嵌入式系统原理及应用

第三章 Linux交叉编译环境

本章内容

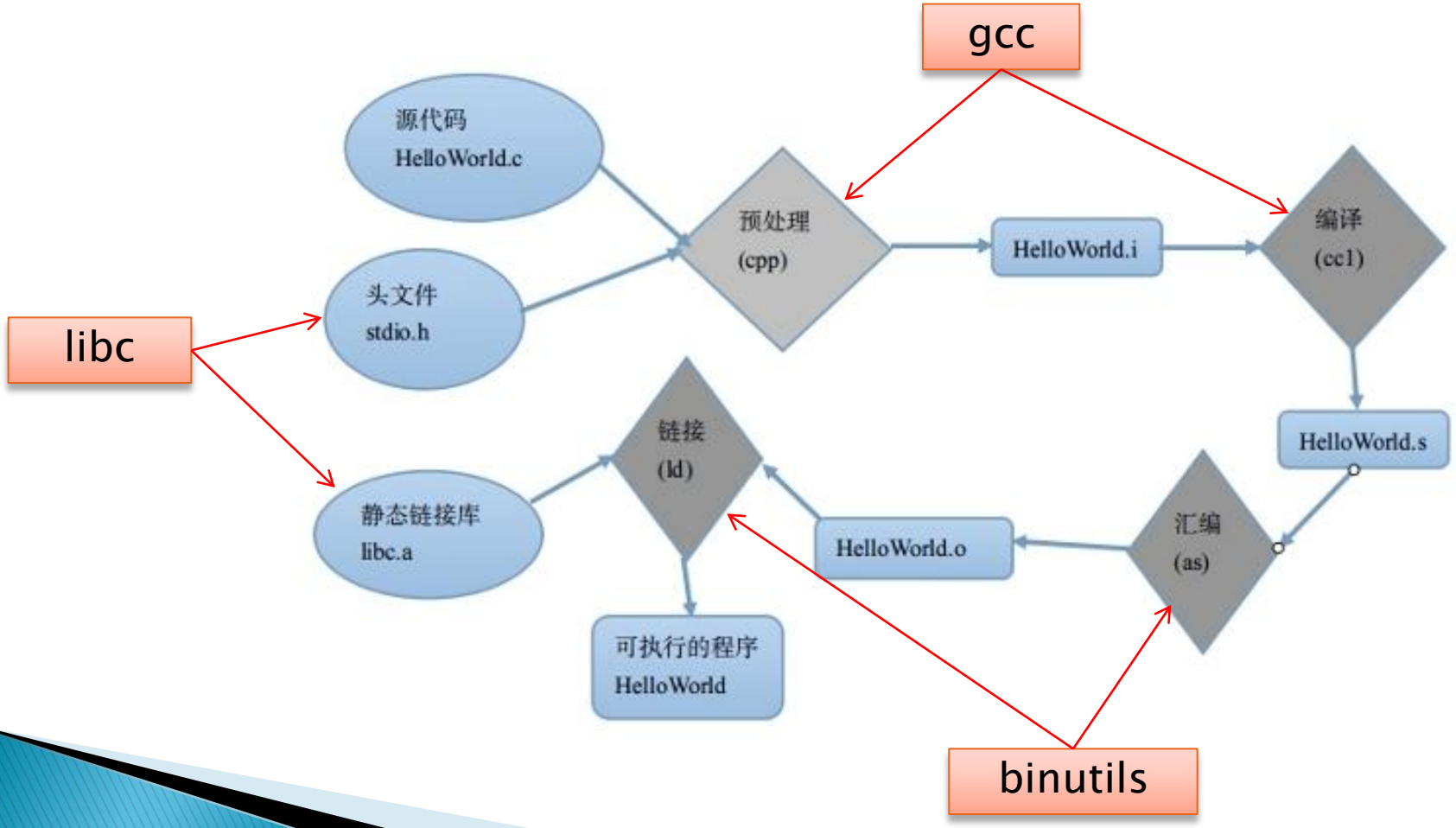
- GNU/Linux 编译工具链介绍
- 交叉编译原理
- GCC交叉编译工具链的构建过程

