stderr != NULL) {
1438             (void) fclose(ctx->i_stderr);
1439             free(ctx->i_stderr);
1440         }
1441
1442         if (ctx->i_stderr == NULL) {
1443             err(1, "exec failed");
1444         }
1445
1446         if (ctx->i_stderr != NULL) {
1447             (void) fclose(ctx->i_stderr);
1448             free(ctx->i_stderr);
1449         }
1450
1451         if (ctx->i_stderr == NULL) {
1452             err(1, "exec failed");
1453         }
1454
1455         if (ctx->i_stderr != NULL) {
1456             (void) fclose(ctx->i_stderr);
1457             free(ctx->i_stderr);
1458         }
1459
1460         if (ctx->i_stderr == NULL) {
1461             err(1, "exec failed");
1462         }
1463
1464         if (ctx->i_stderr != NULL) {
1465             (void) fclose(ctx->i_stderr);
1466             free(ctx->i_stderr);
1467         }
1468
1469         if (ctx->i_stderr == NULL) {
1470             err(1, "exec failed");
1471         }
1472
1473         if (ctx->i_stderr != NULL) {
1474             (void) fclose(ctx->i_stderr);
1475             free(ctx->i_stderr);
1476         }
1477
1478         if (ctx->i_stderr == NULL) {
1479             err(1, "exec failed");
1480         }
1481
1482         if (ctx->i_stderr != NULL) {
1483             (void) fclose(ctx->i_stderr);
1484             free(ctx->i_stderr);
1485         }
1486
1487         if (ctx->i_stderr == NULL) {
1488             err(1, "exec failed");
1489         }
1490
1491         if (ctx->i_stderr != NULL) {
1492             (void) fclose(ctx->i_stderr);
1493             free(ctx->i_stderr);
1494         }
1495
1496         if (ctx->i_stderr == NULL) {
1497             err(1, "exec failed");
1498         }
1499
1500         if (ctx->i_stderr != NULL) {
1501             (void) fclose(ctx->i_stderr);
1502             free(ctx->i_stderr);
1503         }
1504
1505         if (ctx->i_stderr == NULL) {
1506             err(1, "exec failed");
1507         }
1508
1509         if (ctx->i_stderr != NULL) {
1510             (void) fclose(ctx->i_stderr);
1511             free(ctx->i_stderr);
1512         }
1513
1514         if (ctx->i_stderr == NULL) {
1515             err(1, "exec failed");
1516         }
1517
1518         if (ctx->i_stderr != NULL) {
1519             (void) fclose(ctx->i_stderr);
1520             free(ctx->i_stderr);
1521         }
1522
1523         if (ctx->i_stderr == NULL) {
1524             err(1, "exec failed");
1525         }
1526
1527         if (ctx->i_stderr != NULL) {
1528             (void) fclose(ctx->i_stderr);
1529             free(ctx->i_stderr);
1530         }
1531
1532         if (ctx->i_stderr == NULL) {
1533             err(1, "exec failed");
1534         }
1535
1536         if (ctx->i_stderr != NULL) {
1537             (void) fclose(ctx->i_stderr);
1538             free(ctx->i_stderr);
1539         }
1540
1541         if (ctx->i_stderr == NULL) {
1542             err(1, "exec failed");
1543         }
1544
1545         if (ctx->i_stderr != NULL) {
1546             (void) fclose(ctx->i_stderr);
1547             free(ctx->i_stderr);
1548         }
1549
1550         if (ctx->i_stderr == NULL) {
1551             err(1, "exec failed");
1552         }
1553
1554         if (ctx->i_stderr != NULL) {
1555             (void) fclose(ctx->i_stderr);
1556             free(ctx->i_stderr);
1557         }
1558
1559         if (ctx->i_stderr == NULL) {
1560             err(1, "exec failed");
1561         }
1562
1563         if (ctx->i_stderr != NULL) {
1564             (void) fclose(ctx->i_stderr);
1565             free(ctx->i_stderr);
1566         }
1567
1568         if (ctx->i_stderr == NULL) {
