```
**********
   37366 Sun Jan 25 13:29:10 2015
new/usr/src/Makefile.master
we should be using the arm cross-linker
**********
   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 #

  22 #
  23 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright (c) 2012 by Delphix. All rights reserved.
  25 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  26 #

  28 #
  29 # Makefile.master, global definitions for system source
  30 #
  31 ROOT=           /proto

  33 #
  34 # Adjunct root, containing an additional proto area to be used for headers
  35 # and libraries.
  36 #
  37 ADJUNCT_PROTO=

  39 #
  40 # Adjunct for building things that run on the build machine.
  41 #
  42 NATIVE_ADJUNCT= /usr

  44 #
  45 # RELEASE_BUILD should be cleared for final release builds.
  46 # NOT_RELEASE_BUILD is exactly what the name implies.
  47 #
  48 # __GNUC toggles the building of ON components using gcc and related tools.
  49 # Normally set to '#', set it to '' to do gcc build.
  50 #
  51 # The declaration POUND_SIGN is always '#'. This is needed to get around the
  52 # make feature that '#' is always a comment delimiter, even when escaped or
  53 # quoted. We use this macro expansion method to get POUND_SIGN rather than
  54 # always breaking out a shell because the general case can cause a noticable
  55 # slowdown in build times when so many Makefiles include Makefile.master.
  56 #
  57 # While the majority of users are expected to override the setting below
  58 # with an env file (via nightly or bldenv), if you aren't building that way
  59 # (ie, you're using "ws" or some other bootstrapping method) then you need
  60 # this definition in order to avoid the subshell invocation mentioned above.
  61 #
```



```
  63 PRE_POUND=                              pre\#
  64 POUND_SIGN=                             $(PRE_POUND:pre\%=%)

  66 NOT_RELEASE_BUILD=
  67 RELEASE_BUILD=                          $(POUND_SIGN)
  68 $(RELEASE_BUILD)NOT_RELEASE_BUILD=      $(POUND_SIGN)
  69 PATCH_BUILD=                            $(POUND_SIGN)

  71 # SPARC_BLD is '#' for an Intel build.
  72 # INTEL_BLD is '#' for a Sparc build.
  73 SPARC_BLD_2=    $(MACH:arm=$(POUND_SIGN))
  74 SPARC_BLD_1=    $(MACH:i386=$(POUND_SIGN))
  75 SPARC_BLD=      $(SPARC_BLD_1:sparc=)
  76 INTEL_BLD_2=    $(MACH:arm=$(POUND_SIGN))
  77 INTEL_BLD_1=    $(MACH:sparc=$(POUND_SIGN))
  78 INTEL_BLD=      $(INTEL_BLD_1:i386=)
  79 ARM_BLD_2=      $(MACH:i386=$(POUND_SIGN))
  80 ARM_BLD_1=      $(MACH:sparc=$(POUND_SIGN))
  81 ARM_BLD=        $(ARM_BLD_1:arm=)

  83 # CROSS_BLD is "#" if NATIVE_MACH and MACH are the same, and the empty string
  84 # otherwise.  NATIVE_BLD is the opposite
  85 CROSS_BLD_1=    $(NATIVE_MACH:$(MACH)=$(POUND_SIGN))
  86 CROSS_BLD=      $(CROSS_BLD_1:$(NATIVE_MACH)=)
  87 NATIVE_BLD=
  88 $(CROSS_BLD)NATIVE_BLD= $(POUND_SIGN)

  90 # The variables below control the compilers used during the build.
  91 # There are a number of permutations.
  92 #
  93 # __GNUC and __SUNC control (and indicate) the primary compiler.  Whichever
  94 # one is not POUND_SIGN is the primary, with the other as the shadow.  They
  95 # may also be used to control entirely compiler-specific Makefile assignments.
  96 # __GNUC and GCC are the default.
  97 #
  98 # __GNUC64 indicates that the 64bit build should use the GNU C compiler.
  99 # There is no Sun C analogue.
 100 #
 101 # The following version-specific options are operative regardless of which
 102 # compiler is primary, and control the versions of the given compilers to be
 103 # used.  They also allow compiler-version specific Makefile fragments.
 104 #

 106 __SUNC=                 $(POUND_SIGN)
 107 $(__SUNC)__GNUC=        $(POUND_SIGN)
 108 __GNUC64=               $(__GNUC)

 110 # CLOSED is the root of the tree that contains source which isn't released
 111 # as open source
 112 CLOSED=         $(SRC)/../closed

 114 # BUILD_TOOLS is the root of all tools including compilers.
 115 # ONBLD_TOOLS is the root of all the tools that are part of SUNWonbld.

 117 BUILD_TOOLS=            /ws/onnv-tools
 118 ONBLD_TOOLS=            $(BUILD_TOOLS)/onbld

 120 JAVA_ROOT=      /usr/java

 122 SFW_ROOT=       /usr/sfw
 123 SFWINCDIR=      $(SFW_ROOT)/include
 124 SFWLIBDIR=      $(SFW_ROOT)/lib
 125 SFWLIBDIR64=    $(SFW_ROOT)/lib/$(MACH64)

 127 i386_GCC_ROOT=  /opt/gcc/4.4.4
```

```
128 sparc_GCC_ROOT= /opt/gcc/4.4.4
129 arm_GCC_ROOT=   /opt/gcc/4.4.4

131 NATIVE_GCC_ROOT=        $($(NATIVE_MACH)_GCC_ROOT)

133 GCC_ROOT=               $($(MACH)_GCC_ROOT)
134 GCCLIBDIR=              $(GCC_ROOT)/lib
135 GCCLIBDIR64=            $(GCC_ROOT)/lib/$(MACH64)

137 DOCBOOK_XSL_ROOT=       /usr/share/sgml/docbook/xsl-stylesheets

139 RPCGEN=          /usr/bin/rpcgen
140 STABS=           $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/stabs
141 ELFEXTRACT=      $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/elfextract
142 MBH_PATCH=       $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/mbh_patch
143 ECHO=            echo
144 INS=             install
145 TRUE=            true
146 SYMLINK=         /usr/bin/ln -s
147 LN=              /usr/bin/ln
148 CHMOD=           /usr/bin/chmod
149 MV=              /usr/bin/mv -f
150 RM=              /usr/bin/rm -f
151 CUT=             /usr/bin/cut
152 NM=              /usr/ccs/bin/nm
153 DIFF=            /usr/bin/diff
154 GREP=            /usr/bin/grep
155 EGREP=           /usr/bin/egrep
156 ELFWRAP=         /usr/bin/elfwrap
157 KSH93=           /usr/bin/ksh93
158 SED=             /usr/bin/sed
159 NAWK=            /usr/bin/nawk
160 CP=              /usr/bin/cp -f
161 MCS=             /usr/ccs/bin/mcs
162 CAT=             /usr/bin/cat
163 ELFDUMP=         /usr/ccs/bin/elfdump
164 M4=              /usr/ccs/bin/m4
165 STRIP=           /usr/ccs/bin/strip
166 LEX=             /usr/ccs/bin/lex
167 FLEX=            $(SFW_ROOT)/bin/flex
168 YACC=            /usr/ccs/bin/yacc
169 CPP=             /usr/lib/cpp
170 JAVAC=           $(JAVA_ROOT)/bin/javac
171 JAVAH=           $(JAVA_ROOT)/bin/javah
172 JAVADOC=         $(JAVA_ROOT)/bin/javadoc
173 RMIC=            $(JAVA_ROOT)/bin/rmic
174 JAR=             $(JAVA_ROOT)/bin/jar
175 CTFSTABS=        $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfstabs
176 CTFSTRIP=        $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfstrip
177 NDRGEN=          $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ndrgen
178 GENOFFSETS=      $(ONBLD_TOOLS)/bin/genoffsets
179 CTFCVTPTBL=      $(ONBLD_TOOLS)/bin/ctfcvtptbl
180 CTFFINDMOD=      $(ONBLD_TOOLS)/bin/ctffindmod
181 XREF=            $(ONBLD_TOOLS)/bin/xref
182 FIND=            /usr/bin/find
183 PERL=            /usr/bin/perl
184 PERL_VERSION=    5.10.0
185 PERL_PKGVERS=    -510
186 PYTHON_26=       /usr/bin/python2.6
187 PYTHON=          $(PYTHON_26)
188 SORT=            /usr/bin/sort
189 TOUCH=           /usr/bin/touch
190 WC=              /usr/bin/wc
191 XARGS=           /usr/bin/xargs
192 ELFEDIT=         /usr/bin/elfedit
193 ELFSIGN=         $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/elfsign
```



```
194 DTRACE=          /usr/sbin/dtrace -xnolibs
195 UNIQ=            /usr/bin/uniq
196 TAR=             /usr/bin/tar
197 ASTBINDIR=       /usr/ast/bin
198 MSGCC=           $(ASTBINDIR)/msgcc

200 FILEMODE=        644
201 DIRMODE=         755

203 #
204 # The version of the patch makeup table optimized for build-time use.  Used
205 # during patch builds only.
206 $(PATCH_BUILD)PMTMO_FILE=$(SRC)/patch_makeup_table.mo

208 # Declare that nothing should be built in parallel.
209 # Individual Makefiles can use the .PARALLEL target to declare otherwise.
210 .NO_PARALLEL:

212 # For stylistic checks
213 #
214 # Note that the X and C checks are not used at this time and may need
215 # modification when they are actually used.
216 #
217 CSTYLE=          $(ONBLD_TOOLS)/bin/cstyle
218 CSTYLE_TAIL=
219 HDRCHK=          $(ONBLD_TOOLS)/bin/hdrchk
220 HDRCHK_TAIL=
221 JSTYLE=          $(ONBLD_TOOLS)/bin/jstyle

223 DOT_H_CHECK=     \
224         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL); \
225         $(HDRCHK) $< $(HDRCHK_TAIL)

227 DOT_X_CHECK=     \
228         @$(ECHO) "checking $<"; $(RPCGEN) -C -h $< | $(CSTYLE) $(CSTYLE_TAIL); \
229         $(RPCGEN) -C -h $< | $(HDRCHK) $< $(HDRCHK_TAIL)

231 DOT_C_CHECK=     \
232         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL)

234 MANIFEST_CHECK= \
235         @$(ECHO) "checking $<"; \
236         SVCCFG_DTD=$(SRC)/cmd/svc/dtd/service_bundle.dtd.1 \
237         SVCCFG_REPOSITORY=$(SRC)/cmd/svc/seed/global.db \
238         SVCCFG_CONFIGD_PATH=$(SRC)/cmd/svc/configd/svc.configd-native \
239         $(SRC)/cmd/svc/svccfg/svccfg-native validate $<

241 INS.file=        $(RM) $@; $(INS) -s -m $(FILEMODE) -f $(@D) $<
242 INS.dir=         $(INS) -s -d -m $(DIRMODE) $@
243 # installs and renames at once
244 #
245 INS.rename=      $(INS.file); $(MV) $(@D)/$(<F) $@

247 # install a link
248 INSLINKTARGET=  $<
249 INS.link=        $(RM) $@; $(LN) $(INSLINKTARGET) $@
250 INS.symlink=     $(RM) $@; $(SYMLINK) $(INSLINKTARGET) $@

252 #
253 # Python bakes the mtime of the .py file into the compiled .pyc and
254 # rebuilds if the baked-in mtime != the mtime of the source file
255 # (rather than only if it's less than), thus when installing python
256 # files we must make certain to not adjust the mtime of the source
257 # (.py) file.
258 #
259 INS.pyfile=      $(INS.file); $(TOUCH) -r $< $@
```

```
261 # MACH must be set in the shell environment to describe the target machine.
262 # If $MACH does not match uname -p on the build host, NATIVE_MACH must be set
263 # too.  More specific architecture variables should be set in lower makefiles.
264 #
265 # MACH64 is derived from MACH, and BUILD64 is set to '#' for
266 # architectures on which we do not build 64-bit versions.
267 # (There are no such architectures at the moment.)
268 #
269 # Set BUILD64=# in the environment to disable 64-bit amd64
270 # builds on i386 machines.

272 MACH64_1=       $(MACH:sparc=sparcv9)
273 MACH64=         $(MACH64_1:i386=amd64)

275 MACH32_1=       $(MACH:sparc=sparcv7)
276 MACH32=         $(MACH32_1:i386=i86)

278 sparc_BUILD64=
279 i386_BUILD64=
280 arm_BUILD64=    $(POUND_SIGN)
281 BUILD64=        $($(MACH)_BUILD64)

283 #
284 # C compiler mode. Future compilers may change the default on us,
285 # so force extended ANSI mode globally. Lower level makefiles can
286 # override this by setting CCMODE.
287 #
288 CCMODE=                 -Xa
289 CCMODE64=               -Xa

291 #
292 # C compiler verbose mode. This is so we can enable it globally,
293 # but turn it off in the lower level makefiles of things we cannot
294 # (or aren't going to) fix.
295 #
296 CCVERBOSE=              -v

298 # set this to the secret flag "-Wc,-Qiselect-v9abiwarn=1" to get warnings
299 # from the compiler about places the -xarch=v9 may differ from -xarch=v9c.
300 V9ABIWARN=

302 # set this to the secret flag "-Wc,-Qiselect-regsym=0" to disable register
303 # symbols (used to detect conflicts between objects that use global registers)
304 # we disable this now for safety, and because genunix doesn't link with
305 # this feature (the v9 default) enabled.
306 #
307 # REGSYM is separate since the C++ driver syntax is different.
308 CCREGSYM=               -Wc,-Qiselect-regsym=0
309 CCCREGSYM=              -Qoption cg -Qiselect-regsym=0

311 # Prevent the removal of static symbols by the SPARC code generator (cg).
312 # The x86 code generator (ube) does not remove such symbols and as such
313 # using this workaround is not applicable for x86.
314 #
315 CCSTATICSYM=            -Wc,-Qassembler-ounrefsym=0
316 #
317 # generate 32-bit addresses in the v9 kernel. Saves memory.
318 CCABS32=                -Wc,-xcode=abs32
319 #
320 # generate v9 code which tolerates callers using the v7 ABI, for the sake of
321 # system calls.
322 CC32BITCALLERS=         -_gcc=-massume-32bit-callers

324 # GCC, especially, is increasingly beginning to auto-inline functions and
325 # sadly does so separately not under the general -fno-inline-functions
```



```
326 # Additionally, we wish to prevent optimisations which cause GCC to clone
327 # functions -- in particular, these may cause unhelpful symbols to be
328 # emitted instead of function names
329 CCNOAUTOINLINE= -_gcc=-fno-inline-small-functions \
330         -_gcc=-fno-inline-functions-called-once \
331         -_gcc=-fno-ipa-cp

333 # One optimization the compiler might perform is to turn this:
334 #       #pragma weak foo
335 #       extern int foo;
336 #       if (&foo)
337 #               foo = 5;
338 # into
339 #       foo = 5;
340 # Since we do some of this (foo might be referenced in common kernel code
341 # but provided only for some cpu modules or platforms), we disable this
342 # optimization.
343 #
344 sparc_CCUNBOUND = -Wd,-xsafe=unboundsym
345 i386_CCUNBOUND  =
346 CCUNBOUND       = $($(MACH)_CCUNBOUND)

348 #
349 # compiler '-xarch' flag. This is here to centralize it and make it
350 # overridable for testing.
351 sparc_XARCH=    -m32
352 sparcv9_XARCH=  -m64
353 i386_XARCH=
354 amd64_XARCH=    -m64 -Ui386 -U__i386
355 arm_XARCH=

357 # assembler '-xarch' flag.  Different from compiler '-xarch' flag.
358 sparc_AS_XARCH=         -xarch=v8plus
359 sparcv9_AS_XARCH=       -xarch=v9
360 i386_AS_XARCH=
361 amd64_AS_XARCH=         -xarch=amd64 -P -Ui386 -U__i386
362 arm_AS_XARCH=

364 #
365 # These flags define what we need to be 'standalone' i.e. -not- part
366 # of the rather more cosy userland environment.  This basically means
367 # the kernel.
368 #
369 # XX64  future versions of gcc will make -mcmodel=kernel imply -mno-red-zone
370 #
371 sparc_STAND_FLAGS=      -_gcc=-ffreestanding
372 sparcv9_STAND_FLAGS=    -_gcc=-ffreestanding
373 # Disabling MMX also disables 3DNow, disabling SSE also disables all later
374 # additions to SSE (SSE2, AVX ,etc.)
375 NO_SIMD=                -_gcc=-mno-mmx -_gcc=-mno-sse
376 i386_STAND_FLAGS=       -_gcc=-ffreestanding $(NO_SIMD)
377 amd64_STAND_FLAGS=      -xmodel=kernel $(NO_SIMD)
378 arm_STAND_FLAGS=        -_gcc=-ffreestanding

380 SAVEARGS=               -Wu,-save_args
381 amd64_STAND_FLAGS       += $(SAVEARGS)

383 STAND_FLAGS_32 = $($(MACH)_STAND_FLAGS)
384 STAND_FLAGS_64 = $($(MACH64)_STAND_FLAGS)

386 #
387 # disable the incremental linker
388 ILDOFF=                 -xildoff
389 #
390 XDEPEND=                -xdepend
391 XFFLAG=                 -xF=%all
```

```
392 XESS=                   -xs
393 XSTRCONST=              -xstrconst

395 #
396 # turn warnings into errors (C)
397 CERRWARN = -errtags=yes -errwarn=%all
398 CERRWARN += -erroff=E_EMPTY_TRANSLATION_UNIT
399 CERRWARN += -erroff=E_STATEMENT_NOT_REACHED

401 CERRWARN += -_gcc=-Wno-missing-braces
402 CERRWARN += -_gcc=-Wno-sign-compare
403 CERRWARN += -_gcc=-Wno-unknown-pragmas
404 CERRWARN += -_gcc=-Wno-unused-parameter
405 CERRWARN += -_gcc=-Wno-missing-field-initializers

407 # Unfortunately, this option can misfire very easily and unfixably.
408 CERRWARN +=     -_gcc=-Wno-array-bounds

410 # DEBUG v. -nd make for frequent unused variables, empty conditions, etc. in
411 # -nd builds
412 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused
413 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-empty-body

415 #
416 # turn warnings into errors (C++)
417 CCERRWARN=              -xwe

419 # C99 mode
420 C99_ENABLE=     -xc99=%all
421 C99_DISABLE=    -xc99=%none
422 C99MODE=        $(C99_DISABLE)
423 C99LMODE=       $(C99MODE:-xc99%=-Xc99%)

425 # In most places, assignments to these macros should be appended with +=
426 # (CPPFLAGS.master allows values to be prepended to CPPFLAGS).
427 sparc_CFLAGS=   $(sparc_XARCH) $(CCSTATICSYM)
428 sparcv9_CFLAGS= $(sparcv9_XARCH) -dalign $(CCVERBOSE) $(V9ABIWARN) $(CCREGSYM) \
429                 $(CCSTATICSYM)
430 i386_CFLAGS=    $(i386_XARCH)
431 amd64_CFLAGS=   $(amd64_XARCH)