GNU/Linux编译工具链介绍

- 编译原理基础知识回顾
- GNU/Linux C编译环境和运行时环境
- GNU/Linux 编译工具的构成

编译原理基础知识回顾

- “Hello World!” 是如何产生的？



GNU/Linux C编译环境和运行时环境

- 编译环境
  - 应用程序在编译时需要从其包含的头文件中查找变量和各种接口的声明
  - 应用程序在编译时还需要链接到实现其调用功能的各种库
  - 如果编译驱动等调用到内核所提供的功能的程序还需要内核头文件
- 运行时环境
  - C语言运行时库 crt

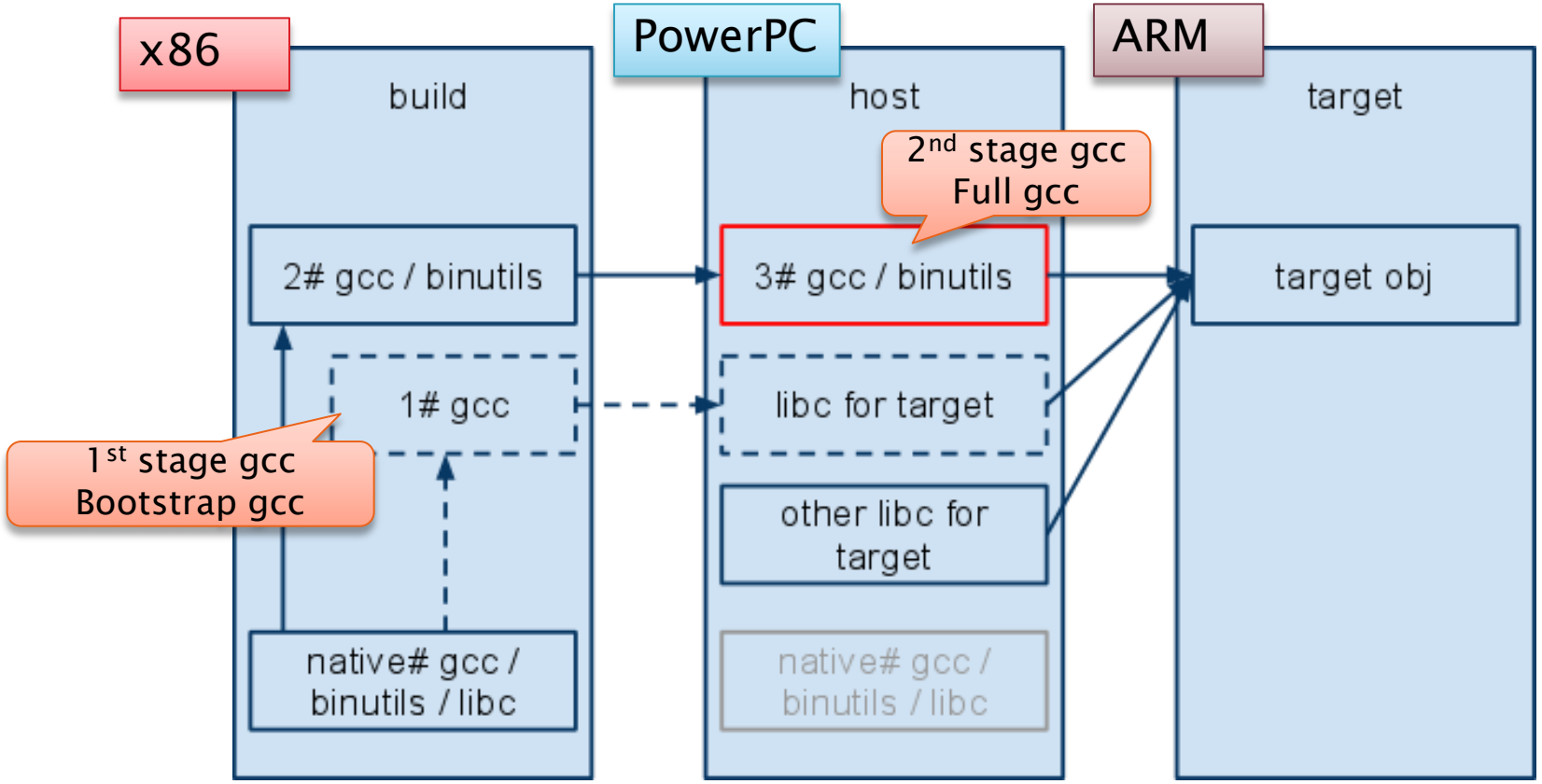
交叉编译原理

- 什么是交叉编译，为什么要进行交叉编译？
- 交叉编译中的关键概念：host, build和target
- 交叉编译中的依赖关系
- 交叉编译的完整过程
- 交叉编译器的验证

交叉编译的概念

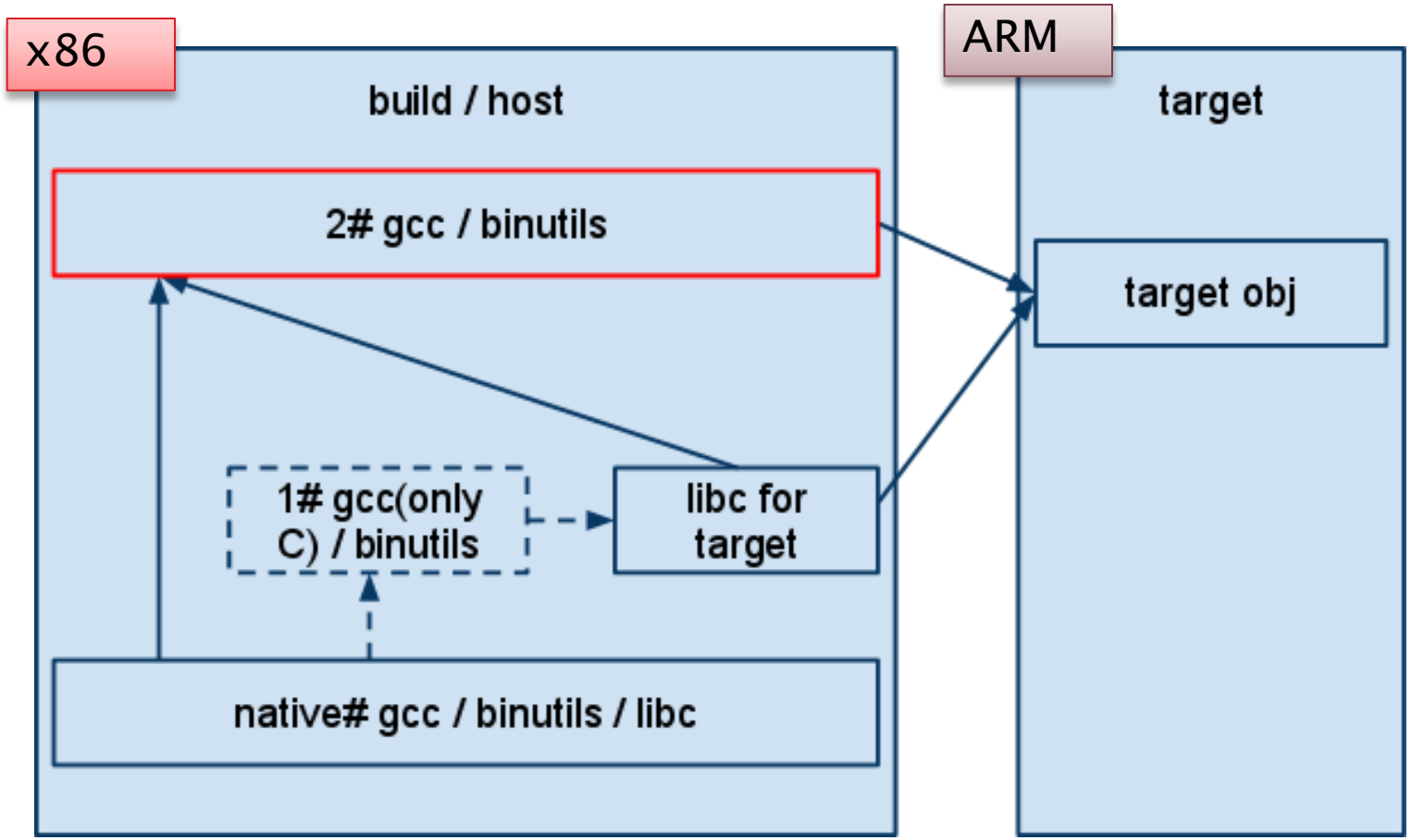
- 交叉编译（Cross Compile）:交叉编译是指在某个系统平台下产生另一个系统平台的可执行文件的编译过程。交叉编译在目标系统平台（开发出来的应用程序所运行的平台）难以或不容易编译目标程序时非常有用。