1569             err(1, "exec failed");
1570         }
1571
1572         if (ctx->i_stderr != NULL) {
1573             (void) fclose(ctx->i_stderr);
1574             free(ctx->i_stderr);
1575         }
1576
1577         if (ctx->i_stderr == NULL) {
1578             err(1, "exec failed");
1579         }
1580
1581         if (ctx->i_stderr != NULL) {
1582             (void) fclose(ctx->i_stderr);
1583             free(ctx->i_stderr);
1584         }
1585
1586         if (ctx->i_stderr == NULL) {
1587             err(1, "exec failed");
1588         }
1589
1590         if (ctx->i_stderr != NULL) {
1591             (void) fclose(ctx->i_stderr);
1592             free(ctx->i_stderr);
1593         }
1594
1595         if (ctx->i_stderr == NULL) {
1596             err(1, "exec failed");
1597         }
1598
1599         if (ctx->i_stderr != NULL) {
1600             (void) fclose(ctx->i_stderr);
1601             free(ctx->i_stderr);
1602         }
1603
1604         if (ctx->i_stderr == NULL) {
1605             err(1, "exec failed");
1606         }
1607
1608         if (ctx->i_stderr != NULL) {
1609             (void) fclose(ctx->i_stderr);
1610             free(ctx->i_stderr);
1611         }
1612
1613         if (ctx->i_stderr == NULL) {
1614             err(1, "exec failed");
1615         }
1616
1617         if (ctx->i_stderr != NULL) {
1618             (void) fclose(ctx->i_stderr);
1619             free(ctx->i_stderr);
1620         }
1621
1622         if (ctx->i_stderr == NULL) {
1623             err(1, "exec failed");
1624         }
1625
1626         if (ctx->i_stderr != NULL) {
1627             (void) fclose(ctx->i_stderr);
1628             free(ctx->i_stderr);
1629         }
1630
1631         if (ctx->i_stderr == NULL) {
1632             err(1, "exec failed");
1633         }
1634
1635         if (ctx->i_stderr != NULL) {
1636             (void) fclose(ctx->i_stderr);
1637             free(ctx->i_stderr);
1638         }
1639
1640         if (ctx->i_stderr == NULL) {
1641             err(1, "exec failed");
1642         }
1643
1644         if (ctx->i_stderr != NULL) {
1645             (void) fclose(ctx->i_stderr);
1646             free(ctx->i_stderr);
1647         }
1648
1649         if (ctx->i_stderr == NULL) {
1650             err(1, "exec failed");
1651         }
1652
1653         if (ctx->i_stderr != NULL) {
1654             (void) fclose(ctx->i_stderr);
1655             free(ctx->i_stderr);
1656         }
1657
1658         if (ctx->i_stderr == NULL) {
1659             err(1, "exec failed");
1660         }
1661
1662         if (ctx->i_stderr != NULL) {
1663             (void) fclose(ctx->i_stderr);
1664             free(ctx->i_stderr);
1665         }
1666
1667         if (ctx->i_stderr == NULL) {
1668             err(1, "exec failed");
1669         }
1670
1671         if (ctx->i_stderr != NULL) {
1672             (void) fclose(ctx->i_stderr);
1673             free(ctx->i_stderr);
1674         }
1675
1676         if (ctx->i_stderr == NULL) {
1677             err(1, "exec failed");
1678         }
1679
1680         if (ctx->i_stderr != NULL) {
1681             (void) fclose(ctx->i_stderr);
1682             free(ctx->i_stderr);
1683         }
1684
1685         if (ctx->i_stderr == NULL) {
1686             err(1, "exec failed");
1687         }
1688
1689         if (ctx->i_stderr != NULL) {
1690             (void) fclose(ctx->i_stderr);
1691             free(ctx->i_stderr);
1692         }
1693
1694         if (ctx->i_stderr == NULL) {
1695             err(1, "exec failed");
1696         }
1697
1698         if (ctx->i_stderr != NULL) {
1699             (void) fclose(ctx->i_stderr);