432 arm_CFLAGS=     $(arm_XARCH)

434 sparc_ASFLAGS=  $(sparc_AS_XARCH)
435 sparcv9_ASFLAGS=$(sparcv9_AS_XARCH)
436 i386_ASFLAGS=   $(i386_AS_XARCH)
437 amd64_ASFLAGS=  $(amd64_AS_XARCH)
438 arm_ASFLAGS=    $(arm_AS_XARCH)

440 #
441 sparc_COPTFLAG=         -xO3
442 sparcv9_COPTFLAG=       -xO3
443 i386_COPTFLAG=          -O
444 amd64_COPTFLAG=         -xO3
445 arm_COPTFLAG=           -xO3

447 COPTFLAG= $($(MACH)_COPTFLAG)
448 COPTFLAG64= $($(MACH64)_COPTFLAG)

450 # When -g is used, the compiler globalizes static objects
451 # (gives them a unique prefix). Disable that.
452 CNOGLOBAL= -W0,-noglobal

454 # Direct the Sun Studio compiler to use a static globalization prefix based on t
455 # name of the module rather than something unique. Otherwise, objects
456 # will not build deterministically, as subsequent compilations of identical
457 # source will yeild objects that always look different.
```



```
458 #
459 # In the same spirit, this will also remove the date from the N_OPT stab.
460 CGLOBALSTATIC= -W0,-xglobalstatic

462 # Sometimes we want all symbols and types in debugging information even
463 # if they aren't used.
464 CALLSYMS=       -W0,-xdbggen=no%usedonly

466 #
467 # Default debug format for Sun Studio 11 is dwarf, so force it to
468 # generate stabs.
469 #
470 DEBUGFORMAT=    -xdebugformat=stabs

472 #
473 # Flags used to build in debug mode for ctf generation.  Bugs in the Devpro
474 # compilers currently prevent us from building with cc-emitted DWARF.
475 #
476 CTF_FLAGS_sparc = -g -Wc,-Qiselect-T1 $(C99MODE) $(CNOGLOBAL) $(CDWARFSTR)
477 CTF_FLAGS_i386  = -g $(C99MODE) $(CNOGLOBAL) $(CDWARFSTR)
478 CTF_FLAGS_arm   = -g $(C99MODE) $(CNOGLOBAL) $(CDWARFSTR)

480 CTF_FLAGS_sparcv9       = $(CTF_FLAGS_sparc)
481 CTF_FLAGS_amd64         = $(CTF_FLAGS_i386)

483 # Sun Studio produces broken userland code when saving arguments.
484 $(__GNUC)CTF_FLAGS_amd64 += $(SAVEARGS)

486 CTF_FLAGS_32    = $(CTF_FLAGS_$(MACH)) $(DEBUGFORMAT)
487 CTF_FLAGS_64    = $(CTF_FLAGS_$(MACH64)) $(DEBUGFORMAT)
488 CTF_FLAGS       = $(CTF_FLAGS_32)

490 #
491 # Flags used with genoffsets
492 #
493 GOFLAGS = -_noecho \
494         $(CALLSYMS) \
495         $(CDWARFSTR)

497 OFFSETS_CREATE = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
498         $(CC) $(GOFLAGS) $(CFLAGS) $(CPPFLAGS)

500 OFFSETS_CREATE64 = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
501         $(CC) $(GOFLAGS) $(CFLAGS64) $(CPPFLAGS)

503 #
504 # tradeoff time for space (smaller is better)
505 #
506 sparc_SPACEFLAG         = -xspace -W0,-Lt
507 sparcv9_SPACEFLAG       = -xspace -W0,-Lt
508 i386_SPACEFLAG          = -xspace
509 amd64_SPACEFLAG         =
510 arm_SPACFLAGE           =       #XXXARM: Really?

512 SPACEFLAG               = $($(MACH)_SPACEFLAG)
513 SPACEFLAG64             = $($(MACH64)_SPACEFLAG)

515 #
516 # The Sun Studio 11 compiler has changed the behaviour of integer
517 # wrap arounds and so a flag is needed to use the legacy behaviour
518 # (without this flag panics/hangs could be exposed within the source).
519 #
520 sparc_IROPTFLAG         = -W2,-xwrap_int
521 sparcv9_IROPTFLAG       = -W2,-xwrap_int
522 i386_IROPTFLAG          =
523 amd64_IROPTFLAG         =
```

```
524 arm_IROPTFLAG           =

526 IROPTFLAG               = $($(MACH)_IROPTFLAG)
527 IROPTFLAG64             = $($(MACH64)_IROPTFLAG)

529 sparc_XREGSFLAG         = -xregs=no%appl
530 sparcv9_XREGSFLAG       = -xregs=no%appl
531 i386_XREGSFLAG          =
532 amd64_XREGSFLAG         =
533 arm_XREGSFLAG           =

535 XREGSFLAG               = $($(MACH)_XREGSFLAG)
536 XREGSFLAG64             = $($(MACH64)_XREGSFLAG)

538 # dmake SOURCEDEBUG=yes ... enables source-level debugging information, and
539 # avoids stripping it.
540 SOURCEDEBUG     = $(POUND_SIGN)
541 SRCDBGBLD       = $(SOURCEDEBUG:yes=)

543 #
544 # These variables are intended ONLY for use by developers to safely pass extra
545 # flags to the compilers without unintentionally overriding Makefile-set
546 # flags.  They should NEVER be set to any value in a Makefile.
547 #
548 # They come last in the associated FLAGS variable such that they can
549 # explicitly override things if necessary, there are gaps in this, but it's
550 # the best we can manage.
551 #
552 CUSERFLAGS              =
553 CUSERFLAGS64            = $(CUSERFLAGS)
554 CCUSERFLAGS             =
555 CCUSERFLAGS64           = $(CCUSERFLAGS)

557 CSOURCEDEBUGFLAGS       =
558 CCSOURCEDEBUGFLAGS      =
559 $(SRCDBGBLD)CSOURCEDEBUGFLAGS   = -g -xs
560 $(SRCDBGBLD)CCSOURCEDEBUGFLAGS  = -g -xs

562 CFLAGS=         $(COPTFLAG) $($(MACH)_CFLAGS) $(SPACEFLAG) $(CCMODE) \
563                 $(ILDOFF) $(CERRWARN) $(C99MODE) $(CCUNBOUND) $(IROPTFLAG) \
564                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
565                 $(CUSERFLAGS)
566 CFLAGS64=       $(COPTFLAG64) $($(MACH64)_CFLAGS) $(SPACEFLAG64) $(CCMODE64) \
567                 $(ILDOFF) $(CERRWARN) $(C99MODE) $(CCUNBOUND) $(IROPTFLAG64) \
568                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CSOURCEDEBUGFLAGS) \
569                 $(CUSERFLAGS64)
570 #
571 # Flags that are used to build parts of the code that are subsequently
572 # run on the build machine (also known as the NATIVE_BUILD).
573 #
574 NATIVE_CFLAGS=  $(COPTFLAG) $($(NATIVE_MACH)_CFLAGS) $(CCMODE) \
575                 $(ILDOFF) $(CERRWARN) $(C99MODE) $($(NATIVE_MACH)_CCUNBOUND) \
576                 $(IROPTFLAG) $(CGLOBALSTATIC) $(CCNOAUTOINLINE) \
577                 $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)

579 DTEXTDOM=-DTEXT_DOMAIN=\"$(TEXT_DOMAIN)\"       # For messaging.
580 DTS_ERRNO=-D_TS_ERRNO
581 CPPFLAGS.master=$(DTEXTDOM) $(DTS_ERRNO) \
582         $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4) \
583         $(ADJUNCT_PROTO:%=-I%/usr/include)
584 CPPFLAGS.native=$(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) \
585                 $(ENVCPPFLAGS4) -I$(NATIVE_ADJUNCT)/include
586 CPPFLAGS=       $(CPPFLAGS.master)
587 AS_CPPFLAGS=    $(CPPFLAGS.master)
588 JAVAFLAGS=      -deprecation
```



```
590 #
591 # For source message catalogue
592 #
593 .SUFFIXES: $(SUFFIXES) .i .po
594 MSGROOT= $(ROOT)/catalog
595 MSGDOMAIN= $(MSGROOT)/$(TEXT_DOMAIN)
596 MSGDOMAINPOFILE = $(MSGDOMAIN)/$(POFILE)
597 DCMSGDOMAIN= $(MSGROOT)/LC_TIME/$(TEXT_DOMAIN)
598 DCMSGDOMAINPOFILE = $(DCMSGDOMAIN)/$(DCFILE:.dc=.po)

600 CLOBBERFILES += $(POFILE) $(POFILES)
601 COMPILE.cpp= $(CC) -E -C $(CFLAGS) $(CPPFLAGS)
602 XGETTEXT= /usr/bin/xgettext
603 XGETFLAGS= -c TRANSLATION_NOTE
604 GNUXGETTEXT= /usr/gnu/bin/xgettext
605 GNUXGETFLAGS= --add-comments=TRANSLATION_NOTE --keyword=_ \
606         --strict --no-location --omit-header
607 BUILD.po= $(XGETTEXT) $(XGETFLAGS) -d $(<F) $<.i ;\
608         $(RM)   $@ ;\
609         $(SED) "/^domain/d" < $(<F).po > $@ ;\
610         $(RM) $(<F).po $<.i

612 #
613 # This is overwritten by local Makefile when PROG is a list.
614 #
615 POFILE= $(PROG).po

617 sparc_CCFLAGS=          -cg92 -compat=4 \
618                         -Qoption ccfe -messages=no%anachronism \
619                         $(CCERRWARN)
620 sparcv9_CCFLAGS=        $(sparcv9_XARCH) -dalign -compat=5 \
621                         -Qoption ccfe -messages=no%anachronism \
622                         -Qoption ccfe -features=no%conststrings \
623                         $(CCCREGSYM) \
624                         $(CCERRWARN)
625 i386_CCFLAGS=           -compat=4 \
626                         -Qoption ccfe -messages=no%anachronism \
627                         -Qoption ccfe -features=no%conststrings \
628                         $(CCERRWARN)
629 amd64_CCFLAGS=          $(amd64_XARCH) -compat=5 \
630                         -Qoption ccfe -messages=no%anachronism \
631                         -Qoption ccfe -features=no%conststrings \
632                         $(CCERRWARN)
633 arm_CCFLAGS=            $(CCERRWARN)

635 sparc_CCOPTFLAG=        -O
636 sparcv9_CCOPTFLAG=      -O
637 i386_CCOPTFLAG=         -O
638 amd64_CCOPTFLAG=        -O
639 arm_CCOPTFLAG=          -O

641 CCOPTFLAG=      $($(MACH)_CCOPTFLAG)
642 CCOPTFLAG64=    $($(MACH64)_CCOPTFLAG)
643 CCFLAGS=        $(CCOPTFLAG) $($(MACH)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
644                 $(CCUSERFLAGS)
645 CCFLAGS64=      $(CCOPTFLAG64) $($(MACH64)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
646                 $(CCUSERFLAGS64)

648 #
649 #
650 #
651 ELFWRAP_FLAGS   =
652 ELFWRAP_FLAGS64 =       -64

654 #
655 # Various mapfiles that are used throughout the build, and delivered to
```

```
656 # /usr/lib/ld.
657 #
658 MAPFILE.NED_i386 =      $(SRC)/common/mapfiles/common/map.noexdata
659 MAPFILE.NED_sparc =
660 MAPFILE.NED =           $(MAPFILE.NED_$(MACH))
661 MAPFILE.PGA =           $(SRC)/common/mapfiles/common/map.pagealign
662 MAPFILE.NES =           $(SRC)/common/mapfiles/common/map.noexstk
663 MAPFILE.FLT =           $(SRC)/common/mapfiles/common/map.filter
664 MAPFILE.LEX =           $(SRC)/common/mapfiles/common/map.lex.yy

666 #
667 # Generated mapfiles that are compiler specific, and used throughout the
668 # build.  These mapfiles are not delivered in /usr/lib/ld.
669 #
670 MAPFILE.NGB_sparc=      $(SRC)/common/mapfiles/gen/sparc_cc_map.noexeglobs
671 $(__GNUC64)MAPFILE.NGB_sparc= \
672                         $(SRC)/common/mapfiles/gen/sparc_gcc_map.noexeglobs
673 MAPFILE.NGB_sparcv9=    $(SRC)/common/mapfiles/gen/sparcv9_cc_map.noexeglobs
674 $(__GNUC64)MAPFILE.NGB_sparcv9= \
675                         $(SRC)/common/mapfiles/gen/sparcv9_gcc_map.noexeglobs
676 MAPFILE.NGB_i386=       $(SRC)/common/mapfiles/gen/i386_cc_map.noexeglobs
677 $(__GNUC64)MAPFILE.NGB_i386= \
678                         $(SRC)/common/mapfiles/gen/i386_gcc_map.noexeglobs
679 MAPFILE.NGB_amd64=      $(SRC)/common/mapfiles/gen/amd64_cc_map.noexeglobs
680 $(__GNUC64)MAPFILE.NGB_amd64= \
681                         $(SRC)/common/mapfiles/gen/amd64_gcc_map.noexeglobs
682 MAPFILE.NGB_arm= \
683                         $(SRC)/common/mapfiles/gen/arm_gcc_map.noexeglobs
684 MAPFILE.NGB =           $(MAPFILE.NGB_$(MACH))

686 #
687 # A generic interface mapfile name, used by various dynamic objects to define
688 # the interfaces and interposers the object must export.
689 #
690 MAPFILE.INT =           mapfile-intf

692 #
693 # LDLIBS32 and LDLIBS64 can be set in the environment to override the following
694 # assignments.
695 #
696 # These environment settings make sure that no libraries are searched outside
697 # of the local workspace proto area:
698 #       LDLIBS32=-YP,$ROOT/lib:$ROOT/usr/lib
699 #       LDLIBS64=-YP,$ROOT/lib/$MACH64:$ROOT/usr/lib/$MACH64
700 #
701 LDLIBS32 =      $(ENVLDLIBS1) $(ENVLDLIBS2) $(ENVLDLIBS3)
702 LDLIBS32 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib -L%/lib)
703 LDLIBS.cmd =    $(LDLIBS32)
704 LDLIBS.lib =    $(LDLIBS32)

706 LDLIBS64 =      $(ENVLDLIBS1:%=%/$(MACH64)) \
707                 $(ENVLDLIBS2:%=%/$(MACH64)) \
708                 $(ENVLDLIBS3:%=%/$(MACH64))
709 LDLIBS64 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib/$(MACH64) -L%/lib/$(MACH64))

711 #
712 # Define compilation macros.
713 #
714 COMPILE.c=      $(CC) $(CFLAGS) $(CPPFLAGS) -c
715 COMPILE64.c=    $(CC) $(CFLAGS64) $(CPPFLAGS) -c
716 COMPILE.cc=     $(CCC) $(CCFLAGS) $(CPPFLAGS) -c
717 COMPILE64.cc=   $(CCC) $(CCFLAGS64) $(CPPFLAGS) -c
718 COMPILE.s=      $(AS) $(ASFLAGS) $(AS_CPPFLAGS)
719 COMPILE64.s=    $(AS) $(ASFLAGS) $($(MACH64)_AS_XARCH) $(AS_CPPFLAGS)
720 COMPILE.d=      $(DTRACE) -G -32
721 COMPILE64.d=    $(DTRACE) -G -64
```



```
722 COMPILE.b=      $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))
723 COMPILE64.b=    $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))

725 CLASSPATH=      .
726 COMPILE.java=   $(JAVAC) $(JAVAFLAGS) -classpath $(CLASSPATH)

728 #
729 # Link time macros
730 #
731 CCNEEDED                = -lC
732 CCEXTNEEDED             = -lCrun -lCstd
733 $(__GNUC)CCNEEDED       = -L$(GCCLIBDIR) -lstdc++ -lgcc_s
734 $(__GNUC)CCEXTNEEDED    = $(CCNEEDED)

736 LINK.c=         $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
737 LINK64.c=       $(CC) $(CFLAGS64) $(CPPFLAGS) $(LDFLAGS)
738 NORUNPATH=      -norunpath -nolib
739 LINK.cc=        $(CCC) $(CCFLAGS) $(CPPFLAGS) $(NORUNPATH) \
740                 $(LDFLAGS) $(CCNEEDED)
741 LINK64.cc=      $(CCC) $(CCFLAGS64) $(CPPFLAGS) $(NORUNPATH) \
742                 $(LDFLAGS) $(CCNEEDED)

744 #
745 # lint macros
746 #
747 # Note that the undefine of __PRAGMA_REDEFINE_EXTNAME can be removed once
748 # ON is built with a version of lint that has the fix for 4484186.
749 #
750 ALWAYS_LINT_DEFS =      -errtags=yes -s
751 ALWAYS_LINT_DEFS +=     -erroff=E_PTRDIFF_OVERFLOW
752 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_NARROW_CONV
753 ALWAYS_LINT_DEFS +=     -U__PRAGMA_REDEFINE_EXTNAME
754 ALWAYS_LINT_DEFS +=     $(C99LMODE)
755 ALWAYS_LINT_DEFS +=     -errsecurity=$(SECLEVEL)
756 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_CREAT_WITHOUT_EXCL
757 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_FORBIDDEN_WARN_CREAT
758 # XX64 -- really only needed for amd64 lint
759 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_INT_TO_SMALL_INT
760 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_CONST_TO_SMALL_INT
761 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_TO_SMALL_INT
762 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_TO_PTR_FROM_INT
763 ALWAYS_LINT_DEFS +=     -erroff=E_COMP_INT_WITH_LARGE_INT
764 ALWAYS_LINT_DEFS +=     -erroff=E_INTEGRAL_CONST_EXP_EXPECTED
765 ALWAYS_LINT_DEFS +=     -erroff=E_PASS_INT_TO_SMALL_INT
766 ALWAYS_LINT_DEFS +=     -erroff=E_PTR_CONV_LOSES_BITS

768 # This forces lint to pick up note.h and sys/note.h from Devpro rather than
769 # from the proto area.  The note.h that ON delivers would disable NOTE().
770 ONLY_LINT_DEFS =        -I$(SPRO_VROOT)/prod/include/lint

772 SECLEVEL=       core
773 LINT.c=         $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS) $(CPPFLAGS) \
774                 $(ALWAYS_LINT_DEFS)
775 LINT64.c=       $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS64) $(CPPFLAGS) \
776                 $(ALWAYS_LINT_DEFS)
777 LINT.s=         $(LINT.c)

779 # For some future builds, NATIVE_MACH and MACH might be different.
780 # Therefore, NATIVE_MACH needs to be redefined in the
781 # environment as ‘uname -p‘ to override this macro.
782 #
783 # For now at least, we cross-compile amd64 on i386 machines.
784 NATIVE_MACH=    $(MACH:amd64=i386)

786 # Define native compilation macros
787 #
```

```
789 # Base directory where compilers are loaded.
790 # Defined here so it can be overridden by developer.
791 #
792 SPRO_ROOT=              $(BUILD_TOOLS)/SUNWspro
793 SPRO_VROOT=             $(SPRO_ROOT)/SS12

795 i386_GNU_ROOT=          $(SFW_ROOT)
796 sparc_GNU_ROOT=         $(SFW_ROOT)
797 arm_GNU_ROOT=           $(SFW_ROOT)

799 GNU_ROOT=               $($(MACH)_GNU_ROOT)
800 NATIVE_GNU_ROOT=        $($(NATIVE_MACH)_GNU_ROOT)

802 # Till SS12u1 formally becomes the NV CBE, LINT is hard
803 # coded to be picked up from the $SPRO_ROOT/sunstudio12.1/
804 # location. Impacted variables are sparc_LINT, sparcv9_LINT,
805 # i386_LINT, amd64_LINT.
806 # Reset them when SS12u1 is rolled out.
807 #

809 # Specify platform compiler versions for languages
810 # that we use (currently only c and c++).
811 #
812 sparc_CC=               $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_cc
813 $(__GNUC)sparc_CC=      $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_gcc
814 sparc_CCC=              $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_CC
815 $(__GNUC)sparc_CCC=     $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_g++
816 sparc_CPP=              /usr/ccs/lib/cpp
817 sparc_AS=               /usr/ccs/bin/as -xregsym=no
818 sparc_LD=               /usr/ccs/bin/ld
819 sparc_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint
820 sparc_CTFCONVERT=       $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfconvert.sparc
821 sparc_CTFMERGE=         $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfmerge.sparc

823 sparcv9_CC=             $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_cc
824 $(__GNUC64)sparcv9_CC=  $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_gcc
825 sparcv9_CCC=            $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_CC
826 $(__GNUC64)sparcv9_CCC= $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.sparc -_g++
827 sparcv9_CPP=            /usr/ccs/lib/cpp
828 sparcv9_AS=             /usr/ccs/bin/as -xregsym=no
829 sparcv9_LD=             /usr/ccs/bin/ld
830 sparcv9_LINT=           $(SPRO_ROOT)/sunstudio12.1/bin/lint
831 sparcv9_CTFCONVERT=     $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfconvert.sparc
832 sparcv9_CTFMERGE=               $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfmerge.sparc

834 i386_CC=               $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_cc
835 $(__GNUC)i386_CC=      $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_gcc
836 i386_CCC=              $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_CC
837 $(__GNUC)i386_CCC=     $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_g++
838 i386_CPP=              /usr/ccs/lib/cpp
839 i386_AS=               /usr/ccs/bin/as
840 $(__GNUC)i386_AS=      $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/aw.i386
841 i386_LD=               /usr/ccs/bin/ld
842 i386_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint
843 i386_CTFCONVERT=       $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfconvert.i386
844 i386_CTFMERGE=         $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfmerge.i386

846 amd64_CC=              $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_cc
847 $(__GNUC64)amd64_CC=   $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_gcc
848 amd64_CCC=             $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_CC
849 $(__GNUC64)amd64_CCC=  $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.i386 -_g++
850 amd64_CPP=             /usr/ccs/lib/cpp
851 amd64_AS=              $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/aw.i386
852 amd64_LD=              /usr/ccs/bin/ld
853 amd64_LINT=            $(SPRO_ROOT)/sunstudio12.1/bin/lint
```



```
854 amd64_CTFCONVERT=      $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfconvert.i386
855 amd64_CTFMERGE=        $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfmerge.i386

857 arm_CC=                $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.arm -_gcc
858 arm_CCC=               $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/cw.arm -_g++
859 arm_CPP=               /usr/ccs/lib/cpp
860 arm_AS=                $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/aw.arm
861 arm_LD=                /opt/armtc/usr/bin/ld
861 arm_LD=                /usr/ccs/bin/ld
862 arm_LINT=              /bin/false
863 arm_CTFCONVERT= $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfconvert.arm
864 arm_CTFMERGE=          $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/ctfmerge.arm

866 NATIVECC=              $($(NATIVE_MACH)_CC)
867 NATIVECCC=             $($(NATIVE_MACH)_CCC)
868 NATIVECPP=             $($(NATIVE_MACH)_CPP)
869 NATIVEAS=              $($(NATIVE_MACH)_AS)
870 NATIVELD=              $($(NATIVE_MACH)_LD)
871 NATIVELINT=            $($(NATIVE_MACH)_LINT)
872 NATIVECTFCONVERT=      $($(NATIVE_MACH)_CTFCONVERT)
873 NATIVECTFMERGE=        $($(NATIVE_MACH)_CTFMERGE)

875 #
876 # Makefile.master.64 overrides these settings
877 #
878 CC=                     $($(MACH)_CC)
879 CCC=                    $($(MACH)_CCC)
880 CPP=                    $($(MACH)_CPP)
881 AS=                     $($(MACH)_AS)
882 LD=                     $($(MACH)_LD)
883 LINT=                   $($(MACH)_LINT)
884 CTFCONVERT=             $($(MACH)_CTFCONVERT)
885 CTFMERGE=               $($(MACH)_CTFMERGE)

887 # The real compilers used for this build
888 CW_CC_CMD=              $(CC) -_compiler
889 CW_CCC_CMD=             $(CCC) -_compiler
890 REAL_CC=                $(CW_CC_CMD:sh)
891 REAL_CCC=               $(CW_CCC_CMD:sh)

893 # Pass -Y flag to cpp (method of which is release-dependent)
894 CCYFLAG=                -Y I,

896 BDIRECT=        -Bdirect
897 BDYNAMIC=       -Bdynamic
898 BLOCAL=         -Blocal
899 BNODIRECT=      -Bnodirect
900 BREDUCE=        -Breduce
901 BSTATIC=        -Bstatic

903 ZDEFS=          -zdefs
904 ZDIRECT=        -zdirect
905 ZIGNORE=        -zignore
906 ZINITFIRST=     -zinitfirst
907 ZINTERPOSE=     -zinterpose
908 ZLAZYLOAD=      -zlazyload
909 ZLOADFLTR=      -zloadfltr
910 ZMULDEFS=       -zmuldefs
911 ZNODEFAULTLIB=  -znodefaultlib
912 ZNODEFS=        -znodefs
913 ZNODELETE=      -znodelete
914 ZNODLOPEN=      -znodlopen
915 ZNODUMP=        -znodump
916 ZNOLAZYLOAD=    -znolazyload
917 ZNOLDYNSYM=     -znoldynsym
918 ZNORELOC=       -znoreloc
```

```
 919 ZNOVERSION=     -znoversion
 920 ZRECORD=        -zrecord
 921 ZREDLOCSYM=     -zredlocsym
 922 ZTEXT=          -ztext
 923 ZVERBOSE=       -zverbose

 925 GSHARED=        -G
 926 CCMT=           -mt

 928 # Handle different PIC models on different ISAs
 929 # (May be overridden by lower-level Makefiles)

 931 sparc_C_PICFLAGS =      -K pic
 932 sparcv9_C_PICFLAGS =    -K pic
 933 i386_C_PICFLAGS =       -K pic
 934 amd64_C_PICFLAGS =      -K pic
 935 arm_C_PICFLAGS =        -K pic
 936 C_PICFLAGS =            $($(MACH)_C_PICFLAGS)
 937 C_PICFLAGS64 =          $($(MACH64)_C_PICFLAGS)

 939 sparc_C_BIGPICFLAGS =   -K PIC
 940 sparcv9_C_BIGPICFLAGS = -K PIC
 941 i386_C_BIGPICFLAGS =    -K PIC
 942 amd64_C_BIGPICFLAGS =   -K PIC
 943 arm_C_BIGPICFLAGS =     -K PIC
 944 C_BIGPICFLAGS =         $($(MACH)_C_BIGPICFLAGS)
 945 C_BIGPICFLAGS64 =       $($(MACH64)_C_BIGPICFLAGS)

 947 # CC requires there to be no space between '-K' and 'pic' or 'PIC'.
 948 sparc_CC_PICFLAGS =     -Kpic
 949 sparcv9_CC_PICFLAGS =   -KPIC
 950 i386_CC_PICFLAGS =      -Kpic
 951 amd64_CC_PICFLAGS =     -Kpic
 952 arm_CC_PICFLAGS =       -Kpic
 953 CC_PICFLAGS =           $($(MACH)_CC_PICFLAGS)
 954 CC_PICFLAGS64 =         $($(MACH64)_CC_PICFLAGS)

 956 AS_PICFLAGS=           $(C_PICFLAGS)
 957 AS_BIGPICFLAGS=        $(C_BIGPICFLAGS)

 959 #
 960 # Default label for CTF sections
 961 #
 962 CTFCVTFLAGS=            -i -L VERSION
 963 $(SRCDBGBLD)CTFCVTFLAGS         += -g

 965 #
 966 # Override to pass module-specific flags to ctfmerge.  Currently used only by
 967 # krtld to turn on fuzzy matching, and source-level debugging to inhibit
 968 # stripping.
 969 #
 970 CTFMRGFLAGS=
 971 $(SRCDBGBLD)CTFMRGFLAGS         += -g


 974 CTFCONVERT_O            = $(CTFCONVERT) $(CTFCVTFLAGS) $@

 976 ELFSIGN_O=      $(TRUE)
 977 ELFSIGN_CRYPTO= $(ELFSIGN_O)
 978 ELFSIGN_OBJECT= $(ELFSIGN_O)

 980 # Rules (normally from make.rules) and macros which are used for post
 981 # processing files. Normally, these do stripping of the comment section
 982 # automatically.
 983 #    RELEASE_CM:        Should be editted to reflect the release.
 984 #    POST_PROCESS_O:    Post-processing for '.o' files.
```



```
 985 #    POST_PROCESS_A:    Post-processing for '.a' files (currently null).
 986 #    POST_PROCESS_SO:   Post-processing for '.so' files.
 987 #    POST_PROCESS:      Post-processing for executable files (no suffix).
 988 # Note that these macros are not completely generalized as they are to be
 989 # used with the file name to be processed following.
 990 #
 991 # It is left as an exercise to Release Engineering to embellish the generation
 992 # of the release comment string.
 993 #
 994 #       If this is a standard development build:
 995 #               compress the comment section (mcs -c)
 996 #               add the standard comment (mcs -a $(RELEASE_CM))
 997 #               add the development specific comment (mcs -a $(DEV_CM))
 998 #
 999 #       If this is an installation build:
1000 #               delete the comment section (mcs -d)
1001 #               add the standard comment (mcs -a $(RELEASE_CM))
1002 #               add the development specific comment (mcs -a $(DEV_CM))
1003 #
1004 #       If this is an release build:
1005 #               delete the comment section (mcs -d)
1006 #               add the standard comment (mcs -a $(RELEASE_CM))
1007 #
1008 # The following list of macros are used in the definition of RELEASE_CM
1009 # which is used to label all binaries in the build:
1010 #
1011 #       RELEASE         Specific release of the build, eg: 5.2
1012 #       RELEASE_MAJOR   Major version number part of $(RELEASE)
1013 #       RELEASE_MINOR   Minor version number part of $(RELEASE)
1014 #       VERSION         Version of the build (alpha, beta, Generic)
1015 #       PATCHID         If this is a patch this value should contain
1016 #                       the patchid value (eg: "Generic 100832-01"), otherwise
1017 #                       it will be set to $(VERSION)
1018 #       RELEASE_DATE    Date of the Release Build
1019 #       PATCH_DATE      Date the patch was created, if this is blank it
1020 #                       will default to the RELEASE_DATE
1021 #
1022 RELEASE_MAJOR=  5
1023 RELEASE_MINOR=  11
1024 RELEASE=        $(RELEASE_MAJOR).$(RELEASE_MINOR)
1025 VERSION=        SunOS Development
1026 PATCHID=        $(VERSION)
1027 RELEASE_DATE=   release date not set
1028 PATCH_DATE=     $(RELEASE_DATE)
1029 RELEASE_CM=     "@($(POUND_SIGN))SunOS $(RELEASE) $(PATCHID) $(PATCH_DATE)"
1030 DEV_CM=         "@($(POUND_SIGN))SunOS Internal Development: non-nightly build"

1032 PROCESS_COMMENT=  @?${MCS} -d -a $(RELEASE_CM) -a $(DEV_CM)
1033 $(RELEASE_BUILD)PROCESS_COMMENT=   @?${MCS} -d -a $(RELEASE_CM)

1035 STRIP_STABS=                       :
1036 $(RELEASE_BUILD)STRIP_STABS=       $(STRIP) -x $@
1037 $(SRCDBGBLD)STRIP_STABS=           :

1039 POST_PROCESS_O=
1040 POST_PROCESS_A=
1041 POST_PROCESS_SO=        $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
1042                         $(ELFSIGN_OBJECT)
1043 POST_PROCESS=           $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
1044                         $(ELFSIGN_OBJECT)

1046 #
1047 # chk4ubin is a tool that inspects a module for a symbol table
1048 # ELF section size which can trigger an OBP bug on older platforms.
1049 # This problem affects only specific sun4u bootable modules.
1050 #
```

```
1051 CHK4UBIN=       $(ONBLD_TOOLS)/bin/$(NATIVE_MACH)/chk4ubin
1052 CHK4UBINFLAGS=
1053 CHK4UBINARY=    $(CHK4UBIN) $(CHK4UBINFLAGS) $@

1055 #
1056 # PKGARCHIVE specifies the default location where packages should be
1057 # placed if built.
1058 #
1059 $(RELEASE_BUILD)PKGARCHIVESUFFIX=       -nd
1060 PKGARCHIVE=$(SRC)/../../packages/$(MACH)/nightly$(PKGARCHIVESUFFIX)

1062 #
1063 # The repositories will be created with these publisher settings.  To
1064 # update an image to the resulting repositories, this must match the
1065 # publisher name provided to "pkg set-publisher."
1066 #
1067 PKGPUBLISHER_REDIST=    on-nightly
1068 PKGPUBLISHER_NONREDIST= on-extra

1070 #       Default build rules which perform comment section post-processing.
1071 #
1072 .c:
1073         $(LINK.c) -o $@ $< $(LDLIBS)
1074         $(POST_PROCESS)
1075 .c.o:
1076         $(COMPILE.c) $(OUTPUT_OPTION) $< $(CTFCONVERT_HOOK)
1077         $(POST_PROCESS_O)
1078 .c.a:
1079         $(COMPILE.c) -o $% $<
1080         $(PROCESS_COMMENT) $%
1081         $(AR) $(ARFLAGS) $@ $%
1082         $(RM) $%
1083 .s.o:
1084         $(COMPILE.s) -o $@ $<
1085         $(POST_PROCESS_O)
1086 .s.a:
1087         $(COMPILE.s) -o $% $<
1088         $(PROCESS_COMMENT) $%
1089         $(AR) $(ARFLAGS) $@ $%
1090         $(RM) $%
1091 .cc:
1092         $(LINK.cc) -o $@ $< $(LDLIBS)
1093         $(POST_PROCESS)
1094 .cc.o:
1095         $(COMPILE.cc) $(OUTPUT_OPTION) $<
1096         $(POST_PROCESS_O)
1097 .cc.a:
1098         $(COMPILE.cc) -o $% $<
1099         $(AR) $(ARFLAGS) $@ $%
1100         $(PROCESS_COMMENT) $%
1101         $(RM) $%
1102 .y:
1103         $(YACC.y) $<
1104         $(LINK.c) -o $@ y.tab.c $(LDLIBS)
1105         $(POST_PROCESS)
1106         $(RM) y.tab.c
1107 .y.o:
1108         $(YACC.y) $<
1109         $(COMPILE.c) -o $@ y.tab.c $(CTFCONVERT_HOOK)
1110         $(POST_PROCESS_O)
1111         $(RM) y.tab.c
1112 .l:
1113         $(RM) $*.c
1114         $(LEX.l) $< > $*.c
1115         $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
1116         $(POST_PROCESS)
```



```
1117         $(RM) $*.c
1118 .l.o:
1119         $(RM) $*.c
1120         $(LEX.l) $< > $*.c
1121         $(COMPILE.c) -o $@ $*.c $(CTFCONVERT_HOOK)
1122         $(POST_PROCESS_O)
1123         $(RM) $*.c

1125 .bin.o:
1126         $(COMPILE.b) -o $@ $<
1127         $(POST_PROCESS_O)

1129 .java.class:
1130         $(COMPILE.java) $<

1132 # Bourne and Korn shell script message catalog build rules.
1133 # We extract all gettext strings with sed(1) (being careful to permit
1134 # multiple gettext strings on the same line), weed out the dups, and
1135 # build the catalogue with awk(1).

1137 .sh.po .ksh.po:
1138         $(SED) -n -e ":a"                                   \
1139                   -e "h"                                    \
1140                   -e "s/.*gettext *\(\"[^\"]*\"\).*/\1/p"   \
1141                   -e "x"                                    \
1142                   -e "s/\(.*\)gettext *\"[^\"]*\"\(.*\)/\1\2/"  \
1143                   -e "t a"                                  \
1144                 $< | sort -u | awk ’{ print "msgid\t" $$0 "\nmsgstr" }’ > $@

1146 #
1147 # Python and Perl executable and message catalog build rules.
1148 #
1149 .SUFFIXES: .pl .pm .py .pyc

1151 .pl:
1152         $(RM) $@;
1153         $(SED) -e "s@TEXT_DOMAIN@\"$(TEXT_DOMAIN)\"@" $< > $@;
1154         $(CHMOD) +x $@

1156 .py:
1157         $(RM) $@; $(CAT) $< > $@; $(CHMOD) +x $@

1159 .py.pyc:
1160         $(RM) $@
1161         $(PYTHON) -mpy_compile $<
1162         @[ $(<)c = $@ ] || $(MV) $(<)c $@

1164 .py.po:
1165         $(GNUXGETTEXT) $(GNUXGETFLAGS) -d $(<F:%.py=%) $< ;

1167 .pl.po .pm.po:
1168         $(XGETTEXT) $(XGETFLAGS) -d $(<F) $< ;
1169         $(RM)   $@ ;
1170         $(SED) "/^domain/d" < $(<F).po > $@ ;
1171         $(RM) $(<F).po

1173 #
1174 # When using xgettext, we want messages to go to the default domain,
1175 # rather than the specified one.  This special version of the
1176 # COMPILE.cpp macro effectively prevents expansion of TEXT_DOMAIN,
1177 # causing xgettext to put all messages into the default domain.
1178 #
1179 CPPFORPO=$(COMPILE.cpp:\"$(TEXT_DOMAIN)\"=TEXT_DOMAIN)

1181 .c.i:
1182         $(CPPFORPO) $< > $@
```

```
1184 .h.i:
1185         $(CPPFORPO) $< > $@

1187 .y.i:
1188         $(YACC) -d $<
1189         $(CPPFORPO) y.tab.c  > $@
1190         $(RM) y.tab.c

1192 .l.i:
1193         $(LEX) $<
1194         $(CPPFORPO) lex.yy.c  > $@
1195         $(RM) lex.yy.c

1197 .c.po:
1198         $(CPPFORPO) $< > $<.i
1199         $(BUILD.po)

1201 .y.po:
1202         $(YACC) -d $<
1203         $(CPPFORPO) y.tab.c  > $<.i
1204         $(BUILD.po)
1205         $(RM) y.tab.c

1207 .l.po:
1208         $(LEX) $<
1209         $(CPPFORPO) lex.yy.c  > $<.i
1210         $(BUILD.po)
1211         $(RM) lex.yy.c

1213 #
1214 # Rules to perform stylistic checks
1215 #
1216 .SUFFIXES: .x .xml .check .xmlchk

1218 .h.check:
1219         $(DOT_H_CHECK)

1221 .x.check:
1222         $(DOT_X_CHECK)

1224 .xml.xmlchk:
1225         $(MANIFEST_CHECK)

1227 #
1228 # Include rules to render automated sccs get rules "safe".
1229 #
1230 include $(SRC)/Makefile.noget
```

```
**********************************************************
    3153 Sun Jan 25 13:29:10 2015
new/usr/src/uts/arm/sys/bootconf.h
fakebop: use a memlist to keep track of physical memory
**********************************************************
```

