说亦不清，道亦不明

bluelan.blog.chinaunix.net

时间不等人啊...

zj47596731

博客访问： 179217
博文数量： 63
博客积分： 1187
博客等级： 少尉
技术积分： 591
用 户 组： 普通用户
注册时间： 2010-03-05 16:53

arm-linux的交叉编译环境的建立

2011-03-09 15:13:44

分类： 嵌入式

一、交叉编译环境介绍

交叉编译是嵌入式开发过程中的一项重要技术，其主要特征是某机器中执行的程序代码不是在本机编译生成，而是由另一台机器编译生成，一般把前者称为目标机，后者称为主机。

采用交叉编译的主要原因在于，多数嵌入式目标系统不能提供足够的资源供编译过程使用，因而只好将编译工程转移到高性能的主机中进行，这就需要在强大的pc机上建立一个用于目标机的交叉编译环境。这是一个由编译器、连接器和解释器组成的综合开发环境。

linux下的交叉编译环境重要包括以下几个部分：

1，针对目标系统的编译器gcc；

2，针对目标系统的二进制工具binutils；

3，目标系统的标准c库glibc，有时出于减小libc 库大小的考虑，你也可以用别的c库来代替glibc，例如uClibc、newlib等；

4，目标系统的linux内核头文件。

二、准备工作

本文实验所使用的主机环境为cygwin，所以在编译交叉工具链的时候要注意cygwin版本的问题，建议将cygwin在线升级至最新版本，本文试验中cygwin DLL版本号为1.5.21。

因为gcc、binutils、glibc以及linux内核头文件均有各自的版本号，并不是任意组合都可以编译成功并最终建立一个交叉编译环境的。一些可以直接利用的组合方式，可以通过该网址查看：http://kegel.com/crosstool/ 当我们选择了某一种组合以后，仍然需要对源代码做相应的修改，才能最终编译成功。

本文使用的组合为gcc-3.4.5+glibc-2.2.5+binutils-2.15+linux-2.6.8(头文件)，因为本文建立的交叉编译环境是用来编译linux-2.6系列内核，以及运行在该系列内核上的程序，故选择了linux-2.6.8内核的头文件，当然也可以选择其他合适的linux-2.6版本的内核。

针对上面的这种组合，我们还需要打相应的补丁，这些补丁可以通过如下网址下载到：http://kegel.com/crosstool/crosstool-0.42.tar.gz 解压后在patch文件夹中可以找到相应的补丁。下面给出一些下载的链接，便于下载。

名称	源代码网址
gcc-3.4.5.tar.bz2	http://ftp.gnu.org/gnu/gcc/
binutils-2.15.tar.bz2	http://ftp.gnu.org/gnu/binutils/
glibc-2.2.5.tar.gz	http://ftp.gnu.org/gnu/glibc/
glibc-linuxthreads-2.2.5.tar.gz	
linux-2.6.8.tar.bz2	http://www.kernel.org/pub/linux/kernel/v2.6/

名称	补丁网址
gcc-3.4.5.tar.bz2	http://kegel.com/crosstool/crosstool-0.42/patches/gcc-3.4.5/
binutils-2.15.tar.bz2	http://kegel.com/crosstool/crosstool-0.42/patches/binutils-2.15/
glibc-2.2.5.tar.gz	http://kegel.com/crosstool/crosstool-0.42/patches/glibc-2.2.5/
glibc-linuxthreads-2.2.5.tar.gz	
linux-2.6.8.tar.bz2	http://kegel.com/crosstool/crosstool-0.42/patches/linux-2.6.8/