1700             free(ctx->i_stderr);
1701         }
1702
1703         if (ctx->i_stderr == NULL) {
1704             err(1, "exec failed");
1705         }
1706
1707         if (ctx->i_stderr != NULL) {
1708             (void) fclose(ctx->i_stderr);
1709             free(ctx->i_stderr);
1710         }
1711
1712         if (ctx->i_stderr == NULL) {
1713             err(1, "exec failed");
1714         }
1715
1716         if (ctx->i_stderr != NULL) {
1717             (void) fclose(ctx->i_stderr);
1718             free(ctx->i_stderr);
1719         }
1720
1721         if (ctx->i_stderr == NULL) {
1722             err(1, "exec failed");
1723         }
1724
1725         if (ctx->i_stderr != NULL) {
1726             (void) fclose(ctx->i_stderr);
1727             free(ctx->i_stderr);
1728         }
1729
1730         if (ctx->i_stderr == NULL) {
1731             err(1, "exec failed");
1732         }
1733
1734         if (ctx->i_stderr != NULL) {
1735             (void) fclose(ctx->i_stderr);
1736             free(ctx->i_stderr);
1737         }
1738
1739         if (ctx->i_stderr == NULL) {
1740             err(1, "exec failed");
1741         }
1742
1743         if (ctx->i_stderr != NULL) {
1744             (void) fclose(ctx->i_stderr);
1745             free(ctx->i_stderr);
1746         }
1747
1748         if (ctx->i_stderr == NULL) {
1749             err(1, "exec failed");
1750         }
1751
1752         if (ctx->i_stderr != NULL) {
1753             (void) fclose(ctx->i_stderr);
1754             free(ctx->i_stderr);
1755         }
1756
1757         if (ctx->i_stderr == NULL) {
1758             err(1, "exec failed");
1759         }
1760
1761         if (ctx->i_stderr != NULL) {
1762             (void) fclose(ctx->i_stderr);
1763             free(ctx->i_stderr);
1764         }
1765
1766         if (ctx->i_stderr == NULL) {
1767             err(1, "exec failed");
1768         }
1769
1770         if (ctx->i_stderr != NULL) {
1771             (void) fclose(ctx->i_stderr);
1772             free(ctx->i_stderr);
1773         }
1774
1775         if (ctx->i_stderr == NULL) {
1776             err(1, "exec failed");
1777         }
1778
1779         if (ctx->i_stderr != NULL) {
1780             (void) fclose(ctx->i_stderr);
1781             free(ctx->i_stderr);
1782         }
1783
1784         if (ctx->i_stderr == NULL) {
1785             err(1, "exec failed");
1786         }
1787
1788         if (ctx->i_stderr != NULL) {
1789             (void) fclose(ctx->i_stderr);
1790             free(ctx->i_stderr);
1791         }
1792
1793         if (ctx->i_stderr == NULL) {
1794             err(1, "exec failed");
1795         }
1796
1797         if (ctx->i_stderr != NULL) {
1798             (void) fclose(ctx->i_stderr);
1799             free(ctx->i_stderr);
1800         }
1801
1802         if (ctx->i_stderr == NULL) {
1803             err(1, "exec failed");
1804         }
1805
1806         if (ctx->i_stderr != NULL) {
1807             (void) fclose(ctx->i_stderr);
1808             free(ctx->i_stderr);
1809         }
1810
1811         if (ctx->i_stderr == NULL) {
1812             err(1, "exec failed");
1813         }
1814
1815         if (ctx->i_stderr != NULL) {
1816             (void) fclose(ctx->i_stderr);
1817             free(ctx->i_stderr);
1818         }
1819
1820         if (ctx->i_stderr == NULL) {
1821             err(1, "exec failed");
1822         }
1823
1824         if (ctx->i_stderr != NULL) {
1825             (void) fclose(ctx->i_stderr);
1826             free(ctx->i_stderr);
1827         }
1828
1829         if (ctx->i_stderr == NULL) {
1830             err(1, "exec failed");
1831         }
1832