GNU/Linux GCC工具链的交叉编译

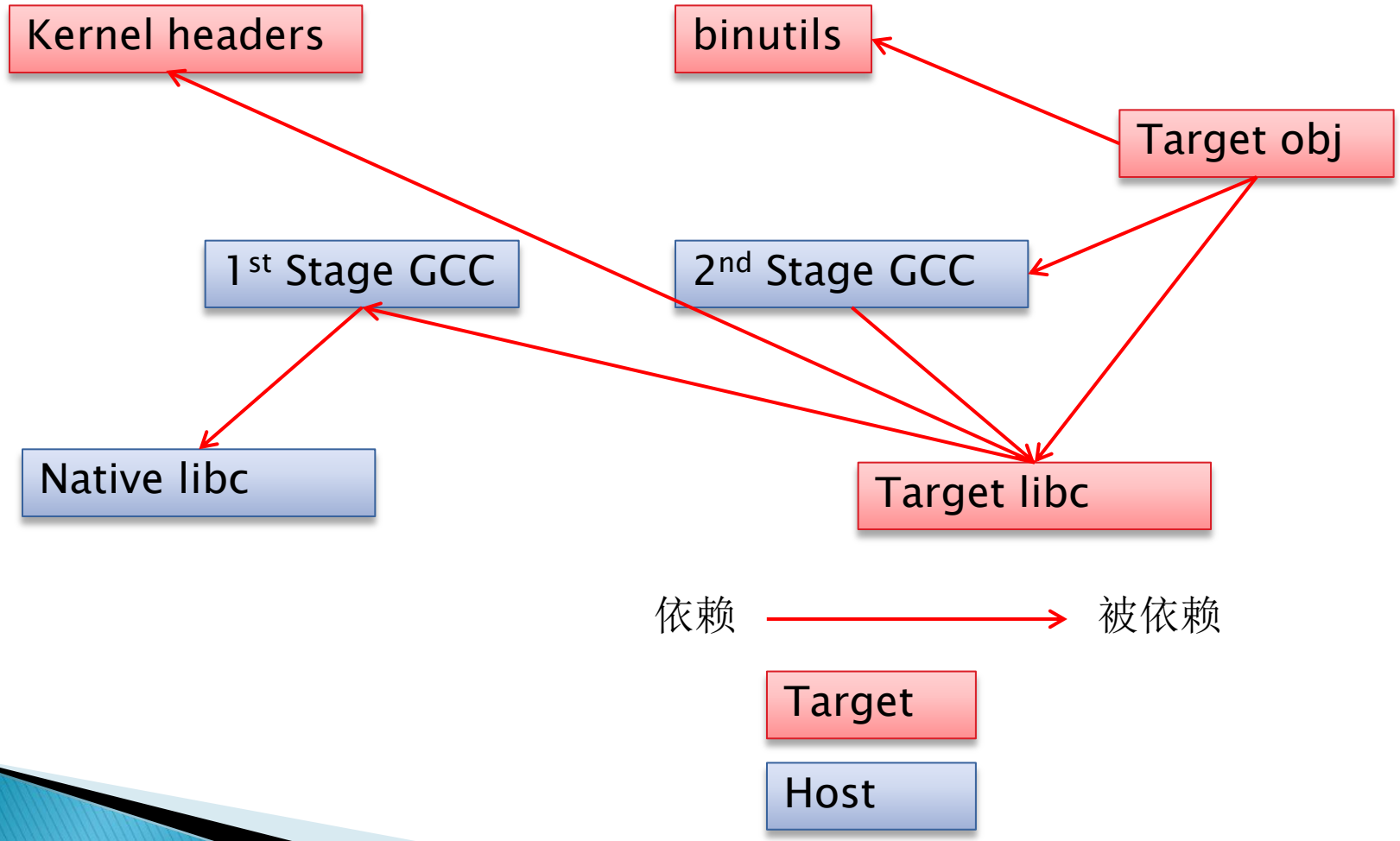


- 交叉编译中build、host、target的概念
- 交叉编译中的依赖关系
- 交叉编译的过程

Host与Build相同时的交叉编译



交叉编译中的组建依赖关系



交叉编译的过程

- 本地环境的准备
- 源码获取
- 编译二进制工具
- 1st Stage GCC – 临时编译环境
- 用临时编译环境编译目标C库
- 2nd Stage GCC – 目标编译环境

交叉编译的完整过程

① 安装本地编译环境：gcc、make、automake
② 配置交叉编译环境
③ 下载交叉编译工具链源码及补丁
④ 配置交叉编译工具链安装目录
⑤ 安装内核头文件
⑥ 编译二进制工具包binutils
⑦ 编译1st stage gcc
⑧ 编译目标c库 libc
⑨ 编译2nd stage gcc

完整的编译过程请参照 http://cross-lfs.org/view/clfs-embedded/arm/cross-tools/chapter.html，其中的内容全部经过实验验证通过，保证能够得到最终结果

安装本地编译环境

- 安装本地编译环境：gcc、make、automake

```
$ aptitude install gcc automake make
```

验证安装成功：

```
$ aptitude search '~ngcc'|egrep '^i\ +gcc'
```