三、交叉编译环境的建立过程

为了更清晰的描述交叉环境的建立过程，在此对各个源码包的作用进行说明。

binutils-2.15.tar.bz2：这个压缩包包含有ld,ar,as等一些产生或者处理二进制文件的工具。其主要目的是为GNU系统提供汇编和连接工具等。

gcc-3.4.5.tar.bz2：这个压缩包主要是为GNU系统提供C 编译器。现在支持多种语言，这其中包括C/C++、Fortran、Java、Objective-C、Ada等。

glibc-2.2.5.tar.gz：Libc是很多用户层应用都要用到的库，用于定义系统调用和其它一些基本的函数调用。

glibc-linuxthreads-2.2.5.tar.gz：这是Libc用于支持Posix线程而单独发布的一个压缩包。

linux-2.6.8.tar.bz2：这个压缩包就是Linux的内核，在编译glibc时，要用到Linux内核中的include目录的内核头文件。

交叉编译环境建立过程如下，同时图3-1也清晰地描述了此过程。

（1）创建编译环境。在这个过程中，将设置一些环境变量，创建安装目录，安装内核源代码和头文件等。

（2）建立内核头文件，主要是生成include/linux/version.h 和 include/linux/autoconf.h 文件，这是编译glibc 是要用到的，version.h 和 autoconf.h 文件的存在，也说明了你生成了正确的头文件。

（3）创建binutils。这个过程结束后，会创建类似arm-linux-ld等工具。binutils是一组开发工具，包括链接器、汇编器以及其他用于目标文件和档案的工具.首先安装软件包binutils是非常重要的，因为glibc和gcc会针对可用的连接器和汇编器进行多种测试，以决定打开某些特性。

（4）创建一个交叉编译版本的gcc（称为bootstrap gcc）。注意：在这个过程中只能编译C程序，而不能编译C++程序。创建一个完整的交叉编译版本gcc，需要交叉编译版本的glibc及其头文件，而交叉编译版本的glibc是通过交叉编译版本的gcc创建的。面对这个先有鸡还是先有蛋的问题，解决办法是先编译仅支持C语言的bootstrap gcc编译器，并禁止支持线程。

（5）创建一个交叉编译版本的glibc。这里最容易出现问题。glibc是一个提供系统调用和基本函数的C语言库，比如open,malloc和printf等，所有动态链接的程序都要用到它。创建glibc需要的时间很长。

（6）重新创建gcc（称为full gcc）。因为前面创建gcc的过程没有编译C++编译器，现在glibc已经准备好了，所以这个步骤将产生一个更完整的full gcc编译器。

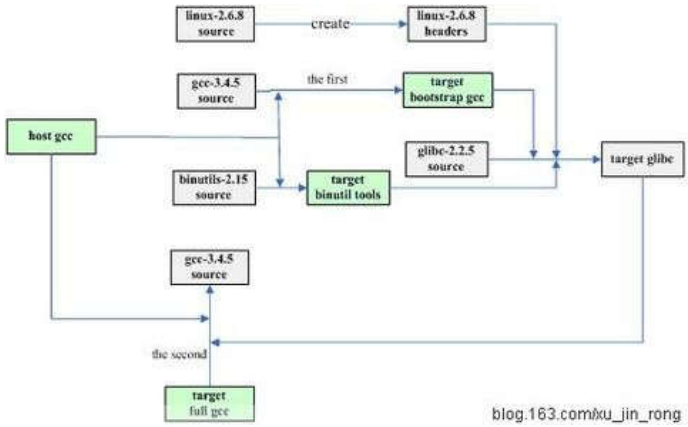

图3-1 交叉编译环境建立过程

四、交叉环境的建立 4．1工作目录及环境变量的设置

因为项目空间中目录众多,我们可以通过export命令设置一些环境变量以方便后面的工作，见表4-1。目录结构如图4-1所示。

环境变量	备注
TARGET=arm-linux	GNU 目标系统标识符
GCC_HOST=i686-pc-cygwin	主机系统标识符
BUILD=i686-pc-cygwin	构建系统标识符
SRC_DIR=/coretek/src_dir	源代码所放目录
BINUTILS_DIR=$ SRC_DIR/binutils-2.15	binutils 源代码目录名
GCC_DIR=$ SRC_DIR/gcc-3.4.5	gcc 源代码目录名
GLIBC_DIR=$ SRC_DIR/glibc-2.2.5	glibc 源代码目录名
LINUX_DIR=$ SRC_DIR/linux-2.6.8	linux 源代码目录名
PREFIX=/coretek/armtools	交叉工具安装目录
BUILD_DIR=/coretek/build_dir	交叉工具编译目录
CORE_PREFIX=$BUILD_DIR/gcc-core-prefix	bootstrap gcc 安装目录，不安装在 $PREFIX 中，以免影响 full gcc 的编译
SYSROOT=$PREFIX/STARGET	交叉工具链的系统根目录
HEADERDIR=$SYSROOT/include	交叉工具链编译时头文件所在目录