1833         if (ctx->i_stderr != NULL) {
1834             (void) fclose(ctx->i_stderr);
1835             free(ctx->i_stderr);
1836         }
1837
1838         if (ctx->i_stderr == NULL) {
1839             err(1, "exec failed");
1840         }
1841
1842         if (ctx->i_stderr != NULL) {
1843             (void) fclose(ctx->i_stderr);
1844             free(ctx->i_stderr);
1845         }
1846
1847         if (ctx->i_stderr == NULL) {
1848             err(1, "exec failed");
1849         }
1850
1851         if (ctx->i_stderr != NULL) {
1852             (void) fclose(ctx->i_stderr);
1853             free(ctx->i_stderr);
1854         }
1855
1856         if (ctx->i_stderr == NULL) {
1857             err(1, "exec failed");
1858         }
1859
1860         if (ctx->i_stderr != NULL) {
1861             (void) fclose(ctx->i_stderr);
1862             free(ctx->i_stderr);
1863         }
1864
1865         if (ctx->i_stderr == NULL) {
1866             err(1, "exec failed");
1867         }
1868
1869         if (ctx->i_stderr != NULL) {
1870             (void) fclose(ctx->i_stderr);
1871             free(ctx->i_stderr);
1872         }
1873
1874         if (ctx->i_stderr == NULL) {
1875             err(1, "exec failed");
1876         }
1877
1878         if (ctx->i_stderr != NULL) {
1879             (void) fclose(ctx->i_stderr);
1880             free(ctx->i_stderr);
1881         }
1882
1883         if (ctx->i_stderr == NULL) {
1884             err(1, "exec failed");
1885         }
1886
1887         if (ctx->i_stderr != NULL) {
1888             (void) fclose(ctx->i_stderr);
1889             free(ctx->i_stderr);
1890         }
1891
1892         if (ctx->i_stderr == NULL) {
1893             err(1, "exec failed");
1894         }
1895
1896         if (ctx->i_stderr != NULL) {
1897             (void) fclose(ctx->i_stderr);
1898             free(ctx->i_stderr);
1899         }
1900
1901         if (ctx->i_stderr == NULL) {
1902             err(1, "exec failed");
1903         }
1904
1905         if (ctx->i_stderr != NULL) {
1906             (void) fclose(ctx->i_stderr);
1907             free(ctx->i_stderr);
1908         }
1909
1910         if (ctx->i_stderr == NULL) {
1911             err(1, "exec failed");
1912         }
1913
1914         if (ctx->i_stderr != NULL) {
1915             (void) fclose(ctx->i_stderr);
1916             free(ctx->i_stderr);
1917         }
1918
1919         if (ctx->i_stderr == NULL) {
1920             err(1, "exec failed");
1921         }
1922
1923         if (ctx->i_stderr != NULL) {
1924             (void) fclose(ctx->i_stderr);
1925             free(ctx->i_stderr);
1926         }
1927
1928         if (ctx->i_stderr == NULL) {
1929             err(1, "exec failed");
1930         }
1931
1932         if (ctx->i_stderr != NULL) {
1933             (void) fclose(ctx->i_stderr);
1934             free(ctx->i_stderr);
1935         }
1936
1937         if (ctx->i_stderr == NULL) {
1938             err(1, "exec failed");
1939         }
1940
1941         if (ctx->i_stderr != NULL) {
1942             (void) fclose(ctx->i_stderr);
1943             free(ctx->i_stderr);
1944         }
1945
1946         if (ctx->i_stderr == NULL) {
1947             err(1, "exec failed");
1948         }
1949
1950         if (ctx->i_stderr != NULL) {
1951             (void) fclose(ctx->i_stderr);
1952             free(ctx->i_stderr);
1953         }
1954
1955         if (ctx->i_stderr == NULL) {
1956             err(1, "exec failed");
1957         }
1958
1959         if (ctx->i_stderr != NULL) {
1960             (void) fclose(ctx->i_stderr);
1961             free(ctx->i_stderr);
1962         }
1963