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

/*
 * Copyright (c) 2013 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
 */


#ifndef _SYS_BOOTCONF_H
#define _SYS_BOOTCONF_H

/*
 * Boot time configuration information objects
 */

#include <sys/types.h>
#include <sys/memlist.h>
#include <sys/ccompile.h>
#include <net/if.h>                     /* for IFNAMSIZ */

#ifdef __cplusplus
extern "C" {
#endif

/*
 * Masks for bsys_alloc memory allocator. These overlap with the ones for intel
 * and sun because they're used by the common kernel.
 */
#define BO_NO_ALIGN     0x00001000
#define BO_ALIGN_DONTCARE       -1

struct bsys_mem {
        struct memlist  physinstalled;
};

#endif /* ! codereview */
#define BO_VERSION      1       /* bootops interface revision */

typedef struct bootops {
        uint_t  bsys_version;
        struct bsys_mem boot_mem;
#endif /* ! codereview */
        caddr_t (*bsys_alloc)(struct bootops *, caddr_t, size_t, int);
        void    (*bsys_free)(struct bootops *, caddr_t, size_t);
        int     (*bsys_getproplen)(struct bootops *, const char *);
        int     (*bsys_getprop)(struct bootops *, const char *, void *);
        void    (*bsys_printf)(struct bootops *, const char *, ...);
} bootops_t;

#define BOP_GETVERSION(bop)             ((bop)->bsys_version)
#define BOP_ALLOC(bop, virthint, size, align)   \
```



```
                ((bop)->bsys_alloc)(bop, virthint, size, align)
#define BOP_FREE(bop, virt, size)       ((bop)->bsys_free)(bop, virt, size)
#define BOP_GETPROPLEN(bop, name)       ((bop)->bsys_getproplen)(bop, name)
#define BOP_GETPROP(bop, name, buf)     ((bop)->bsys_getprop)(bop, name, buf)
#define BOP_PUTSARG(bop, msg, arg)      ((bop)->bsys_printf)(bop, msg, arg)

/*
 * Boot configuration information
 */

#define BO_MAXFSNAME    16
#define BO_MAXOBJNAME   256

struct bootobj {
        char    bo_fstype[BO_MAXFSNAME];        /* vfs type name (e.g. nfs) */
        char    bo_name[BO_MAXOBJNAME];         /* name of object */
        int     bo_flags;                       /* flags, see below */
        int     bo_size;                        /* number of blocks */
        struct vnode *bo_vp;                    /* vnode of object */
        char    bo_devname[BO_MAXOBJNAME];
        char    bo_ifname[BO_MAXOBJNAME];
        int     bo_ppa;
};

/*
 * flags
 */
#define BO_VALID        0x01    /* all information in object is valid */
#define BO_BUSY         0x02    /* object is busy */

extern struct bootobj rootfs;
extern struct bootobj swapfile;

extern char *default_path;
extern int modrootloaded;
extern char kern_bootargs[];
extern char kern_bootfile[];

extern int strplumb(void);
extern char *strplumb_get_netdev_path(void);
extern void consconfig(void);
extern void release_bootstrap(void);

extern void bop_panic(const char *);
extern void boot_prop_finish(void);
extern void bop_printf(struct bootops *, const char *, ...);

extern struct bootops *bootops;
extern int netboot;
extern char *dhcack;
extern int dhcacklen;
extern char dhcifname[IFNAMSIZ];

extern char *netdev_path;