blog.163.com/xu_jin_rong

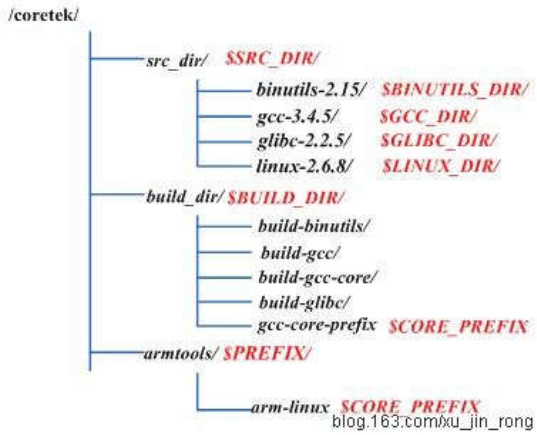

图4-1 目录结构图

4．2、建立内核头文件

因为交叉工具链工具链是针对特定的处理器和操作系统的，因此在编译之前就需要对linux内核进行配制，可以通过“make config”或“make menuconfig”命令对内核进行配制，配制完成后，在linux源文件的目录下就会生成一个.config文件，这就是我们所需要的文件。如果你有现成的配制文件，可以在“make menuconfig”中加载进来，或者使用cp命令把配制文件复制到linux源文件目录下并改名为.config。

此时的.config还不是完整的，因为有些信息在配置文件中没有给出，需要用户通过控制台输入，可以使用“make ARCH=arm oldconfig”命令，该命令可以使内核设置进程读取用户已有的设置信息，从而提示用户输入某一内核设置变量的值,这一变量在已有的内核设置文件中是找不到的,此处ARCH=arm就是我们输入的值。

接下来执行如下命令产生相关文件和链接：

make ARCH=$ARCH include/asm include/linux/version.h include/asm-$ARCH/.arch

执行完后，在linux2.6.8/include目录下生成version.h和autoconfig.h。这两个文件，在编译glibc时会用到。至此，linux的头文件已经生成完毕，现在通过如下命令，将编译交叉工具链时用到的头文件拷贝到$HEADERDIR目录下。图4-2为目录结构图。

cp -r include/asm-generic $HEADERDIR/asm-generic

cp -r include/linux $HEADERDIR

cp -r include/asm-${ARCH} $HEADERDIR/asm

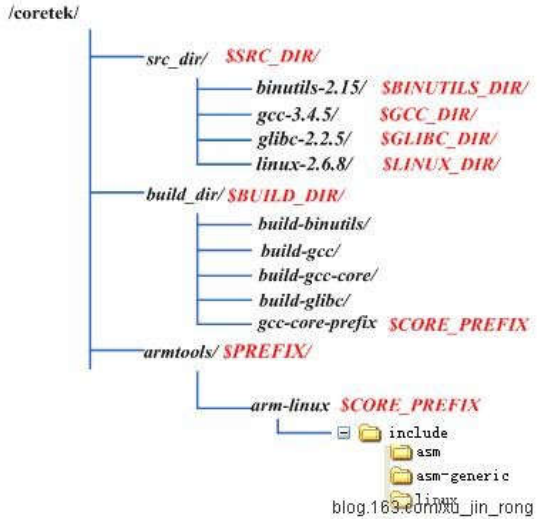

图4-2 linux内核头文件目录结构

4．3、建立二进制工具（binutils）

首先安装二进制工具链，使用主机的gcc进行编译。生成的交叉二进制工具arm-linux-ar,arm-linux-as,arm-linux-ld等是编译其他交叉程序的基础，所以必须放到第一步进行。编译过程如下：

```
cd $BUILD_DIR
mkdir -p build-binutils;
cd build-binutils
${BINUTILS_DIR}/configure --target=$TARGET --host=$GCC_HOST --prefix=$PREFIX --disable-nls –with-sysroot=$SYSROOT
make all
make install
export PATH="$PREFIX/bin:$PATH"
```