1964         if (ctx->i_stderr == NULL) {
1965             err(1, "exec failed");
1966         }
1967
1968         if (ctx->i_stderr != NULL) {
1969             (void) fclose(ctx->i_stderr);
1970             free(ctx->i_stderr);
1971         }
1972
1973         if (ctx->i_stderr == NULL) {
1974             err(1, "exec failed");
1975         }
1976
1977         if (ctx->i_stderr != NULL) {
1978             (void) fclose(ctx->i_stderr);
1979             free(ctx->i_stderr);
1980         }
1981
1982         if (ctx->i_stderr == NULL) {
1983             err(1, "exec failed");
1984         }
1985
1986         if (ctx->i_stderr != NULL) {
1987             (void) fclose(ctx->i_stderr);
1988             free(ctx->i_stderr);
1989         }
1990
1991         if (ctx->i_stderr == NULL) {
1992             err(1, "exec failed");
1993         }
1994
1995         if (ctx->i_stderr != NULL) {
1996             (void) fclose(ctx->i_stderr);
1997             free(ctx->i_stderr);
1998         }
1999
2000         if (ctx->i_stderr == NULL) {
2001             err(1, "exec failed");
2002         }
2003
2004         if (ctx->i_stderr != NULL) {
2005             (void) fclose(ctx->i_stderr);
2006             free(ctx->i_stderr);
2007         }
2008
2009         if (ctx->i_stderr == NULL) {
2010             err(1, "exec failed");
2011         }
2012
2013         if (ctx->i_stderr != NULL) {
2014             (void) fclose(ctx->i_stderr);
2015             free(ctx->i_stderr);
2016         }
2017
2018         if (ctx->i_stderr == NULL) {
2019             err(1, "exec failed");
2020         }
2021
2022         if (ctx->i_stderr != NULL) {
2023             (void) fclose(ctx->i_stderr);
2024             free(ctx->i_stderr);
2025         }
2026
2027         if (ctx->i_stderr == NULL) {
2028             err(1, "exec failed");
2029         }
2030
2031         if (ctx->i_stderr != NULL) {
2032             (void) fclose(ctx->i_stderr);
2033             free(ctx->i_stderr);
2034         }
2035
2036         if (ctx->i_stderr == NULL) {
2037             err(1, "exec failed");
2038         }
2039
2040         if (ctx->i_stderr != NULL) {
2041             (void) fclose(ctx->i_stderr);
2042             free(ctx->i_stderr);
2043         }
2044
2045         if (ctx->i_stderr == NULL) {
2046             err(1, "exec failed");
2047         }
2048
2049         if (ctx->i_stderr != NULL) {
2050             (void) fclose(ctx->i_stderr);
2051             free(ctx->i_stderr);
2052         }
2053
2054         if (ctx->i_stderr == NULL) {
2055             err(1, "exec failed");
2056         }
2057
2058         if (ctx->i_stderr != NULL) {
2059             (void) fclose(ctx->i_stderr);
2060             free(ctx->i_stderr);
2061         }
2062
2063         if (ctx->i_stderr == NULL) {
2064             err(1, "exec failed");
2065         }
2066
2067         if (ctx->i_stderr != NULL) {
2068             (void) fclose(ctx->i_stderr);
2069             free(ctx->i_stderr);
2070         }
2071
2072         if (ctx->i_stderr == NULL) {
2073             err(1, "exec failed");
2074         }
2075
2076         if (ctx->i_stderr != NULL) {
2077             (void) fclose(ctx->i_stderr);
2078             free(ctx->i_stderr);
2079         }
2080
2081         if (ctx->i_stderr == NULL) {
2082             err(1, "exec failed");
2083         }
2084
2085         if (ctx->i_stderr != NULL) {
2086             (void) fclose(ctx->i_stderr);
2087             free(ctx->i_stderr);
2088         }
2089
2090         if (ctx->i_stderr == NULL) {
2091             err(1, "exec failed");
2092         }
2093
2094         if (ctx->i_stderr != NULL) {
2095             (void) fclose(ctx->i_stderr);