这里有个空格

或者：

```
$ gcc -v
```

返回当前系统gcc的版本信息

对于make，可以采取同样的方法验证安装。而automake是一个工具集合，可以采用下列命令验证安装：

```
$automake --version
```

返回当前系统automake的版本信息

配置交叉编译本地环境

- 创建交叉编译用户clfs、交叉编译环境根目录和登陆脚本

```
$ sudo groupadd clfs
$ sudo useradd -s /bin/bash -g clfs -m -k /dev/null clfs
```

为用户clfs创建密码并以clfs用户身份登录：

```
$ sudo passwd clfs
$ su - clfs
```

这里的“-”不是参数前的“-”，两边都有空格

创建交叉编译环境根目录并用一个环境变量CLFS指向它：

```
$ mkdir -p /mnt/clfs
$ export CLFS=/mnt/clfs
```

存放交叉编译工具链源码及补丁

创建一个虚拟磁盘/dev/sdb，仅含一个分区/dev/sdb1，挂载到/mnt/clfs，这样配置好的整个交叉编译环境就全部存放在这个虚拟磁盘内。使用用户clfs登陆系统，之后：

```
$ echo export CLFS=/mnt/clfs >> .bashrc
$ chmod 777 $CLFS
$ sudo chown -Rv clfs $CLFS
$ mkdir -vp $CLFS/src
```