binutils工具生成以后，要将其路径加入环境变量PATH中，以便在后续编译过程中能够找到它们。生成的工具如图4-3所示。

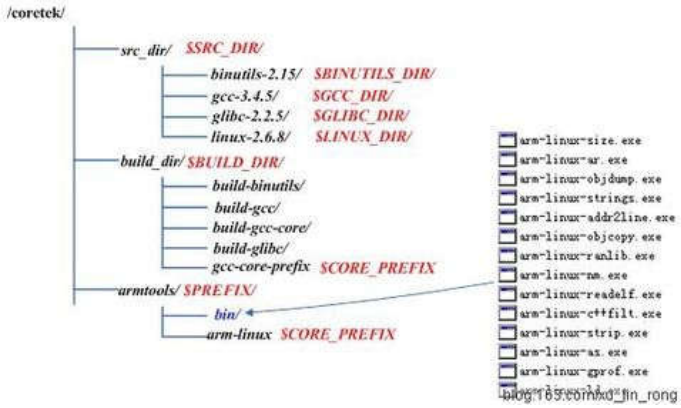

图4-3 生成的binutils工具

4．4、建立初始编译器（bootstrap gcc）

为了生成交叉编译版的glibc，我们就必须创建一个交叉编译版本的gcc。但是在资源有限的条件下，不可能拥有一个完整的交叉编译版的gcc（因为编译完整的gcc是需要交叉编译版的glibc及其头文件，而现在还没有）。我们现在只能够先利用主机的gcc编译出一个简单的交叉编译版gcc，即arm-linux-gcc及相关工具。arm-linux-gcc只能编译C程序，而不能编译C++程序。编译过程如下所示。

```
${GCC_CORE_DIR}/configure --target=$TARGET --host=$GCC_HOST \
    --prefix=$CORE_PREFIX \
  --with-local-prefix=$SYSROOT \
  --disable-multilib \
  --with-newlib \
```

```
    --disable-nls \
    --enable-threads=no \
    --enable-symvers=gnu \
    --enable-__cxa_atexit \
    --enable-languages=c \
    --disable-shared
make all-gcc
make install-gcc
export PATH="$CORE_PREFIX/bin:${PATH}"
```

bootstrap gcc生成以后，要将其路径加入环境变量PATH中，以便在后续编译过程中能够找到它们。生成的工具如图4-4所示。

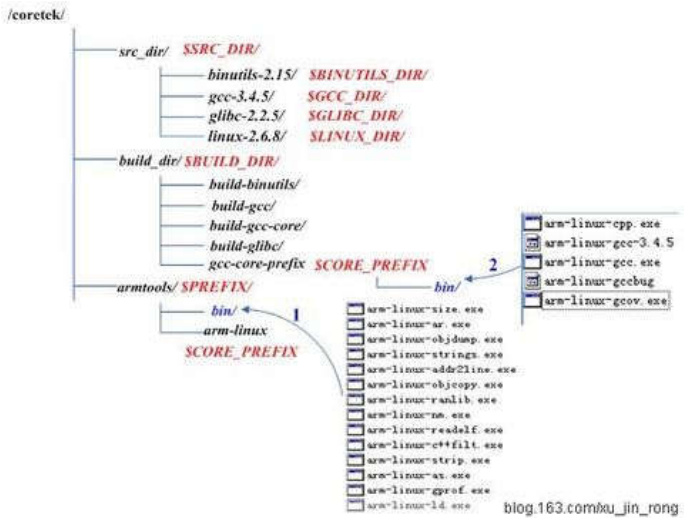

图4-4 生成bootstrap gcc后的目录结构图

4．5、建立c库(glibc)