2096             free(ctx->i_stderr);
2097         }
2098
2099         if (ctx->i_stderr == NULL) {
2100             err(1, "exec failed");
2101         }
2102
2103         if (ctx->i_stderr != NULL) {
2104             (void) fclose(ctx->i_stderr);
2105             free(ctx->i_stderr);
2106         }
2107
2108         if (ctx->i_stderr == NULL) {
2109             err(1, "exec failed");
2110         }
2111
2112         if (ctx->i_stderr != NULL) {
2113             (void) fclose(ctx->i_stderr);
2114             free(ctx->i_stderr);
2115         }
2116
2117         if (ctx->i_stderr == NULL) {
2118             err(1, "exec failed");
2119         }
2120
2121         if (ctx->i_stderr != NULL) {
2122             (void) fclose(ctx->i_stderr);
2123             free(ctx->i_stderr);
2124         }
2125
2126         if (ctx->i_stderr == NULL) {
2127             err(1, "exec failed");
2128         }
2129
2130         if (ctx->i_stderr != NULL) {
2131             (void) fclose(ctx->i_stderr);
2132             free(ctx->i_stderr);
2133         }
2134
2135         if (ctx->i_stderr == NULL) {
2136             err(1, "exec failed");
2137         }
2138
2139         if (ctx->i_stderr != NULL) {
2140             (void) fclose(ctx->i_stderr);
2141             free(ctx->i_stderr);
2142         }
2143
2144         if (ctx->i_stderr == NULL) {
2145             err(1, "exec failed");
2146         }
2147
2148         if (ctx->i_stderr != NULL) {
2149             (void) fclose(ctx->i_stderr);
2150             free(ctx->i_stderr);
2151         }
2152
2153         if (ctx->i_stderr == NULL) {
2154             err(1, "exec failed");
2155         }
2156
2157         if (ctx->i_stderr != NULL) {
2158             (void) fclose(ctx->i_stderr);
2159             free(ctx->i_stderr);
2160         }
2161
2162         if (ctx->i_stderr == NULL) {
2163             err(1, "exec failed");
2164         }
2165
2166         if (ctx->i_stderr != NULL) {
2167             (void) fclose(ctx->i_stderr);
2168             free(ctx->i_stderr);
2169         }
2170
2171         if (ctx->i_stderr == NULL) {
2172             err(1, "exec failed");
2173         }
2174
2175         if (ctx->i_stderr != NULL) {
2176             (void) fclose(ctx->i_stderr);
2177             free(ctx->i_stderr);
2178         }
2179
2180         if (ctx->i_stderr == NULL) {
2181             err(1, "exec failed");
2182         }
2183
2184         if (ctx->i_stderr != NULL) {
2185             (void) fclose(ctx->i_stderr);
2186             free(ctx->i_stderr);
2187         }
2188
2189         if (ctx->i_stderr == NULL) {
2190             err(1, "exec failed");
2191         }
2192
2193         if (ctx->i_stderr != NULL) {
2194             (void) fclose(ctx->i_stderr);
2195             free(ctx->i_stderr);
2196         }
2197
2198         if (ctx->i_stderr == NULL) {
2199             err(1, "exec failed");
2200         }
2201
2202         if (ctx->i_stderr != NULL) {
2203             (void) fclose(ctx->i_stderr);
2204             free(ctx->i_stderr);
2205         }
2206
2207         if (ctx->i_stderr == NULL) {
2208             err(1, "exec failed");
2209         }
2210
2211         if (ctx->i_stderr != NULL) {
2212             (void) fclose(ctx->i_stderr);
2213             free(ctx->i_stderr);
2214         }
2215
2216         if (ctx->i_stderr == NULL) {
2217             err(1, "exec failed");
2218         }
2219
2220         if (ctx->i_stderr != NULL) {
2221             (void) fclose(ctx->i_stderr);
2222             free(ctx->i_stderr);
2223         }
2224
2225         if (ctx->i_stderr == NULL) {
2226             err(1, "exec failed");
222
```