配置交叉编译本地环境

配置登录Shell脚本 .bash_profile：

```
$ cat > ~/.bash_profile << "EOF"
exec env -i HOME=${HOME} TERM=${TERM}  PS1='\u:\w\$ '  /bin/bash
EOF
```

设置非登陆Shell脚本 .bashrc：

```
$cat > ~/.bashrc << "EOF"
set +h
umask 022
CLFS=/mnt/clfs
LC_ALL=POSIX
PATH=${CLFS}/cross-tools/bin:/bin:/usr/bin
export CLFS LC_ALL PATH
EOF
```

在clfs的家目录下生成两个配置文件.bash_profile和.bashrc

清除原有的gcc 编译标志，为源码目录设置一个环境变量方便访问：

```
$ echo unset CFLAGS  >>  .bashrc
$ echo unset CXXFLAGS  >>  .bashrc
$ echo export SRC=${CLFS}/src >> .bashrc
```

下载交叉编译工具链源码及补丁

所有文件放入$CLFS/src

Binutils (2.23.2) - 20937 KB:
- Home page: http://www.gnu.org/software/binutils/
- Download: http://ftp.gnu.org/gnu/binutils/binutils-2.23.2.tar.bz2

GCC (4.7.3) - 80961 KB:
- Home page: http://gcc.gnu.org
- Download: ftp://gcc.gnu.org/pub/gcc/releases/gcc-4.7.3/gcc-4.7.3.tar.bz2

GMP (5.1.2) - 2143 KB:
- Home page: http://gmplib.org/
- Download: http://ftp.gnu.org/gnu/gmp/gmp-5.1.2.tar.bz2

Linux (3.0.80) - 75134 KB:
- Home page: http://www.kernel.org
- Download: http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.0.80.tar.bz2

MPC (1.0.1) - 609 KB:
- Home page: http://www.multiprecision.org/
- Download: http://www.multiprecision.org/mpc/download/mpc-1.0.1.tar.gz

MPFR (3.1.2) - 1196 KB:
- Home page: http://www.mpfr.org/
- Download: http://gforge.inria.fr/frs/download.php/32210/mpfr-3.1.2.tar.bz2

musl-libc (0.9.14) - 806 KB:
- Home page: http://musl-libc.org/
- Download: http://www.musl-libc.org/releases/musl-0.9.14.tar.gz

Binutils musl-libc Patch - 1 KB:
- Download: http://patches.cross-lfs.org/embedded-dev/binutils-2.23.2-musl-1.patch

GCC musl-libc Patch - 26 KB:
- Download: http://patches.cross-lfs.org/embedded-dev/gcc-4.7.3-musl-1.patch

配置目标系统

设定指示目标系统体系结构的环境变量：

```
$ export CLFS_FLOAT=hard
$ export CLFS_FPU=vfpv3-d16
$ export CLFS_HOST=$(echo ${MACHTYPE} | sed "s/-[^-]*/-cross/")
$ export CLFS_TARGET=arm-linux-musleabihf
$ export CLFS_ARCH=arm
$ export CLFS_ARM_ARCH=armv7-a
```

最好是把以上脚本语句都写在 .bashrc 的最后

Table 4.1. ARM Hard Floating Point Versions

fpa	fpe2	fpe3	maverick
vfp	vfpv3	vfpv3-fp16	vfpv3-d16
vfpv3-d16-fp16	vfpv3xd	vfpv3xd-fp16	neon
neon-fp16	vfpv4	vfpv4-d16	fpv4-sp-d16
neon-vfpv4			