#ifdef __cplusplus
}
#endif

#endif /* _SYS_BOOTCONF_H */
```

```
**********************************************************
    1619 Sun Jan 25 13:29:10 2015
new/usr/src/uts/armv6/bcm2835/Makefile
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible.  It claims to
supporte ELF file loading, but what it does is crazy.  It loads the ELF file
into memory, gets the start address from the header, converts it into file
offset, adds it to the address where the file was loaded and jumps there.
This is very wrong.  So, instead of booting the loader as an ELF file, we
objcopy it into a plain ol' binary image.  This should be safe because (1)
the loader has no relocations left, (2) whatever benefit we lose from not
having the whole ELF file in memory is only temporary until the loader hands
off control to unix.
Finally, we force the entry point to appear at the beginning of the binary
file by moving _start into its own section (.text.init) and using the
mapfile to put that section before everything else.
**********************************************************
```

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

#
# Copyright (c) 2013, Joyent, Inc.  All rights reserved.
# Copyright (c) 2015, Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
#

UTSBASE = ../..

include $(UTSBASE)/armv6/bcm2835/Makefile.bcm2835

def             :=      TARGET= def
all             :=      TARGET= all
install         :=      TARGET= install
install_h       :=      TARGET= install_h
clean           :=      TARGET= clean
clobber         :=      TARGET= clobber
lint            :=      TARGET= lint
lintlib         :=      TARGET= lintlib
modlintlib      :=      TARGET= modlintlib
modlist         :=      TARGET= modlist
modlist         :=      NO_STATE= -K $$MODSTATE$$$$
clean.lint      :=      TARGET= clean.lint
check           :=      TARGET= check


#
#       Default build targets.
#
.KEEP_STATE:

def all clean clobber clean.lint: unix loader .WAIT \
def all clean clobber clean.lint: unix .WAIT \
                $(BCM2835_CPU_KMODS) $(BCM2835_KMODS)

modlist:        unix \
                loader \
#endif /* ! codereview */
                $(BCM2835_CPU_KMODS) $(BCM2835_KMODS)
```



```
IMPLEMENTED_PLATFORM = Raspberry,Pi

install: $(ROOT_BCM2835_DIR) $(USR_BCM2835_DIR) \
        $(USR_BCM2835_INC_DIR) \
        $(USR_BCM2835_SBIN_DIR) \
        $(USR_BCM2835_LIB_DIR) \
        $(BCM2835_CRYPTO_LINKS) \
        unix .WAIT $(BCM2835_CPU_KMODS) \
        $(BCM2835_KMODS) loader
        $(BCM2835_KMODS)

unix loader $(BCM2835_KMODS) $(BCM2835_CPU_KMODS): FRC
unix $(BCM2835_KMODS) $(BCM2835_CPU_KMODS): FRC
        @cd $@; pwd; $(MAKE) $(NO_STATE) $(TARGET)

install_h check:        FRC

#
#       Include common targets.
#
include $(UTSBASE)/armv6/bcm2835/Makefile.targ
```

```
**********************************************************
     655 Sun Jan 25 13:29:10 2015
new/usr/src/uts/armv6/bcm2835/Makefile.files
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible.  It claims to
supporte ELF file loading, but what it does is crazy.  It loads the ELF file
into memory, gets the start address from the header, converts it into file
offset, adds it to the address where the file was loaded and jumps there.
This is very wrong.  So, instead of booting the loader as an ELF file, we
objcopy it into a plain ol' binary image.  This should be safe because (1)
the loader has no relocations left, (2) whatever benefit we lose from not
having the whole ELF file in memory is only temporary until the loader hands
off control to unix.
Finally, we force the entry point to appear at the beginning of the binary
file by moving _start into its own section (.text.init) and using the
mapfile to put that section before everything else.
**********************************************************
```

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

#
# Copyright (c) 2013, Joyent, Inc.  All rights reserved.
# Copyright (c) 2015, Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
#

BCM2835_OBJS =                  \
        bcm2835_bsmdep.o        \
        boot_console.o          \
        locore.o                \
        miniuart.o

BCM2835_LOADER_OBJS = \
        bcm2835_ldep.o
#endif /* ! codereview */
```

****
*** 845 Sun Jan 25 13:29:11 2015**
new/usr/src/uts/armv6/bcm2835/Makefile.rules
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible. It claims to
supporte ELF file loading, but what it does is crazy. It loads the ELF file
into memory, gets the start address from the header, converts it into file
offset, adds it to the address where the file was loaded and jumps there.
This is very wrong. So, instead of booting the loader as an ELF file, we
objcopy it into a plain ol' binary image. This should be safe because (1)
the loader has no relocations left, (2) whatever benefit we lose from not
having the whole ELF file in memory is only temporary until the loader hands
off control to unix.
Finally, we force the entry point to appear at the beginning of the binary
file by moving _start into its own section (.text.init) and using the
mapfile to put that section before everything else.
****

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

#
# Copyright (c) 2013, Joyent, Inc.  All rights reserved.
# Copyright (c) 2015, Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
#

$(OBJS_DIR)/%.o:                $(UTSBASE)/$(PLATFORM)/bcm2835/loader/%.c
        $(COMPILE.c) -o $@ $<
#endif /* ! codereview */

$(OBJS_DIR)/%.o:                $(UTSBASE)/$(PLATFORM)/bcm2835/os/%.c
        $(COMPILE.c) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/$(PLATFORM)/bcm2835/ml/%.s
        $(COMPILE.s) -o $@ $<

INC_PATH += -I$(UTSBASE)/$(PLATFORM) -I$(UTSBASE)/$(PLATFORM)/bcm2835/
```

```
**********************************************************
    1094 Sun Jan 25 13:29:11 2015
new/usr/src/uts/armv6/bcm2835/conf/Mapfile
bcm2835: resync unix mapfile with qvpb
The mapfile used by bcm2835's unix was left behind when a bunch of kernel
addresses got changed.  This commit brings it up to date.
**********************************************************
```

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

#
# Copyright (c) 2013, Joyent, Inc.  All rights reserved.
#

$mapfile_version 2

# NB: Order matters for name-based entry!

#endif /* ! codereview */
LOAD_SEGMENT exception_vector {
        FLAGS = READ EXECUTE;
        NOHDR;
        PADDR = 0xFFFF0000;
        VADDR = 0xFFFF0000;
        ALIGN = 0x1000;
#endif /* ! codereview */
        OS_ORDER = .exception_vector;
        ASSIGN_SECTION { IS_NAME = .exception_vector };
};

LOAD_SEGMENT text {
        FLAGS = READ EXECUTE;
        NOHDR;
        VADDR = 0xFE800000;
#       PADDR = 0x8080;
#       VADDR = 0x8080;
        PADDR = 0x0c100000;
        VADDR = 0x0c100000;
        OS_ORDER = .text;
        ASSIGN_SECTION {
                TYPE = PROGBITS;
                FLAGS = ALLOC !WRITE;
        };
};

#
# start the data segment on a new 4MB page boundary
#
#endif /* ! codereview */
LOAD_SEGMENT data {
        FLAGS = READ WRITE EXECUTE;
        NOHDR;
        VADDR = 0xFEC00000;
#       PADDR = 0x80000;
#       VADDR = 0x80000;
        PADDR = 0x0c200000;
        VADDR = 0x0c200000;
        OS_ORDER = .data;
```



```
        ASSIGN_SECTION {
                TYPE = PROGBITS;
                FLAGS = ALLOC  WRITE;
        };
};
```

*_____unchanged_portion_omitted_*

```
**********************************************************
     977 Sun Jan 25 13:29:11 2015
new/usr/src/uts/armv6/bcm2835/conf/Mapfile.loader
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible.  It claims to
supporte ELF file loading, but what it does is crazy.  It loads the ELF file
into memory, gets the start address from the header, converts it into file
offset, adds it to the address where the file was loaded and jumps there.
This is very wrong.  So, instead of booting the loader as an ELF file, we
objcopy it into a plain ol' binary image.  This should be safe because (1)
the loader has no relocations left, (2) whatever benefit we lose from not
having the whole ELF file in memory is only temporary until the loader hands
off control to unix.
Finally, we force the entry point to appear at the beginning of the binary
file by moving _start into its own section (.text.init) and using the
mapfile to put that section before everything else.
**********************************************************
```

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

#
# Copyright (c) 2013, Joyent, Inc.  All rights reserved.
# Copyright (c) 2015, Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#

$mapfile_version 2

# NB: Order matters for name-based entry!
LOAD_SEGMENT text {
        FLAGS = READ EXECUTE;
        NOHDR;
        PADDR = 0x8000;
        VADDR = 0x8000;
        MAX_SIZE = 0x2000;
        OS_ORDER = .text.init .text;
        ASSIGN_SECTION {
                IS_NAME = .text.init;
        };
        ASSIGN_SECTION {
                TYPE = PROGBITS;
                FLAGS = ALLOC !WRITE;
        };
};

LOAD_SEGMENT data {
        FLAGS = READ WRITE;
        NOHDR;
        OS_ORDER = .data;
        ASSIGN_SECTION {
                TYPE = PROGBITS;
                FLAGS = ALLOC WRITE;
        };
};
#endif /* ! codereview */
```

```
**********************************************************
    2105 Sun Jan 25 13:29:11 2015
new/usr/src/uts/armv6/bcm2835/loader/Makefile
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible.  It claims to
supporte ELF file loading, but what it does is crazy.  It loads the ELF file
into memory, gets the start address from the header, converts it into file
offset, adds it to the address where the file was loaded and jumps there.
This is very wrong.  So, instead of booting the loader as an ELF file, we
objcopy it into a plain ol' binary image.  This should be safe because (1)
the loader has no relocations left, (2) whatever benefit we lose from not
having the whole ELF file in memory is only temporary until the loader hands
off control to unix.
Finally, we force the entry point to appear at the beginning of the binary
file by moving _start into its own section (.text.init) and using the
mapfile to put that section before everything else.
**********************************************************
```

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

#
# Copyright (c) 2013, Joyent, Inc.  All rights reserved.
# Copyright (c) 2015, Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../../..

#
#       Define the module and object file sets.
#
LOADER          = loader
LOADER_BIN      = $(OBJS_DIR)/$(LOADER)
LOADER_ELF      = $(LOADER).elf
LOADER_ELF_BIN  = $(OBJS_DIR)/$(LOADER_ELF)
ROOTMODULE      = $(ROOT_BCM2835_KERN_DIR)/$(LOADER)
OBJECTS         = $(ARM_LOADER_OBJS:%=$(OBJS_DIR)/%) \
                  $(BCM2835_LOADER_OBJS:%=$(OBJS_DIR)/%)

#
#       Include common rules.
#
include $(UTSBASE)/armv6/bcm2835/Makefile.bcm2835

#
#       Define targets
#
ALL_TARGET      = $(LOADER_BIN)
INSTALL_TARGET  = $(LOADER_BIN) $(ROOTMODULE)

#
#       Overrides
#
CLEANFILES      += $(OBJECTS)

CLOBBERFILES    = $(CLEANFILES) $(LOADER_BIN)
```



```
CFLAGS          += $(CCVERBOSE) \
                   -DELF_TARGET_ARM

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-but-set-variable

# The mapfile used to link unix
MAPFILE         = $(UTSBASE)/armv6/bcm2835/conf/Mapfile.loader

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

install:        $(INSTALL_DEPS)

symcheck:       $(SYM_DEPS)

$(LOADER_ELF_BIN): $(OBJECTS)
        $(LD) -dy -b -znointerp -o $@ -e _start -M $(MAPFILE) \
        $(OBJECTS)
        $(CTFMERGE) $(CTFMRGFLAGS) -L VERSION -o $(LOADER_ELF_BIN) $(OBJECTS)

$(LOADER_BIN): $(LOADER_ELF_BIN)
        /opt/armtc/usr/gnu/bin/gobjcopy $(LOADER_ELF_BIN) -O binary $(LOADER_BIN