```

1626     if (block)
1627         return (reap(ctx));
1629
1630 }
unchanged_portion_omitted_
1667 static void
1668 cleanup(cw_ictx_t *ctx)
1669 {
1670     DIR *dirp;
1671     struct dirent *dp;
1672     char buf[MAXPATHLEN];
1673
1674     if ((dirp = opendir(ctx->i_tmpdir)) == NULL) {
1675         if (errno != ENOENT) {
1676             err(1, "couldn't open temp directory: %s",
1677                 ctx->i_tmpdir);
1678         } else {
1679             return;
1680         }
1681     }
1682
1683     errno = 0;
1684     while ((dp = readdir(dirp)) != NULL) {
1685         (void) snprintf(buf, MAXPATHLEN, "%s/%s", ctx->i_tmpdir,
1686                         dp->d_name);
1687
1688         if (strncmp(dp->d_name, ".", strlen(dp->d_name)) == 0 ||
1689             strncmp(dp->d_name, "..", strlen(dp->d_name)) == 0)
1690             continue;
1691
1692         if (unlink(buf) == -1)
1693             err(1, "failed to unlink temp file: %s", dp->d_name);
1694         errno = 0;
1695     }
1696
1697     if (errno != 0) {
1698         err(1, "failed to read temporary directory: %s",
1699             ctx->i_tmpdir);
1700     }
1701
1702     (void) closedir(dirp);
1703     if (rmdir(ctx->i_tmpdir) != 0) {
1704         err(1, "failed to unlink temporary directory: %s",
1705             ctx->i_tmpdir);
1706     }
1707 }

1709 #endif /* ! codereview */
1710 int
1711 main(int argc, char **argv)
1712 {
1713     int ch;
1714     cw_compiler_t primary = { NULL, NULL, 0 };
1715     cw_compiler_t shadows[10];
1716     int nshadows = 0;
1717     int ret = 0;
1718     boolean_t do_serial = B_FALSE;
1719     boolean_t do_exec = B_FALSE;
1720     boolean_t vflg = B_FALSE;
1721     boolean_t Cflg = B_FALSE;
1722     boolean_t cflg = B_FALSE;
1723     boolean_t nflg = B_FALSE;
1724     char *tmpdir;

```