glibc是一个提供系统调用和基本函数的C语言库，比如open,malloc和printf等，所有动态链接的程序都要用到它，这里最容易出现问题，并且创建glibc需要的时间很长。这时候编译器将会使用上一步生成的arm-linux-gcc，同时会用到一开始准备的linux内核头文件。编译glibc的命令如下所示。

```
BUILD_CC=gcc CFLAGS="-O -fno-unit-at-a-time" CC="${TARGET}-gcc
    AR=${TARGET}-ar RANLIB=${TARGET}-ranlib \
    ${GLIBC_DIR}/configure --prefix=/usr \
    --build=$BUILD --host=$TARGET \
    --without-cvs --disable-profile --disable-debug --without-gd \
    --enable-shared \
    --enable-add-ons=linuxthreads --with-headers=$HEADERDIR
make LD=${TARGET}-ld RANLIB=${TARGET}-ranlib all
make install_root=${SYSROOT} install
```

4．6、建立全套编译器（full gcc）

因为前面创建gcc的过程没有编译C++编译器，现在glibc已经准备好了，所以这个步骤将产生一个更完整的full gcc编译器,命令如下所示。

```
${GCC_DIR}/configure --target=$TARGET --host=$GCC_HOST --prefix=$PREFIX \
    --with-local-prefix=${SYSROOT} \
    --disable-nls \
    --enable-threads=posix \
    --enable-symvers=gnu \
    --enable-__cxa_atexit \
    --enable-languages= c,c++ \
    --enable-shared \
```

--enable-c99 \

--enable-long-long

make all

make install

至此，已经生成一个完整的交叉编译版gcc，此时生成的arm-linux-gcc等工具和4.3节生成的arm-linux-gcc并不相同，完整的交叉编译版的gcc被添加至$PREFIX/bin目录，其结构如图4-5所示。

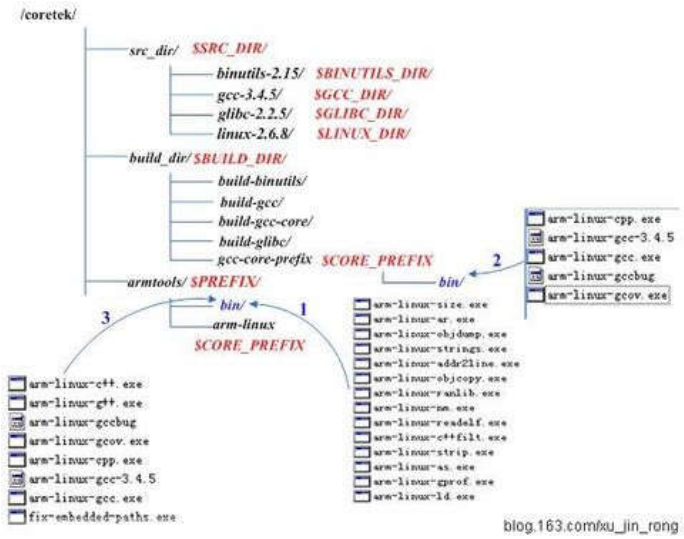

图4-5生成full gcc后的目录结构图

五、configure选项说明

完整的信息可以参考gcc-3.4.5\INSTALL\configure.html

--prefix=dirname

Specify the toplevel installation directory. This is the recommended way to install the tools into a directory other than the default. The toplevel installation directory defaults to /usr/local.

--with-local-prefix=dirname

Specify the installation directory for local include files. The default is /usr/local. Specify this option if you want the compiler to search directory dirname/include for locally installed header files instead of /usr/local/include.

You should specify --with-local-prefix **only** if your site has a different convention (not /usr/local) for where to put site-specific files.

The default value for --with-local-prefix is /usr/local regardless of the value of --prefix. Specifying --prefix has no effect on which directory GCC searches for local header files. This may seem counterintuitive, but actually it is logical.

The purpose of --prefix is to specify where to install GCC. The local header files in /usr/local/include—if you put any in that directory—are not part of GCC. They are part of other programs—perhaps many others. (GCC installs its own header files in another directory which is based on the --prefix value.)

Both the local-prefix include directory and the GCC-prefix include directory are part of GCC's "system include" directories. Although these two directories are not fixed, they need to be searched in the proper order for the correct processing of the include_next directive. The local-prefix include directory is searched before the GCC-prefix include directory. Another characteristic of system include directories is that pedantic warnings are turned off for headers in these directories.