#
#       Include common targets.
#
include $(UTSBASE)/armv6/bcm2835/Makefile.targ
#endif /* ! codereview */
```

```
**********************************************************
    4664 Sun Jan 25 13:29:11 2015
new/usr/src/uts/armv6/bcm2835/loader/bcm2835_ldep.c
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible.  It claims to
supporte ELF file loading, but what it does is crazy.  It loads the ELF file
into memory, gets the start address from the header, converts it into file
offset, adds it to the address where the file was loaded and jumps there.
This is very wrong.  So, instead of booting the loader as an ELF file, we
objcopy it into a plain ol' binary image.  This should be safe because (1)
the loader has no relocations left, (2) whatever benefit we lose from not
having the whole ELF file in memory is only temporary until the loader hands
off control to unix.
Finally, we force the entry point to appear at the beginning of the binary
file by moving _start into its own section (.text.init) and using the
mapfile to put that section before everything else.
**********************************************************
```

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

/*
 * Copyright (c) 2013 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 */

#include <sys/elf.h>
#include <sys/atag.h>

/*
 * The primary serial console that we end up using is not in fact a normal UART,
 * but is instead actually a mini-uart that shares interrupts and registers with
 * the SPI masters as well. While the RPi also supports another more traditional
 * UART, that isn't what we are actually hooking up to generally with the
 * adafruit cable. We already wasted our time having to figure that out. -_-
 */

#define AUX_BASE        0x20215000
#define AUX_ENABLES     0x4
#define AUX_MU_IO_REG   0x40
#define AUX_MU_IER_REG  0x44
#define AUX_MU_IIR_REG  0x48
#define AUX_MU_LCR_REG  0x4C
#define AUX_MU_MCR_REG  0x50
#define AUX_MU_LSR_REG  0x54
#define AUX_MU_CNTL_REG 0x60
#define AUX_MU_BAUD     0x68

#define AUX_MU_RX_READY 0x01
#define AUX_MU_TX_READY 0x20

/*
 * For the mini UART, all we care about are pins 14 and 15 for the UART.
 * Specifically, alt5 for GPIO14 is TXD1 and GPIO15 is RXD1. Those are
 * controlled by FSEL1.
 */
#define GPIO_BASE       0x20200000
#define GPIO_FSEL1      0x4
#define GPIO_PUD        0x94
```



```
#define GPIO_PUDCLK0    0x98

#define GPIO_SEL_ALT5   0x2
#define GPIO_UART_MASK  0xfffc0fff
#define GPIO_UART_TX_SHIFT      12
#define GPIO_UART_RX_SHIFT      15

#define GPIO_PUD_DISABLE        0x0
#define GPIO_PUDCLK_UART        0x0000c000

static __GNU_INLINE uint32_t arm_reg_read(uint32_t reg)
{
        volatile uint32_t *ptr = (volatile uint32_t *)reg;

        return *ptr;
}

static __GNU_INLINE void arm_reg_write(uint32_t reg, uint32_t val)
{
        volatile uint32_t *ptr = (volatile uint32_t *)reg;

        *ptr = val;
}

/*
 * A simple nop
 */
static void
bcm2835_miniuart_nop(void)
{
        __asm__ volatile("mov r0, r0\n" : : :);
}

void fakeload_backend_putc(int);

static void
fakeload_puts(const char *str)
{
        while (*str != '\0') {
                fakeload_backend_putc(*str);
                str++;
        }
}

void
fakeload_backend_init(void)
{
        uint32_t v;
        int i;

        /* Enable the mini UAT */
        arm_reg_write(AUX_BASE + AUX_ENABLES, 0x1);

        /* Disable interrupts */
        arm_reg_write(AUX_BASE + AUX_MU_IER_REG, 0x0);

        /* Disable the RX and TX */
        arm_reg_write(AUX_BASE + AUX_MU_CNTL_REG, 0x0);

        /*
         * Enable 8-bit word length. External sources tell us the PRM is buggy
         * here and that even though bit 1 is reserved, we need to actually set
         * it to get 8-bit words.
         */
        arm_reg_write(AUX_BASE + AUX_MU_LCR_REG, 0x3);
```

```
116         /* Set RTS high */
117         arm_reg_write(AUX_BASE + AUX_MU_MCR_REG, 0x0);

119         /* Disable interrupts */
120         arm_reg_write(AUX_BASE + AUX_MU_IER_REG, 0x0);

122         /* Set baud rate */
123         arm_reg_write(AUX_BASE + AUX_MU_IIR_REG, 0xc6);
124         arm_reg_write(AUX_BASE + AUX_MU_BAUD, 0x10e);

126         /* TODO: Factor out the gpio bits */
127         v = arm_reg_read(GPIO_BASE + GPIO_FSEL1);
128         v &= GPIO_UART_MASK;
129         v |= GPIO_SEL_ALT5 << GPIO_UART_RX_SHIFT;
130         v |= GPIO_SEL_ALT5 << GPIO_UART_TX_SHIFT;
131         arm_reg_write(GPIO_BASE + GPIO_FSEL1, v);

133         arm_reg_write(GPIO_BASE + GPIO_PUD, GPIO_PUD_DISABLE);
134         for (i = 0; i < 150; i++)
135                 bcm2835_miniuart_nop();
136         arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, GPIO_PUDCLK_UART);
137         for (i = 0; i < 150; i++)
138                 bcm2835_miniuart_nop();
139         // XXX: GPIO_PUD_DISABLE again?
140         arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, 0);

142         /* Finally, go back and enable RX and TX */
143         arm_reg_write(AUX_BASE + AUX_MU_CNTL_REG, 0x3);
144 }

146 void
147 fakeload_backend_putc(int c)
148 {
149         if (c == '\n')
150                 fakeload_backend_putc('\r');

152         for (;;) {
153                 if (arm_reg_read(AUX_BASE + AUX_MU_LSR_REG) & AUX_MU_TX_READY)
154                         break;
155         }
156         arm_reg_write(AUX_BASE + AUX_MU_IO_REG, c & 0x7f);
157 }

159 /*
160  * Add a map for the uart.
161  */
162 void
163 fakeload_backend_addmaps(atag_header_t *chain)
164 {
165         atag_illumos_mapping_t aim;

167         aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
168         aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
169         aim.aim_paddr = GPIO_BASE;
170         aim.aim_vaddr = GPIO_BASE;
171         aim.aim_vlen = 0x1000;
172         aim.aim_plen = 0x1000;
173         aim.aim_mapflags = PF_R | PF_W | PF_NORELOC | PF_DEVICE;
174         atag_append(chain, &aim.aim_header);

176         aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
177         aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
178         aim.aim_paddr = AUX_BASE;
179         aim.aim_vaddr = AUX_BASE;
180         aim.aim_vlen = 0x1000;
181         aim.aim_plen = 0x1000;
```



```
182         aim.aim_mapflags = PF_R | PF_W | PF_NORELOC | PF_DEVICE;
183         atag_append(chain, &aim.aim_header);
184 }
185 #endif /* ! codereview */
```

```
**********************************************************
    3457 Sun Jan 25 13:29:11 2015
new/usr/src/uts/armv6/bcm2835/ml/locore.s
bcm2835: remove workaround for u-boot
Since we are not using u-boot, we don't need a work-around for it.
bcm2835: enable access to p10 and p11
When determining whether or not we are running on a supported processor, we
call arm_cpuid_vfpidreg.  This function attempts to get the FPSID which
involves talking to p10.  It however turns out that after a reset only p14
and p15 are accessible.  So, before handing off control to _fakebop_start,
let's enable access to p10 and p11 (privileged mode only).
NOTE: qemu doesn't seem to behave the same - it lets us access p10 & p11
without us doing anything special.
**********************************************************
```

```
 1 /*
 2  * This file and its contents are supplied under the terms of the
 3  * Common Development and Distribution License ("CDDL"), version 1.0.
 4  * You may only use this file in accordance with the terms of version
 5  * 1.0 of the CDDL.
 6  *
 7  * A full copy of the text of the CDDL should have accompanied this
 8  * source.  A copy of the CDDL is also available via the Internet at
 9  * http://www.illumos.org/license/CDDL.
10  */

12 /*
13  * Copyright 2013 (c) Joyent, Inc. All rights reserved.
14  * Copyright 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15 #endif /* ! codereview */
16  */

18 #include <sys/asm_linkage.h>
19 #include <sys/machparam.h>
20 #include <sys/cpu_asm.h>

22 /*
23  * Every story needs a beginning. This is ours.
24  */

26 /*
27  * We are in a primordial world here. The BMC2835 is going to come along and
28  * boot us at _start. Normally we would go ahead and use a main() function, but
29  * for now, we'll do that ourselves. As we've started the world, we also need to
30  * set up a few things about us, for example our stack pointer. To help us out,
31  * it's useful to remember the rough memory map. Remember, this is for physcial
32  * addresses. There is no virtual memory here. These sizes are often manipulated
33  * by the 'configuration' in the bootloader.
34  *
35  * +----------------+ <---- Max physical memory
36  * |                |
37  * |                |
38  * |                |
39  * +----------------+
40  * |                |
41  * |      I/O       |
42  * |  Peripherals   |
43  * |                |
44  * +----------------+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
45  * |                |
46  * |     Main       |
47  * |    Memory      |
48  * |                |
49  * +----------------+ <---- Top of SDRAM
50  * |                |
51  * |       VC       |
52  * |     SDRAM      |
```



```
53  * |                |
54  * +----------------+ <---- Split determined by bootloader config
55  * |                |
56  * |      ARM       |
57  * |     SDRAM      |
58  * |                |
59  * +----------------+ <---- Bottom of physical memory 0x00000000
60  *
61  * With the Raspberry Pi Model B, we have 512 MB of SDRAM. That means we have a
62  * range of addresses from [0, 0x20000000). If we assume that the minimum amount
63  * of DRAM is given to the GPU - 32 MB, that means we really have the following
64  * range: [0, 0x1e000000).
65  *
66  * By default, this binary will be loaded into 0x8000. For now, that means we
67  * will set our initial stack to 0x10000000.
68  */

70 /*
71  * Recall that _start is the traditional entry point for an ELF binary.
72  */
73         ENTRY(_start)
74         ldr sp, =t0stack
75         ldr r4, =DEFAULTSTKSZ
76         add sp, r4
77         bic sp, sp, #0xff

79         /*
80          * establish bogus stacks for exceptional CPU states, our exception
81          * code should never make use of these, and we want loud and violent
82          * failure should we accidentally try.
83          */
84         cps #(CPU_MODE_UND)
85         mov sp, #-1
86         cps #(CPU_MODE_ABT)
87         mov sp, #-1
88         cps #(CPU_MODE_FIQ)
89         mov sp, #-1
90         cps #(CPU_MODE_IRQ)
91         mov sp, #-1
92         cps #(CPU_MODE_SVC)

94         /* Enable highvecs (moves the base of the exception vector) */
95         mrc     p15, 0, r3, c1, c0, 0
96         mov     r4, #1
97         lsl     r4, r4, #13
98         orr     r3, r3, r4
99         mcr     p15, 0, r3, c1, c0, 0

101        /* Disable A (disables strict alignment checks) */
102        mrc     p15, 0, r3, c1, c0, 0
103        bic     r3, r3, #2
104        mcr     p15, 0, r3, c1, c0, 0

106        /* Enable access to p10 and p11 (privileged mode only) */
107        mrc     p15, 0, r0, c1, c0, 2
108        orr     r0, #0x00500000
109        mcr     p15, 0, r0, c1, c0, 2

14         /*
15          * XXX Currently we're using u-boot to allow us to make forward progress
16          * while the .data section is a bit tumultuous. It loads that, but we
17          * can say for certain that it does not correctly pass in the machid and
18          * tagstart. Since we know what it is, we manually fix it up here.
19          */
20         mov r2,#0x100
111        bl _fakebop_start
```

```
 112         SET_SIZE(_start)
______unchanged_portion_omitted_
```

```
**********************************************************
    3665 Sun Jan 25 13:29:12 2015
new/usr/src/uts/armv6/bcm2835/unix/Makefile
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible.  It claims to
supporte ELF file loading, but what it does is crazy.  It loads the ELF file
into memory, gets the start address from the header, converts it into file
offset, adds it to the address where the file was loaded and jumps there.
This is very wrong.  So, instead of booting the loader as an ELF file, we
objcopy it into a plain ol' binary image.  This should be safe because (1)
the loader has no relocations left, (2) whatever benefit we lose from not
having the whole ELF file in memory is only temporary until the loader hands
off control to unix.
Finally, we force the entry point to appear at the beginning of the binary
file by moving _start into its own section (.text.init) and using the
mapfile to put that section before everything else.
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../../..

#
#       Define the module and object file sets.
#
UNIX            = unix
OBJECTS         = $(BCM2835_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(KRTLD_OBJS:%=$(OBJS_DIR)/%)

ROOTMODULE      = $(ROOT_BCM2835_KERN_DIR)/$(UNIX)
UNIX_BIN        = $(OBJS_DIR)/$(UNIX)

LIBS            = $(GENLIB)

GENUNIX         = genunix
GENUNIX_DIR     = ../../../arm/$(GENUNIX)
GENOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)
```



```
LIBOPTS         = $(GENOPTS)

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o

ARCHIVE         = boot_archive

INITRD          = initrd
BOOT_MODULE     = $(ROOT_BCM2835_KERN_DIR)/$(INITRD)
INITRD_BIN      = $(OBJS_DIR)/$(INITRD)
#endif /* ! codereview */

#
#       Include common rules.
#
include $(UTSBASE)/armv6/bcm2835/Makefile.bcm2835

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE) $(BOOT_MODULE)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      += $(OBJECTS) \
                   $(UNIX_O)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)

CFLAGS += $(CCVERBOSE)

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-but-set-variable

# The mapfile used to link unix
MAPFILE         = $(UTSBASE)/armv6/bcm2835/conf/Mapfile

# Tools to build the platform and loader kernel.
MKPLATFORM      = $(UTSBASE)/$(PLATFORM)/tools/mkplatform
MKUNI           = $(UTSBASE)/$(PLATFORM)/tools/mkuni

#endif /* ! codereview */
#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)
```

```
116 install:        $(INSTALL_DEPS) $(MKPLATFORM) $(MKUNI)
 52 MKPLATFORM      = $(UTSBASE)/$(PLATFORM)/tools/mkplatform.sh

 54 install:        $(INSTALL_DEPS)
 55         pfexec $(MKPLATFORM) -u $(ROOT_BCM2835_MOD_DIR:$(ROOT)/%=/%) $(UNIX_BIN)
 56             $(GENUNIX_DIR)/$(OBJS_DIR)/genunix
 57         $(MV) boot_archive $(OBJS_DIR)/

118 symcheck:       $(SYM_DEPS)

120 $(UNIX_BIN): $(UNIX_O) $(MAPFILE) $(LIBS) $(DTRACESTUBS)
121         $(LD) -dy -b -znointerp -o $@ -e _start -M $(MAPFILE) \
122             $(UNIX_O) $(LIBOPTS) $(DTRACESTUBS)
123         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
124         $(POST_PROCESS)

126 $(UNIX_O):      $(OBJECTS) $(OBJS_DIR)/vers.o
127         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

129 symcheck.targ:  $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBS) $(DTRACESTUBS)
130         $(LD) -dy -b -o $(SYM_MOD) -M $(MAPFILE) \
131         $(UNIX_O) $(KRTLD_O) $(MODSTUBS_O) $(LIBOPTS) $(DTRACESTUBS)

133 $(KRTLD_O):     $(KRTLD_OBJECTS)
134         $(LD) -r -o $@ -M$(KRTLD_MAPFILE) $(KRTLD_OBJECTS)

136 $(MKPLATFORM) $(MKUNI):
137         cd ../../tools && pwd && dmake

139 $(INITRD_BIN): $(MKPLATFORM) $(MKUNI) $(UNIX_BIN) $(GENUNIX_DIR)/$(OBJS_DIR)/$(G
140         pfexec $(MKPLATFORM) -u $(ROOT_BCM2835_MOD_DIR:$(ROOT)/%=/%) $(UNIX_BIN)
141             $(GENUNIX_DIR)/$(OBJS_DIR)/$(GENUNIX)
142         $(MV) $(ARCHIVE) $(OBJS_DIR)/
143         $(MKUNI) $(UNIX_BIN) $(OBJS_DIR)/$(ARCHIVE) $(OBJS_DIR)/$(INITRD)
144 #endif /* ! codereview */

146 #
147 #       Include common targets.
148 #
149 include $(UTSBASE)/armv6/bcm2835/Makefile.targ
```

```
**********************************************************
   18576 Sun Jan 25 13:29:12 2015
new/usr/src/uts/armv6/loader/fakeloader.c
loader: pass args along to unix in C
There's no reason why we can't pass the args gotten from the bootloader to
unix in C.
Note: loader's _start sets the ATAG pointer to 0x100.  This change simply
propagates it to unix.
**********************************************************
```

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

/*
 * Copyright (c) 2014 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
 */

#include "fakeloader.h"

#include <sys/types.h>
#include <sys/param.h>
#include <sys/elf.h>
#include <sys/atag.h>
#include <sys/sysmacros.h>
#include <sys/machparam.h>

#include <vm/pte.h>

/*
 * This is the stock ARM fake uniboot loader.
 *
 * Here's what we have to do:
 *   o Read the atag header and find the combined archive header
 *   o Determine the set of mappings we need to add for the following:
 *             - unix
 *             - boot_archive
 *             - atags
 *   o Enable unaligned access
 *   o Enable the caches + virtual memory
 *
 * There are several important constraints that we have here:
 *
 *   o We cannot use any .data! Several loaders that come before us are broken
 *     and only provide us with the ability to map our .text and potentially our
 *     .bss. We should strive to avoid even that if we can.
 */

#ifdef  DEBUG
#define FAKELOAD_DPRINTF(x)     fakeload_puts(x)
#else
#define FAKELOAD_DPRINTF(x)
#endif  /* DEBUG */

/*
 * XXX ASSUMES WE HAVE Free memory following the boot archive
 */
static uintptr_t freemem;
```



```
static uintptr_t pt_arena;
static uintptr_t pt_arena_max;
static uint32_t *pt_addr;
static int nl2pages;

/* Simple copy routines */
void
bcopy(const void *s, void *d, size_t n)
{
        const char *src = s;
        char *dest = d;

        if (n == 0 || s == d)
                return;

        if (dest < src && dest + n < src) {
                /* dest overlaps with the start of src, copy forward */
                for (; n > 0; n--, src++, dest++)
                        *dest = *src;
        } else {
                /* src overlaps with start of dest or no overlap, copy rev */
                src += n - 1;
                dest += n - 1;
                for (; n > 0; n--, src--, dest--)
                        *dest = *src;
        }
}

void
bzero(void *s, size_t n)
{
        char *c = s;
        while (n > 0) {
                *c = 0;
                c++;
                n--;
        }
}

static void
fakeload_puts(const char *str)
{
        while (*str != '\0') {
                fakeload_backend_putc(*str);
                str++;
        }
}

static void
fakeload_panic(const char *reason)
{
        fakeload_puts("panic!\n");
        fakeload_puts(reason);
        fakeload_puts("\n");
        fakeload_puts("spinning forever... goodbye...\n");
        for (;;)
                ;
}

static void
fakeload_ultostr(unsigned long value)
{
        char buf[16];
        ulong_t t, val = (ulong_t)value;
        char c;
        char *ptr = &(buf[14]);
```

```
124         buf[15] = '\0';
125
126         do {
127                 c = (char)('0' + val - 16 * (t = (val >> 4)));
128                 if (c > '9')
129                         c += 'A' - '9' - 1;
130                 *--ptr = c;
131         } while ((val = t) != 0);
132
133         *--ptr = 'x';
134         *--ptr = '0';
135         fakeload_puts(ptr);
136 }
137
138 static void
139 fakeload_selfmap(atag_header_t *chain)
140 {
141         atag_illumos_mapping_t aim;
142
143         aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
144         aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
145         aim.aim_paddr = 0x7000;
146         aim.aim_vaddr = aim.aim_paddr;
147         aim.aim_plen = 0x3000;
148         aim.aim_vlen = aim.aim_plen;
149         aim.aim_mapflags = PF_R | PF_X | PF_LOADER;
150         atag_append(chain, &aim.aim_header);
151 }
152
153 static void
154 fakeload_map_1mb(uintptr_t pa, uintptr_t va, int prot)
155 {
156         int entry;
157         armpte_t *pte;
158         arm_l1s_t *l1e;
159
160         entry = ARMPT_VADDR_TO_L1E(va);
161         pte = &pt_addr[entry];
162         if (ARMPT_L1E_ISVALID(*pte))
163                 fakeload_panic("armboot_mmu: asked to map a mapped region!\n");
164         l1e = (arm_l1s_t *)pte;
165         *pte = 0;
166         l1e->al_type = ARMPT_L1_TYPE_SECT;
167         /* Assume it's not device memory */
168         l1e->al_bbit = 1;
169         l1e->al_cbit = 1;
170         l1e->al_tex = 1;
171         l1e->al_sbit = 1;
172
173         if (!(prot & PF_X))
174                 l1e->al_xn = 1;
175         l1e->al_domain = 0;
176
177         if (prot & PF_W) {
178                 l1e->al_ap2 = 1;
179                 l1e->al_ap = 1;
180         } else {
181                 l1e->al_ap2 = 0;
182                 l1e->al_ap = 1;
183         }
184         l1e->al_ngbit = 0;
185         l1e->al_issuper = 0;
186         l1e->al_addr = ARMPT_PADDR_TO_L1SECT(pa);
187 }
188
189 /*
```