```

1725 #endif /* ! codereview */
1726
1727     cw_ictx_t *main_ctx;
1728
1729     static struct option longopts[] = {
1730         { "compiler", no_argument, NULL, 'c' },
1731         { "noecho", no_argument, NULL, 'n' },
1732         { "primary", required_argument, NULL, 'p' },
1733         { "shadow", required_argument, NULL, 's' },
1734         { "versions", no_argument, NULL, 'v' },
1735         { NULL, 0, NULL, 0 },
1736     };
1737
1738
1739     if ((main_ctx = newictx()) == NULL)
1740         nomem();
1741
1742     while ((ch = getopt_long(argc, argv, "C", longopts, NULL)) != -1) {
1743         switch (ch) {
1744         case 'c':
1745             cflg = B_TRUE;
1746             break;
1747         case 'C':
1748             Cfllg = B_TRUE;
1749             break;
1750         case 'n':
1751             nflg = B_TRUE;
1752             break;
1753         case 'p':
1754             if (primary.c_path != NULL) {
1755                 warnx("Only one primary compiler may "
1756                     "be specified");
1757                 usage();
1758             }
1759             parse_compiler(optarg, &primary);
1760             break;
1761         case 's':
1762             if (nshadows >= 10)
1763                 errx(1, "May only use 10 shadows at "
1764                     "the moment");
1765             parse_compiler(optarg, &shadows[nshadows]);
1766             nshadows++;
1767             break;
1768         case 'v':
1769             vflg = B_TRUE;
1770             break;
1771         default:
1772             (void) fprintf(stderr, "Did you forget '--?\n");
1773             usage();
1774         }
1775     }
1776
1777     if (primary.c_path == NULL) {
1778         warnx("A primary compiler must be specified");
1779         usage();
1780     }
1781
1782     do_serial = (getenv("CW_SHADOW_SERIAL") == NULL) ? B_FALSE : B_TRUE;
1783     do_exec = (getenv("CW_NO_EXEC") == NULL) ? B_TRUE : B_FALSE;
1784
1785     /* Leave room for argv[0] */
1786     argc -= (optind - 1);
1787     argv += (optind - 1);
1788
1789     main_ctx->i_oldargc = argc;

```

```

1791     main_ctx->i_oldargv = argv;
1792     main_ctx->i_flags = CW_F_XLATE;
1793     if (nflg == 0)
1794         main_ctx->i_flags |= CW_F_ECHO;
1795     if (do_exec)
1796         main_ctx->i_flags |= CW_F_EXEC;
1797     if (Cflg)
1798         main_ctx->i_flags |= CW_F_CXX;
1799     main_ctx->i_compiler = &primary;

1801     if (cflg) {
1802         (void) fputs(primary.c_path, stdout);
1803     }

1805     if (vflg) {
1806         (void) printf("cw version %s\n", CW_VERSION);
1807         (void) fflush(stdout);
1808         main_ctx->i_flags &= ~CW_F_ECHO;
1809         main_ctx->i_flags |= CW_F_PROG | CW_F_EXEC;
1810         do_serial = 1;
1811     }

1813     tmpdir = getenv("TMPDIR");
1814     if (tmpdir == NULL)
1815         tmpdir = "/tmp";
1816
1817     if (asprintf(&main_ctx->i_tmpdir, "%s/cw.XXXXXX", tmpdir) == -1)
1818         nomem();

1820     if ((main_ctx->i_tmpdir = mkdtemp(main_ctx->i_tmpdir)) == NULL)
1821         errx(1, "failed to create temporary directory");

1823 #endif /* ! codereview */
1824     ret |= exec_ctx(main_ctx, do_serial);

1826     for (int i = 0; i < nshadows; i++) {
1827         int r;
1828         cw_ictx_t *shadow_ctx;
1829
1830         if ((shadow_ctx = newictx()) == NULL)
1831             nomem();
1832
1833         (void) memcpy(shadow_ctx, main_ctx, sizeof(cw_ictx_t));
1834         memcpy(shadow_ctx, main_ctx, sizeof(cw_ictx_t));
1835
1836         shadow_ctx->i_flags |= CW_F_SHADOW;
1837         shadow_ctx->i_compiler = &shadows[i];
1838
1839         r = exec_ctx(shadow_ctx, do_serial);
1840         if (r == 0) {
1841             shadow_ctx->i_next = main_ctx->i_next;
1842             main_ctx->i_next = shadow_ctx;
1843         }
1844         ret |= r;
1845     }

1846     if (!do_serial) {
1847         cw_ictx_t *next = main_ctx;
1848         while (next != NULL) {
1849             cw_ictx_t *toreap = next;
1850             next = next->i_next;
1851             ret |= reap(toreap);
1852         }
1853     }

1855     cleanup(main_ctx);

```

```

1856 #endif /* ! codereview */
1857         return (ret);
1858 }

```