Some autoconf macros add -I directory options to the compiler command line, to ensure that directories containing installed packages' headers are searched. When directory

is one of GCC's system include directories, GCC will ignore the option so that system directories continue to be processed in the correct order. This may result in a search order different from what was specified but the directory will still be searched.

GCC automatically searches for ordinary libraries using GCC_EXEC_PREFIX. Thus, when the same installation prefix is used for both GCC and packages, GCC will automatically search for both headers and libraries. This provides a configuration that is easy to use. GCC behaves in a manner similar to that when it is installed as a system compiler in /usr.

Sites that need to install multiple versions of GCC may not want to use the above simple configuration. It is possible to use the --program-prefix, --program-suffix and --program-transform-name options to install multiple versions into a single directory, but it may be simpler to use different prefixes and the --with-local-prefix option to specify the location of the site-specific files for each version. It will then be necessary for users to specify explicitly the location of local site libraries (e.g., with LIBRARY_PATH).

The same value can be used for both --with-local-prefix and --prefix provided it is not /usr. This can be used to avoid the default search of /usr/local/include.

--enable-shared[=package[,...]]

Build shared versions of libraries, if shared libraries are supported on the target platform. Unlike GCC 2.95.x and earlier, shared libraries are enabled by default on all platforms that support shared libraries, except for `libobjc' which is built as a static library only by default.

If a list of packages is given as an argument, build shared libraries only for the listed packages. For other packages, only static libraries will be built. Package names currently recognized in the GCC tree are `libgcc' (also known as `gcc'), `libstdc++' (not `libstdc++-v3'), `libffi', `zlib', `boehm-gc' and `libjava'. Note that `libobjc' does not recognize itself by any name, so, if you list package names in --enable-shared, you will only get static Objective-C libraries. `libf2c' and `libiberty' do not support shared libraries at all.

Use --disable-shared to build only static libraries. Note that --disable-shared does not accept a list of package names as argument, only --enable-shared does.

--with-sysroot

--with-sysroot=dir

Tells GCC to consider dir as the root of a tree that contains a (subset of) the root filesystem of the target operating system. Target system headers, libraries and run-time object files will be searched in there. The specified directory is not copied into the install tree, unlike the options --with-headers and --with-libs that this option obsoletes. The default value, in case --with-sysroot is not given an argument, is ${gcc_tooldir}/sys-root. If the specified directory is a subdirectory of ${exec_prefix}, then it will be found relative to the GCC binaries if the installation tree is moved.

--with-headers

--with-headers=dir

Deprecated in favor of --with-sysroot. Specifies that target headers are available when building a cross compiler. The dir argument specifies a directory which has the target include files. These include files will be copied into the gcc install directory. This option with the dir argument is required when building a cross compiler, if prefix/target/sys-include doesn't pre-exist. If prefix/target/sys-include does pre-exist, the dir argument may be omitted. fixincludes will be run on these files to make them compatible with GCC.

--without-headers

Tells GCC not use any target headers from a libc when building a cross compiler. When crossing to GNU/Linux, you need the headers so GCC can build the exception handling for libgcc. See CrossGCC for more information on this option.

--with-libs

--with-libs=``dir1 dir2 ... dirN''

Deprecated in favor of --with-sysroot. Specifies a list of directories which contain the target runtime libraries. These libraries will be copied into the gcc install directory. If the directory list is omitted, this option has no effect.

--with-newlib

Specifies that `newlib' is being used as the target C library. This causes __eprintf to be omitted from libgcc.a on the assumption that it will be provided by `newlib'.

上一篇：linux多线程pthread使用记录
下一篇：linux c libcurl的简单使用

0

相关热门文章

SHTML是什么_SSI有什么用... linux dhcp peizhi roc
shell中字符串操作 关于Unix文件的软链接
卡尔曼滤波的原理说明... 求教这个命令什么意思，我是新...
关于java中的“错误：找不到或... sed -e "/grep/d" 是什么意思...
linux设备驱动归纳总结... 谁能够帮我解决LINUX 2.6 10...

给主人留下些什么吧！~~

评论热议

请登录后评论。
登录 注册