创建交叉编译工具安装目录

- 在$CLFS下创建目标系统的交叉编译工具目录，整个叉编译工具链都将被安装在该目录下

```
$mkdir  -p  ${CLFS}/cross-tools/${CLFS_TARGET}
$ln -sfv  .  ${CLFS}/cross-tools  /${CLFS_TARGET}/usr
$echo export SYSROOT=${CLFS}/cross-tools/${CLFS_TARGET} >> .bashrc
```

- 现在在$CLFS目录下有两个目录：src目录里面是下载的所有编译工具链组件源码包；cross-tools目录下是交叉编译工具安装目录arm-linux-musleabihf，我们用环境变量SYSROOT来指向它

安装目标系统的内核头文件

- GCC 1st Stage的编译需要使用到目标系统的内核头文件，先将其安装至$SYSROOT

```
$make  mrproper
$make  ARCH=${CLFS_ARCH}  headers_check
$make  ARCH=${CLFS_ARCH}  INSTALL_HDR_PATH=${CLFS}/cross-tools/
${CLFS_TARGET}    headers_install
```

- 安装好后在$SYSROOT/include下可以看到如下目录：

asm,asm-generic,drm,linux,mtd,rdma,scsi,sound,video,xen

安装目标系统的二进制工具包binutils

- Binutils的编译不依赖宿主环境的头文件和库，只需要指定编译的binutils针对的目标系统架构即可

```
$cd  $SRC
$tar  jxf   binutils-2.23.2.tar.bz2
$cd  binutils-2.23.2
$patch  -Np1  ../binutils-2.23.2-musl-1.patch
$mkdir -v ../binutils-build
$cd ../binutils-build
$../binutils-2.23.2/configure   \
 --prefix=${CLFS}/cross-tools \
 --target=${CLFS_TARGET}  \
 --with-sysroot=${SYSROOT}  \
 --disable-nls  \
 --disable-multilib
$make  configure-host
$make
$make install
```

这里是阿拉伯数字1，给binutils打上补丁，以便能配合musl的使用

Binutils和后面的gcc都要求在源码目录外编译，我们创建一个专门的文件夹用来存放编译结果

一行写不下的时候采取这种方法来分行写，注意“\”前面要留空格

安装了哪些二进制工具？参见：http://cross-lfs.org/view/clfs-embedded/arm/cross-tools/binutils.html

1st Stage GCC – 准备源码

```
$cd  $SRC
$tar  jxf  gcc-4.7.3.tar.bz2
$mkdir gcc-build-1st
$cd  gcc-4.7.3
$patch  -Np1  ../gcc-4.7.3-musl-1.patch
$tar  jxf  ../mpfr-3.1.2.tar.bz2
$mv -v  mpfr-3.1.2  mpfr
$tar  jxf  ../gmp-5.1.2.tar.bz2
$mv  -v  gmp-5.1.2  gmp
$tar  zxf  ../mpc-1.0.1.tar.gz
$mv  -v  mpc-1.0.1  mpc

$cd gcc-build-1st
```

把几个编译GCC要用到的插件mpfr, gmp, mpc都解压到gcc源码目录并重命名（去掉版本号）

1st Stage GCC – 编译并安装

```
$../gcc-4.7.3/configure \
--prefix=${CLFS}/cross-tools \
--build=${CLFS_HOST} \
--host=${CLFS_HOST} \
--target=${CLFS_TARGET} \
--with-sysroot=${SYSROOT} \
--disable-nls \
--disable-shared \
--without-headers \
--with-newlib \
--disable-decimal-float \
--disable-libgomp \
--disable-libmudflap \
--disable-libssp \
--disable-libatomic \
--disable-libquadmath \
--disable-threads \
--enable-languages=c \
--disable-multilib \
--with-mpfr-include=$(pwd)/../gcc-4.7.3/mpfr/src \
--with-mpfr-lib=$(pwd)/mpfr/src/.libs \
--with-arch=${CLFS_ARM_ARCH} \
--with-float=${CLFS_FLOAT} \
--with-fpu=${CLFS_FPU}
```