```
190  * Set freemem to be 1 MB aligned at the end of boot archive. While the L1 Page
191  * table only needs to be 16 KB aligned, we opt for 1 MB alignment so that way
192  * we can map it and all the other L2 page tables we might need. If we don't do
193  * this, it'll become problematic for unix to actually modify this.
194  */
195 static void
196 fakeload_pt_arena_init(const atag_initrd_t *aii)
197 {
198         int entry, i;
199         armpte_t *pte;
200         arm_l1s_t *l1e;
201
202         pt_arena = aii->ai_start + aii->ai_size;
203         if (pt_arena & MMU_PAGEOFFSET1M) {
204                 pt_arena &= MMU_PAGEMASK1M;
205                 pt_arena += MMU_PAGESIZE1M;
206         }
207         pt_arena_max = pt_arena + 4 * MMU_PAGESIZE1M;
208         freemem = pt_arena_max;
209
210         /* Set up the l1 page table by first invalidating it */
211         pt_addr = (armpte_t *)pt_arena;
212         pt_arena += ARMPT_L1_SIZE;
213         bzero(pt_addr, ARMPT_L1_SIZE);
214         for (i = 0; i < 4; i++)
215                 fakeload_map_1mb((uintptr_t)pt_addr + i * MMU_PAGESIZE1M,
216                     (uintptr_t)pt_addr + i * MMU_PAGESIZE1M,
217                     PF_R | PF_W);
218 }
219
220 /*
221  * This is our generally entry point. We're passed in the entry point of the
222  * header.
223  */
224 static uintptr_t
225 fakeload_archive_mappings(atag_header_t *chain, const void *addr,
226     atag_illumos_status_t *aisp)
227 {
228         atag_illumos_mapping_t aim;
229         fakeloader_hdr_t *hdr;
230         Elf32_Ehdr *ehdr;
231         Elf32_Phdr *phdr;
232         int nhdrs, i;
233         uintptr_t ret;
234         uintptr_t text = 0, data = 0;
235         size_t textln = 0, dataln = 0;
236
237         hdr = (fakeloader_hdr_t *)addr;
238
239         if (hdr->fh_magic[0] != FH_MAGIC0)
240                 fakeload_panic("fh_magic[0] is wrong!\n");
241         if (hdr->fh_magic[1] != FH_MAGIC1)
242                 fakeload_panic("fh_magic[1] is wrong!\n");
243         if (hdr->fh_magic[2] != FH_MAGIC2)
244                 fakeload_panic("fh_magic[2] is wrong!\n");
245         if (hdr->fh_magic[3] != FH_MAGIC3)
246                 fakeload_panic("fh_magic[3] is wrong!\n");
247
248         if (hdr->fh_unix_size == 0)
249                 fakeload_panic("hdr unix size is zero\n");
250         if (hdr->fh_unix_offset == 0)
251                 fakeload_panic("hdr unix offset is zero\n");
252         if (hdr->fh_archive_size == 0)
253                 fakeload_panic("hdr archive size is zero\n");
254         if (hdr->fh_archive_offset == 0)
255                 fakeload_panic("hdr archive_offset is zero\n");
```

```
257         ehdr = (Elf32_Ehdr *)((uintptr_t)addr + hdr->fh_unix_offset);

259         if (ehdr->e_ident[EI_MAG0] != ELFMAG0)
260                 fakeload_panic("magic[0] wrong");
261         if (ehdr->e_ident[EI_MAG1] != ELFMAG1)
262                 fakeload_panic("magic[1] wrong");
263         if (ehdr->e_ident[EI_MAG2] != ELFMAG2)
264                 fakeload_panic("magic[2] wrong");
265         if (ehdr->e_ident[EI_MAG3] != ELFMAG3)
266                 fakeload_panic("magic[3] wrong");
267         if (ehdr->e_ident[EI_CLASS] != ELFCLASS32)
268                 fakeload_panic("wrong elfclass");
269         if (ehdr->e_ident[EI_DATA] != ELFDATA2LSB)
270                 fakeload_panic("wrong encoding");
271         if (ehdr->e_ident[EI_OSABI] != ELFOSABI_SOLARIS)
272                 fakeload_panic("wrong os abi");
273         if (ehdr->e_ident[EI_ABIVERSION] != EAV_SUNW_CURRENT)
274                 fakeload_panic("wrong abi version");
275         if (ehdr->e_type != ET_EXEC)
276                 fakeload_panic("unix is not an executable");
277         if (ehdr->e_machine != EM_ARM)
278                 fakeload_panic("unix is not an ARM Executible");
279         if (ehdr->e_version != EV_CURRENT)
280                 fakeload_panic("wrong version");
281         if (ehdr->e_phnum == 0)
282                 fakeload_panic("no program headers");
283         ret = ehdr->e_entry;

285         FAKELOAD_DPRINTF("validated unix's headers\n");

287         nhdrs = ehdr->e_phnum;
288         phdr = (Elf32_Phdr *)((uintptr_t)addr + hdr->fh_unix_offset +
289             ehdr->e_phoff);
290         for (i = 0; i < nhdrs; i++, phdr++) {
291                 if (phdr->p_type != PT_LOAD) {
292                         fakeload_puts("skipping non-PT_LOAD header\n");
293                         continue;
294                 }

296                 if (phdr->p_filesz == 0 || phdr->p_memsz == 0) {
297                         fakeload_puts("skipping PT_LOAD with 0 file/mem\n");
298                         continue;
299                 }

301                 /*
302                  * Create a mapping record for this in the atags.
303                  */
304                 aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
305                 aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
306                 aim.aim_paddr = (uintptr_t)addr + hdr->fh_unix_offset +
307                     phdr->p_offset;
308                 aim.aim_plen = phdr->p_filesz;
309                 aim.aim_vaddr = phdr->p_vaddr;
310                 aim.aim_vlen = phdr->p_memsz;
311                 /* Round up vlen to be a multiple of 4k */
312                 if (aim.aim_vlen & 0xfff) {
313                         aim.aim_vlen &= ~0xfff;
314                         aim.aim_vlen += 0x1000;
315                 }
316                 aim.aim_mapflags = phdr->p_flags;
317                 atag_append(chain, &aim.aim_header);

319                 /*
320                  * When built with highvecs we need to account for the fact that
321                  * _edata, _etext and _end are built assuming that the highvecs
```



```
322                  * are normally part of our segments. ld is not doing anything
323                  * wrong, but this breaks the assumptions that krtld currently
324                  * has. As such, unix will use this information to overwrite the
325                  * normal entry points that krtld uses in a similar style to
326                  * SPARC.
327                  */
328                 if (aim.aim_vaddr != 0xffff0000) {
329                         if ((phdr->p_flags & PF_W) != 0) {
330                                 data = aim.aim_vaddr;
331                                 dataln = aim.aim_vlen;
332                         } else {
333                                 text = aim.aim_vaddr;
334                                 textln = aim.aim_vlen;
335                         }
336                 }
337         }

339         aisp->ais_stext = text;
340         aisp->ais_etext = text + textln;
341         aisp->ais_sdata = data;
342         aisp->ais_edata = data + dataln;

344         /* 1:1 map the boot archive */
345         aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
346         aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
347         aim.aim_paddr = (uintptr_t)addr + hdr->fh_archive_offset;
348         aim.aim_plen = hdr->fh_archive_size;
349         aim.aim_vaddr = aim.aim_paddr;
350         aim.aim_vlen = aim.aim_plen;
351         aim.aim_mapflags = PF_R | PF_W | PF_X;
352         atag_append(chain, &aim.aim_header);
353         aisp->ais_archive = aim.aim_paddr;
354         aisp->ais_archivelen = aim.aim_plen;

356         return (ret);
357 }

359 static void
360 fakeload_mkatags(atag_header_t *chain)
361 {
362         atag_illumos_status_t ais;
363         atag_illumos_mapping_t aim;

365         bzero(&ais, sizeof (ais));
366         bzero(&aim, sizeof (aim));

368         ais.ais_header.ah_size = ATAG_ILLUMOS_STATUS_SIZE;
369         ais.ais_header.ah_tag = ATAG_ILLUMOS_STATUS;
370         atag_append(chain, &ais.ais_header);
371         aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
372         aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
373         atag_append(chain, &aim.aim_header);
374 }

376 static uintptr_t
377 fakeload_alloc_l2pt(void)
378 {
379         uintptr_t ret;

381         if (pt_arena & ARMPT_L2_MASK) {
382                 ret = pt_arena;
383                 ret &= ~ARMPT_L2_MASK;
384                 ret += ARMPT_L2_SIZE;
385                 pt_arena = ret + ARMPT_L2_SIZE;
386         } else {
387                 ret = pt_arena;
```

```
388                 pt_arena = ret + ARMPT_L2_SIZE;
389         }
390         if (pt_arena >= pt_arena_max) {
391                 fakeload_puts("pt_arena, max\n");
392                 fakeload_ultostr(pt_arena);
393                 fakeload_puts("\n");
394                 fakeload_ultostr(pt_arena_max);
395                 fakeload_puts("\n");
396                 fakeload_puts("l2pts alloced\n");
397                 fakeload_ultostr(nl2pages);
398                 fakeload_puts("\n");
399                 fakeload_panic("ran out of page tables!");
400         }

402         bzero((void *)ret, ARMPT_L2_SIZE);
403         nl2pages++;
404         return (ret);
405 }

407 /*
408  * Finally, do all the dirty work. Let's create some page tables. The L1 page
409  * table is full of 1 MB mappings by default. The L2 Page table is 1k in size
410  * and covers that 1 MB. We're going to always create L2 page tables for now
411  * which will use 4k and 64k pages.
412  */
413 static void
414 fakeload_map(armpte_t *pt, uintptr_t pstart, uintptr_t vstart, size_t len,
415     uint32_t prot)
416 {
417         int entry, chunksize;
418         armpte_t *pte, *l2pt;
419         arm_l1pt_t *l1pt;

421         /*
422          * Make sure both pstart + vstart are 4k aligned, along with len.
423          */
424         if (pstart & MMU_PAGEOFFSET)
425                 fakeload_panic("pstart is not 4k aligned");
426         if (vstart & MMU_PAGEOFFSET)
427                 fakeload_panic("vstart is not 4k aligned");
428         if (len & MMU_PAGEOFFSET)
429                 fakeload_panic("len is not 4k aligned");

431         /*
432          * We're going to logically deal with each 1 MB chunk at a time.
433          */
434         while (len > 0) {
435                 if (vstart & MMU_PAGEOFFSET1M) {
436                         chunksize = MIN(len, MMU_PAGESIZE1M -
437                             (vstart & MMU_PAGEOFFSET1M));
438                 } else {
439                         chunksize = MIN(len, MMU_PAGESIZE1M);
440                 }

442                 entry = ARMPT_VADDR_TO_L1E(vstart);
443                 pte = &pt[entry];

445                 if (!ARMPT_L1E_ISVALID(*pte)) {
446                         uintptr_t l2table;

448                         if (!(vstart & MMU_PAGEOFFSET1M) &&
449                             !(pstart & MMU_PAGEOFFSET1M) &&
450                             len == MMU_PAGESIZE1M) {
451                                 fakeload_map_1mb(pstart, vstart, prot);
452                                 vstart += MMU_PAGESIZE1M;
453                                 pstart += MMU_PAGESIZE1M;
```



```
454                                 len -= MMU_PAGESIZE1M;
455                                 continue;
456                         }

458                         l2table = fakeload_alloc_l2pt();
459                         *pte = 0;
460                         l1pt = (arm_l1pt_t *)pte;
461                         l1pt->al_type = ARMPT_L1_TYPE_L2PT;
462                         l1pt->al_ptaddr = ARMPT_ADDR_TO_L1PTADDR(l2table);
463                 } else if ((*pte & ARMPT_L1_TYPE_MASK) != ARMPT_L1_TYPE_L2PT) {
464                         fakeload_panic("encountered l1 entry that's not a "
465                             "pointer to a level 2 table\n");
466                 } else {
467                         l1pt = (arm_l1pt_t *)pte;
468                 }

470                 /* Now that we have the l1pt fill in l2 entries */
471                 l2pt = (void *)(l1pt->al_ptaddr << ARMPT_L1PT_TO_L2_SHIFT);
472                 len -= chunksize;
473                 while (chunksize > 0) {
474                         arm_l2e_t *l2pte;

476                         entry = ARMPT_VADDR_TO_L2E(vstart);
477                         pte = &l2pt[entry];

479 #ifdef  MAP_DEBUG
480                         fakeload_puts("4k page pa->va, l2root, entry\n");
481                         fakeload_ultostr(pstart);
482                         fakeload_puts("->");
483                         fakeload_ultostr(vstart);
484                         fakeload_puts(", ");
485                         fakeload_ultostr((uintptr_t)l2pt);
486                         fakeload_puts(", ");
487                         fakeload_ultostr(entry);
488                         fakeload_puts("\n");
489 #endif

491                         if ((*pte & ARMPT_L2_TYPE_MASK) !=
492                             ARMPT_L2_TYPE_INVALID)
493                                 fakeload_panic("found existing l2 page table, "
494                                     "overlap in requested mappings detected!");
495                         /* Map vaddr to our paddr! */
496                         l2pte = ((arm_l2e_t *)pte);
497                         *pte = 0;
498                         if (!(prot & PF_X))
499                                 l2pte->ale_xn = 1;
500                         l2pte->ale_ident = 1;
501                         if (prot & PF_DEVICE) {
502                                 l2pte->ale_bbit = 1;
503                                 l2pte->ale_cbit = 0;
504                                 l2pte->ale_tex = 0;
505                                 l2pte->ale_sbit = 1;
506                         } else {
507                                 l2pte->ale_bbit = 1;
508                                 l2pte->ale_cbit = 1;
509                                 l2pte->ale_tex = 1;
510                                 l2pte->ale_sbit = 1;
511                         }
512                         if (prot & PF_W) {
513                                 l2pte->ale_ap2 = 1;
514                                 l2pte->ale_ap = 1;
515                         } else {
516                                 l2pte->ale_ap2 = 0;
517                                 l2pte->ale_ap = 1;
518                         }
519                         l2pte->ale_ngbit = 0;
```

```
520                         l2pte->ale_addr = ARMPT_PADDR_TO_L2ADDR(pstart);

522                         chunksize -= MMU_PAGESIZE;
523                         vstart += MMU_PAGESIZE;
524                         pstart += MMU_PAGESIZE;
525                 }
526         }
527 }

529 static void
530 fakeload_create_map(armpte_t *pt, atag_illumos_mapping_t *aimp)
531 {
532 #ifdef MAP_DEBUG
533         fakeload_puts("paddr->vaddr\n");
534         fakeload_ultostr(aimp->aim_paddr);
535         fakeload_puts("->");
536         fakeload_ultostr(aimp->aim_vaddr);
537         fakeload_puts("\n");
538         fakeload_puts("plen-vlen\n");
539         fakeload_ultostr(aimp->aim_plen);
540         fakeload_puts("-");
541         fakeload_ultostr(aimp->aim_vlen);
542         fakeload_puts("\n");
543 #endif /* MAP_DEBUG */

545         /*
546          * Can we map this in place or do we need to basically allocate a new
547          * region and bcopy everything into place for proper alignment?
548          *
549          * Criteria for this: we have a vlen > plen. plen is not page aligned.
550          */
551         if (aimp->aim_vlen > aimp->aim_plen ||
552             (aimp->aim_paddr & MMU_PAGEOFFSET) != 0) {
553                 uintptr_t start;

