osx 下搭建操作系统开发环境之 32 位交叉开发工具集（gcc +gdb）v1.1

boxcounter

November 4, 2013

目录

版本记录

- v1.0 – 2013-09-07，初始发布。
- v1.1 – 2013-11-04，增加“编译交叉版的 gdb”章节。

1 前言

《linux、osx 下搭建操作系统开发环境的完整步骤》一文中讲解了一些基本的搭建方法，并提供了一个 nasm 汇编编写的简单的系统内核源码。实际开发过程中更多使用的是 C 语言，就需要有一个配套的 C 编译器。因为我使用的可执行文件是 elf 格式，所以我选择的是 GCC。但是 osx 下安装的 GCC 生成的是 osx 的可执行文件格式，并不是 elf。所以我需要一个能在 osx 下生成 elf 的 GCC，俗称的交叉编译器。

我的环境：osx 10.8.4 & 10.9

2 安装 osx 版的 gcc

```
brew install gcc48
```

推荐下载最新的稳定版 gcc。

3 配置编译环境

1. 下载 gcc 源码

   根据参考资料 1 的建议，最好使用最新的 gcc 来进行编译，被编译的源码也推荐使用一样版本的。也就是说，用 gcc 4.8.1 来编译 gcc 4.8.1 的源码。

   下载源码包并解压，得到的目录名称之为“$gcc-4.8.1”。

2. 下载 gcc 依赖项

   需要的依赖项有：

   (a) GNU Binutils

   (b) GMP

   (c) MPFR

   (d) MPC

   将它们都解压出来，把解压出来的 (b)、(c)、(d) 的目录都放到 gcc 源码目录下。都需要去掉版本号，比如解压出来的目录名为“mpc-1.0.1”，那么现在就是“$gcc-4.8.1/mpc”。(a) 无需这么做，因为它需要单独编译，参考后续的步骤 4。

   其中 GMP 源码包是 lzip 压缩格式，需要下载 lzip 工具解压 (brew 安装)。

3. 下载 gdb 源码

   下载源码包并解压，得到的目录名称之为“$gdb-7.6.1”。

4. 设置环境变量

```
export CC=/usr/local/bin/gcc-4.8
export CXX=/usr/local/bin/g++-4.8
export CPP=/usr/local/bin/cpp-4.8
export LD=/usr/local/bin/gcc-4.8
```

   这些都是 brew 版 gcc4.8.1 的软链接。如果不设置，那么会使用系统中默认自带的工具，这些工具版本都很陈旧。比如 osx 10.8.4 带的/usr/bin/gcc 是 4.2 版本的。

```
export PREFIX=$HOME/opt/cross
export TARGET=i586-elf
export PATH="$PREFIX/bin:$PATH"
```

   这些是编译时候使用的选项。

5. 编译交叉版的 binutils

```
cd $binutils-x.y.z
mkdir build-binutils
cd build-binutils
../configure --target=$TARGET --prefix="$PREFIX" --disable-nls
make
make install
```

4 编译交叉版的 gcc

```
cd $gcc-4.8.1
mkdir build-gcc
cd build-gcc
../configure --target=$TARGET --prefix="$PREFIX" --disable-nls --enable-languages=c,c++ --
without-headers
make all-gcc
make all-target-libgcc
make install-gcc
make install-target-libgcc
```

完成后，在“~/opt/cross/bin”下就能看到编译好的交叉版的编译套件了，包括“i586-elf-gcc”、“i586-elf-g++”和“i586-elf-ld ”等等。可以用“$HOME/opt/cross/bin/$TARGET-gcc -version”来验证一下版本是否正确。

另外，为了方便使用，可以在.bashrc 或者.zshrc 中调整环境变量：

```
export PATH="$HOME/opt/cross/bin:$PATH"
```

5 测试源码

现在咱有了交叉编译器了，试试效果吧：

kernel.c

```
#include ”multiboot2.h”


#define INFO_REQ_COUNT  2

struct antos_multiboot_header_tag_information_request
{
    struct multiboot_header_tag_information_request info_req __attribute__((aligned(MULTIBOOT_TAG_ALIGN)));
    multiboot_uint32_t req[INFO_REQ_COUNT];
} __attribute__((packed));


struct antos_multiboot_header
{
    struct multiboot_header header __attribute__((aligned(MULTIBOOT_TAG_ALIGN)));
    struct antos_multiboot_header_tag_information_request info_req __attribute__((aligned(MULTIBOOT_TAG_ALIGN)));
    struct multiboot_header_tag end __attribute__((aligned(MULTIBOOT_TAG_ALIGN)));
} __attribute__((packed));


struct antos_multiboot_header amb =
{
    {
        MULTIBOOT2_HEADER_MAGIC,
        MULTIBOOT_ARCHITECTURE_I386,
        sizeof(struct antos_multiboot_header),
        -(MULTIBOOT2_HEADER_MAGIC + MULTIBOOT_ARCHITECTURE_I386 + sizeof(struct antos_multiboot_header))
    },

    {
        {
            MULTIBOOT_HEADER_TAG_INFORMATION_REQUEST,
            MULTIBOOT_HEADER_TAG_OPTIONAL,
            sizeof(struct antos_multiboot_header_tag_information_request)
        },
        MULTIBOOT_TAG_TYPE_BASIC_MEMINFO,
        MULTIBOOT_TAG_TYPE_FRAMEBUFFER
    },

    {
        MULTIBOOT_HEADER_TAG_END,
        MULTIBOOT_HEADER_TAG_OPTIONAL,
        sizeof(struct multiboot_header_tag)
    }
};
```

```
void breakpoint()
{
    asm("xchg␣%bx,␣%bx");
}


int _start()
{
    breakpoint();
    return 0;
}
```

multiboot2.h 头文件是从 grub2.0.0 的源码里拷贝过来的，主要定义了符合 multiboot2 规范的数据结构。

编译方法:

```
~/opt/cross/bin/i586-elf-gcc -c -o kernel.o kernel.c
~/opt/cross/bin/i586-elf-ld -Ttext=0x100000 -o kernel.bin kernel.o
```

“-Ttext=0x100000”是为了让代码段加载到 0x100000，而不是默认的 08048074（我的环境中），后者超出我的 bochs 虚拟机的物理内存空间。

然后用 kernel.bin 替换之前的虚拟磁盘中的同名文件，再运行 bochs 虚拟机就能看到熟悉的 magic breakpoint 了。

6 编译交叉版的 gdb

```
cd $gdb-7.6.1
mkdir build-$TARGET
cd build-$TARGET
../configure --target=$TARGET --prefix=$PREFIX --disable-nls
make
sudo make install
```

完成后，在“~/opt/cross/bin”下就能看到编译好的交叉版的 i586-elf-gdb 了。

7 参考资料

1. GCC Cross-Compiler