这些--disable，--enable, --with, --without选项决定了gcc是否具备哪些特定的功能，这里disable很多特性是因为我们目前只需要一个临时的，能够用来编译目标系统所用libc库的gcc（而这是本地编译环境中的gcc无法做到的）

配置完成后运行make编译gcc并make install-gcc 安装，注意all-gcc all-target-libgcc，以及install-gcc, install-target-libgcc都是gcc的开发者事先设定好的特定make目标，方便用户进行编译和安装。安装后在$SYSROOT/bin下

```
$make  all-gcc  all-target-libgcc
$make install-gcc  \
  install-target-libgcc
```

编译目标系统的C语言库并安装

```
$cd  $SRC
$tar  zxf  musl-0.9.14.tar.gz
$cd musl-0.9.14
$CC=${CLFS_TARGET}-gcc ./configure --prefix=/ --target=${CLFS_TARGET}
$CC=${CLFS_TARGET}-gcc make
$DESTDIR=${SYSROOT} make install
```

ld-musl-libc	The musl-libc dynamic linker / loader
libc	The C library
libcrypt	The cryptographic library
libdl	The musl-libc dynamic linker / loader library
libm	The math library
libpthread	The POSIX thread library
librt	The clock and timer library

安装后目标系统编译工具安装目录下的C语言库内容

2nd Stage GCC – 编译并安装

```
$cd  $SRC
$mkdir -v gcc-build-2nd
$cd gcc-build-2nd
$../gcc-4.7.3/configure \
 --prefix=${CLFS}/cross-tools \
 --build=${CLFS_HOST} \
 --target=${CLFS_TARGET} \
 --host=${CLFS_HOST} \
 --with-sysroot=${SYSROOT} \
 --disable-nls \
 --enable-languages=c \
 --enable-c99 \
 --enable-long-long \
 --disable-libmudflap \
 --disable-multilib \
 --with-mpfr-include=$(pwd)/../gcc-4.7.3/mpfr/src \
 --with-mpfr-lib=$(pwd)/mpfr/src/.libs \
 --with-arch=${CLFS_ARM_ARCH} \
 --with-float=${CLFS_FLOAT} \
 --with-fpu=${CLFS_FPU}

$make
$make install
```

第二阶段的源码和第一阶段完全一样，所以不用重复做一遍解压mpfr, gmp, mpc插件的动作，但需要另建一个build目录

和第一阶段相比，少了那些选项，多了那些选项？这些选项都有什么作用？（第三次作业）

交叉编译器的验证

我要在一台PowerPC主机上编译生成一个gcc，这个gcc能够运行在x86主机上，并且专门用来编译运行在ARM平台上的应用程序，请问--build，--host, --target分别应该如何设置？（第三次作业）

```
 $cat > $SRC/hello.c << "EOF"
#include <stdio.h>
#include <math.h> /* no use, for link test */
int main ()
{
        printf("hello world!\n");
        return 0;
}
EOF
```

用自制的交叉编译器编译一个ARM版的hello world吧！（第三次作业）请将你编译出来的gcc可执行程序打包发送给我

```
 $./${CLFS}/cross-tools/bin/${CLFS_TARGET}-gcc -g -o hello

 $ file hello
 $./${CLFS}/cross-tools/bin/${CLFS_TARGET}-objdump -x hello |less
```

看看这个文件是什么类型？它能在编译它的机器上运行么？（第三次作业）

看看这个ARM版的hello world程序内部，它都链接到了哪些库，这些库都在哪里？（第三次作业）

本章总结

- GNU/Linux 编译工具链介绍
- 交叉编译原理
- GCC交叉编译工具链的构建过程
- 交叉编译工具链的构建是嵌入式软件开发的基础，后续介绍的Bootloader定制、内核定制、根文件系统定制、Android系统工程的编译等都要使用到它，熟练掌握交叉编译是嵌入式软件工程师的基本功