555                 if (aimp->aim_mapflags & PF_NORELOC)
556                         fakeload_panic("tried to reloc unrelocatable mapping");
557 #ifdef  MAP_DEBUG
558                 FAKELOAD_DPRINTF("reloacting paddr\n");
559 #endif
560                 start = freemem;
561                 if (start & MMU_PAGEOFFSET) {
562                         start &= MMU_PAGEMASK;
563                         start += MMU_PAGESIZE;
564                 }
565                 bcopy((void *)aimp->aim_paddr, (void *)start,
566                     aimp->aim_plen);
567                 if (aimp->aim_vlen > aimp->aim_plen) {
568                         bzero((void *)(start + aimp->aim_plen),
569                             aimp->aim_vlen - aimp->aim_plen);
570                 }
571                 aimp->aim_paddr = start;
572                 freemem = start + aimp->aim_vlen;
573 #ifdef MAP_DEBUG
574                 fakeload_puts("new paddr: ");
575                 fakeload_ultostr(start);
576                 fakeload_puts("\n");
577 #endif /* MAP_DEBUG */
578         }

580         /*
581          * Now that everything has been set up, go ahead and map the new region.
582          */
583         fakeload_map(pt, aimp->aim_paddr, aimp->aim_vaddr, aimp->aim_vlen,
584             aimp->aim_mapflags);
585 #ifdef MAP_DEBUG
```



```
586         FAKELOAD_DPRINTF("\n");
587 #endif /* MAP_DEBUG */
588 }

590 void
591 fakeload_init(void *ident, void *ident2, void *atag)
592 {
593         atag_header_t *hdr;
594         atag_header_t *chain = (atag_header_t *)atag;
595         const atag_initrd_t *initrd;
596         atag_illumos_status_t *aisp;
597         atag_illumos_mapping_t *aimp;
598         uintptr_t unix_start;

600         fakeload_backend_init();
601         fakeload_puts("Hello from the loader\n");
602         initrd = (atag_initrd_t *)atag_find(chain, ATAG_INITRD2);
603         if (initrd == NULL)
604                 fakeload_panic("missing the initial ramdisk\n");

606         /*
607          * Create the status atag header and the initial mapping record for the
608          * atags. We'll hold onto both of these.
609          */
610         fakeload_mkatags(chain);
611         aisp = (atag_illumos_status_t *)atag_find(chain, ATAG_ILLUMOS_STATUS);
612         if (aisp == NULL)
613                 fakeload_panic("can't find ATAG_ILLUMOS_STATUS");
614         aimp = (atag_illumos_mapping_t *)atag_find(chain, ATAG_ILLUMOS_MAPPING);
615         if (aimp == NULL)
616                 fakeload_panic("can't find ATAG_ILLUMOS_MAPPING");
617         FAKELOAD_DPRINTF("created proto atags\n");

619         fakeload_pt_arena_init(initrd);

621         fakeload_selfmap(chain);

623         /*
624          * Map the boot archive and all of unix
625          */
626         unix_start = fakeload_archive_mappings(chain,
627             (const void *)(uintptr_t)initrd->ai_start, aisp);
628         FAKELOAD_DPRINTF("filled out unix and the archive's mappings\n");

630         /*
631          * Fill in the atag mapping header for the atags themselves. 1:1 map it.
632          */
633         aimp->aim_paddr = (uintptr_t)chain & ~0xfff;
634         aimp->aim_plen = atag_length(chain) & ~0xfff;
635         aimp->aim_plen += 0x1000;
636         aimp->aim_vaddr = aimp->aim_paddr;
637         aimp->aim_vlen = aimp->aim_plen;
638         aimp->aim_mapflags = PF_R | PF_W | PF_NORELOC;

640         /*
641          * Let the backend add mappings
642          */
643         fakeload_backend_addmaps(chain);

645         /*
646          * Turn on unaligned access
647          */
648         FAKELOAD_DPRINTF("turning on unaligned access\n");
649         fakeload_unaligned_enable();
650         FAKELOAD_DPRINTF("successfully enabled unaligned access\n");
```

```
652         /*
653          * To turn on the MMU we need to do the following:
654          *  o Program all relevant CP15 registers
655          *  o Program 1st and 2nd level page tables
656          *  o Invalidate and Disable the I/D-cache
657          *  o Fill in the last bits of the ATAG_ILLUMOS_STATUS atag
658          *  o Turn on the MMU in SCTLR
659          *  o Jump to unix
660          */

662         /* Last bits of the atag */
663         aisp->ais_freemem = freemem;
664         aisp->ais_version = 1;
665         aisp->ais_ptbase = (uintptr_t)pt_addr;

667         /*
668          * Our initial page table is a series of 1 MB sections. While we really
669          * should map 4k pages, for the moment we're just going to map 1 MB
670          * regions, yay team!
671          */
672         hdr = chain;
673         FAKELOAD_DPRINTF("creating mappings\n");
674         while (hdr != NULL) {
675                 if (hdr->ah_tag == ATAG_ILLUMOS_MAPPING)
676                         fakeload_create_map(pt_addr,
677                             (atag_illumos_mapping_t *)hdr);
678                 hdr = atag_next(hdr);
679         }

681         /*
682          * Now that we've mapped everything, update the status atag.
683          */
684         aisp->ais_freeused = freemem - aisp->ais_freemem;
685         aisp->ais_pt_arena = pt_arena;
686         aisp->ais_pt_arena_max = pt_arena_max;

688         /* Cache disable */
689         FAKELOAD_DPRINTF("Flushing and disabling caches\n");
690         armv6_dcache_flush();
691         armv6_dcache_disable();
692         armv6_dcache_inval();
693         armv6_icache_disable();
694         armv6_icache_inval();

696         /* Program the page tables */
697         FAKELOAD_DPRINTF("programming cp15 regs\n");
698         fakeload_pt_setup((uintptr_t)pt_addr);


701         /* MMU Enable */
702         FAKELOAD_DPRINTF("see you on the other side\n");
703         fakeload_mmu_enable();

705         FAKELOAD_DPRINTF("why helo thar\n");

707         /* Renable caches */
708         armv6_dcache_enable();
709         armv6_icache_enable();

711         /* we should never come back */
712         fakeload_exec(ident, ident2, chain, unix_start);
 14         fakeload_exec(unix_start);
713         fakeload_panic("hit the end of the world\n");
714 }
```

*unchanged_portion_omitted_*

****
1939 Sun Jan 25 13:29:12 2015
new/usr/src/uts/armv6/loader/fakeloader.h
loader: pass args along to unix in C
There's no reason why we can't pass the args gotten from the bootloader to unix in C.
Note: loader's _start sets the ATAG pointer to 0x100. This change simply propagates it to unix.
****

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

/*
 * Copyright (c) 2013 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
#endif /* ! codereview */
 */

#ifndef _FAKELOADER_H
#define _FAKELOADER_H

/*
 * The hacky version of arm uniboot that is exactly for a few systems.
 */

#include <sys/stdint.h>
#include <sys/atag.h>

#ifdef __cplusplus
extern "C" {
#endif

typedef struct fakeloader_hdr {
        unsigned char fh_magic[4];      /* Magic! */
        uint32_t fh_unix_size;          /* How large is unix */
        uint32_t fh_unix_offset;        /* Offset from start to unix */
        uint32_t fh_archive_size;       /* How large is the archive */
        uint32_t fh_archive_offset;     /* Offset from start to archive */
} fakeloader_hdr_t;

#define FH_MAGIC0       'i'
#define FH_MAGIC1       'f'
#define FH_MAGIC2       'b'
#define FH_MAGIC3       'h'

/*
 * Backend operations, eg. what a given board must implement at the moment
 */
extern void fakeload_backend_init(void);
extern void fakeload_backend_putc(int);
extern void fakeload_backend_addmaps(atag_header_t *);

/*
 * ASM operations
 */
extern void fakeload_unaligned_enable(void);
extern void fakeload_mmu_enable(void);
extern void fakeload_pt_setup(uintptr_t);
```



```
extern void fakeload_exec(void *, void *, atag_header_t *, uintptr_t);
extern void fakeload_exec(uintptr_t);

extern void armv6_dcache_disable(void);
extern void armv6_dcache_enable(void);
extern void armv6_dcache_inval(void);
extern void armv6_dcache_flush(void);

extern void armv6_icache_disable(void);
extern void armv6_icache_enable(void);
extern void armv6_icache_inval(void);

#ifdef __cplusplus
}
```

*unchanged_portion_omitted_*

****
2350 Sun Jan 25 13:29:12 2015
new/usr/src/uts/armv6/loader/fakeloader_core.s
loader: pass args along to unix in C
There's no reason why we can't pass the args gotten from the bootloader to unix in C.
Note: loader's _start sets the ATAG pointer to 0x100. This change simply propagates it to unix.
bcm2835: we need a loader on this platform as well
As stated before, the Raspberry Pi loader is terrible. It claims to supporte ELF file loading, but what it does is crazy. It loads the ELF file into memory, gets the start address from the header, converts it into file offset, adds it to the address where the file was loaded and jumps there. This is very wrong. So, instead of booting the loader as an ELF file, we objcopy it into a plain ol' binary image. This should be safe because (1) the loader has no relocations left, (2) whatever benefit we lose from not having the whole ELF file in memory is only temporary until the loader hands off control to unix.
Finally, we force the entry point to appear at the beginning of the binary file by moving _start into its own section (.text.init) and using the mapfile to put that section before everything else.
****

```
  1 /*
  2  * This file and its contents are supplied under the terms of the
  3  * Common Development and Distribution License ("CDDL"), version 1.0.
  4  * You may only use this file in accordance with the terms of version
  5  * 1.0 of the CDDL.
  6  *
  7  * A full copy of the text of the CDDL should have accompanied this
  8  * source.  A copy of the CDDL is also available via the Internet at
  9  * http://www.illumos.org/license/CDDL.
 10  */

 12 /*
 13  * Copyright 2013 (c) Joyent, Inc. All rights reserved.
 14  * Copyright 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 15 #endif /* ! codereview */
 16  */

 18 /*
 19  * Every story needs a beginning, this is the loader's.
 20  */

 22 #include <sys/asm_linkage.h>

 24 /*
 25  * We put _start into the .text.init section so we can more easily shove it
 26  * at the front of the .text.
 27  */
 28         .section .text.init
 29         .align  4
 30         .globl  _start
 31         .type   _start, %function
 32 _start:
 14         ENTRY(_start)
 33         mov     sp, #0x8000
 34         /*
 35          * XXX manually fix up the tag start
 36          */
 37         mov     r2, #0x100
 38         bl      fakeload_init
 39         SET_SIZE(_start)
______unchanged_portion_omitted_
111 #endif  /* __lint */
```



```
114         ENTRY(fakeload_exec)
115         blx     r3
 97         mov     r4, r0
 98         mov     r2, #0x100
 99         blx     r4
116         /* We should never execute this. If we do we'll go back to a panic */
117         bx      lr
118         SET_SIZE(fakeload_exec)
______unchanged_portion_omitted_
```

```
**********************************************************
   24751 Sun Jan 25 13:29:12 2015
new/usr/src/uts/armv6/os/fakebop.c
fakebop: use a memlist to keep track of physical memory
**********************************************************
```

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

/*
 * Copyright (c) 2014 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 */

/*
 * Just like in i86pc, we too get the joys of mimicking the SPARC boot system.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/bootconf.h>
#include <sys/bootsvcs.h>
#include <sys/boot_console.h>
#include <sys/atag.h>
#include <sys/varargs.h>
#include <sys/cmn_err.h>
#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/ctype.h>
#include <sys/bootstat.h>
#include <sys/privregs.h>
#include <sys/cpu_asm.h>
#include <sys/boot_mmu.h>
#include <sys/elf.h>

static bootops_t bootop;

/*
 * Debugging help
 */
static int fakebop_prop_debug = 0;
static int fakebop_alloc_debug = 0;
static int fakebop_atag_debug = 0;

static uint_t kbm_debug = 1;
#define DBG_MSG(x)      { if (kbm_debug) bcons_puts(x); bcons_puts("\n"); }
#define BUFFERSIZE      256
static char buffer[BUFFERSIZE];

/*
 * fakebop memory allocations scheme
 *
 * It's a virtual world out there. The loader thankfully tells us all the areas
 * that it has mapped for us and it also tells us about the page table arena --
 * a set of addresses that have already been set aside for us. We have two
 * different kinds of allocations to worry about:
 *
 *    o Those that specify a particular vaddr
 *    o Those that do not specify a particular vaddr
```



```
 *
 * Those that do not specify a particular vaddr will come out of our scratch
 * space which is a fixed size arena of 16 MB (FAKEBOP_ALLOC_SIZE) that we set
 * aside at the beginning of the allocator. If we end up running out of that
 * then we'll go ahead and figure out a slightly larger area to worry about.
 *
 * Now, for those that do specify a particular vaddr we'll allocate more
 * physical address space for it. The loader set aside enough L2 page tables for
 * us that we'll go ahead and use the next 4k aligned address.
 */
#define FAKEBOP_ALLOC_SIZE      (16 * 1024 * 1024)

static size_t bop_alloc_scratch_size;
static uintptr_t bop_alloc_scratch_next;        /* Next scratch address */
static uintptr_t bop_alloc_scratch_last;        /* Last scratch address */

static uintptr_t bop_alloc_pnext;               /* Next paddr */
static uintptr_t bop_alloc_plast;               /* cross this paddr and panic */

#define BI_HAS_RAMDISK  0x1

/*
 * TODO Generalize this
 * This is the set of information tha we want to gather from the various atag
 * headers. This is simple and naive and will need to evolve as we have
 * additional boards beyond just the RPi.
 */
typedef struct bootinfo {
        uint_t          bi_flags;
        uint32_t        bi_memsize;
        uint32_t        bi_memstart;
        char            *bi_cmdline;
        uint32_t        bi_ramdisk;
        uint32_t        bi_ramsize;
} bootinfo_t;
```

*unchanged_portion_omitted_*

```
static void
fakebop_getatags(void *tagstart)
{
        atag_mem_t *amp;
        atag_cmdline_t *alp;
        atag_header_t *ahp = tagstart;
        atag_illumos_status_t *aisp;
        bootinfo_t *bp = &bootinfo;
        boolean_t got_mem = B_FALSE;

        bp->bi_flags = 0;
        while (ahp != NULL) {
                switch (ahp->ah_tag) {
                case ATAG_MEM:
                        amp = (atag_mem_t *)ahp;
                        /*
                         * We may actually get more than one ATAG_MEM if the
                         * system has discontiguous physical memory
                         */
                        if (got_mem) {
                                bop_printf(NULL, "found multiple ATAG_MEM\n");
                                bop_printf(NULL, "ignoring: %#x - %#x\n",
                                    amp->am_start, amp->am_start +
                                    amp->am_size - 1);
                                break;
                        }

                        bootop.boot_mem.physinstalled.ml_address = amp->am_start
                        bootop.boot_mem.physinstalled.ml_size = amp->am_size;
```

```
                        bootop.boot_mem.physinstalled.ml_prev = NULL;
                        bootop.boot_mem.physinstalled.ml_next = NULL;
                        bp->bi_memsize = amp->am_size;
                        bp->bi_memstart = amp->am_start;
                        got_mem = B_TRUE;
                        break;
                case ATAG_CMDLINE:
                        alp = (atag_cmdline_t *)ahp;
                        bp->bi_cmdline = alp->al_cmdline;
                        break;
                case ATAG_ILLUMOS_STATUS:
                        aisp = (atag_illumos_status_t *)ahp;
                        bp->bi_ramdisk = aisp->ais_archive;
                        bp->bi_ramsize = aisp->ais_archivelen;
                        bp->bi_flags |= BI_HAS_RAMDISK;
                        break;
                default:
                        break;
                }
                ahp = atag_next(ahp);
        }
}
```

*unchanged_portion_omitted_*