**1.前言**

公司要求在Linux系统上实现交叉编译并动态下发Android和iOS的静态库或者动态库，而其中最为关键的就是Android NDK和iOS工具链的生成。由于Android 的实现起来相对容易，这里重点讲述iOS端生成工具链的实现过程。下面的操作全部都是在Macbook Pro上完成的。

**2.所需资源**

2.1 VWware Fusion 10(<https://my.vmware.com/web/vmware/details?downloadGroup=FUS-1001&productId=688&rPId=18811>)

2.2 Ubuntu（16.04及以上版本）(<https://www.ubuntu.com/download/desktop>)

  2.3 LLVM/Clang（>=5.0）(<http://releases.llvm.org/download.html#5.0.0>)

  2.4 openssl（<https://github.com/openssl/openssl>）

  2.5 automake（<http://ftp.gnu.org/gnu/automake/>）

  2.6 autogen（<http://ftp.gnu.org/gnu/autogen/>）

  2.7 libtool（<http://ftp.gnu.org/gnu/libtool/>）

  2.8 autoconf（<http://ftp.gnu.org/gnu/autoconf/>）

2.9 libssl-dev（<https://pkgs.org/download/libssl-dev>）

2.10 python（<https://www.python.org/downloads/>）

2.11 cctools-port（<https://github.com/tpoechtrager/cctools-port>）

2.12 cmake（<https://cmake.org/download/>）

2.13 iOS-SDK（<http://resources.airnativeextensions.com/ios/>）

**3.资源配置和安装**

**3.1 VWware Fusion 10 和Ubuntu 的安装方法**

<http://blog.csdn.net/jackjia2015/article/details/50757430>

**3.2 LLVM/Clang 的安装方法**

3.2.1： 下载LLVM、Clang、Compiler-rt（可选）、Clang-tools-extra（可选）源码并解压。LLVM解压后的文件名改为llvm，Clang解压后的文件名改为clang，Compiler-rt解压后的文件名改为compiler-rt,Clang-tools-extra解压后的文件名改为clang-tools-extra

3.2.2： 把clang放入llvm/tools/目录下，把compiler-rt放入llvm/projects/目录下，把clang-tools-extra放入llvm/tools/clang/tools/目录下

3.2.3： 返回到llvm的同级目录，mkdir llvmbuild（文件名自己取，合理即可）

3.2.4 ：cd llvmbuild,然后执行 cmake ../llvm -DLLVM\_ENABLE\_ASSERTIONS=OFF -DCMAKE\_BUILD\_TYPE=RELEASE

3.2.5 ：3.2.4执行结束后，执行make -jN（其中N为你为你的虚拟机设置的处理器内核数，如下图所示）



如上图，我为我的虚拟机设置了1个处理器内核，所以我应该写 make -j1

3.2.6：3.2.5 执行完成成功后，执行 sudo make install ，到此llvm、clang的安装已经完毕

**3.3 安装openssl**

3.3.1 官方给出的方法

3.3.1.1 解压文件

tar -xf openssl-1.0.1g.tar.gz（建议下载最新的版本）

3.3.1.2 配置

./config --prefix=/usr/local/openssl --openssldir=/usr/local/openssl

3.3.1.3 编译

make

3.3.1.4 安装

sudo make install

3.3.1.5 设置环境变量

sudo gedit ~/.bashrc,添加export PATH=/usr/local/openssl/bin:$PATH，保存并推出。最后执行 source ~/.bashrc 使环境变量生效。

3.3.2 建议使用的安装方法

3.3.2.1 直接使用 sudo apt-get install openssl 指令安装

**3.4 安装automake**

3.4.1 建议直接使用 sudo apt-get install automake 指令安装

**3.5 安装autogen**

3.5.1 建议直接使用 sudo apt-get install autogen 指令安装

**3.6 安装libtool**

3.6.1 建议直接使用 sudo apt-get install libtool 指令安装

**3.7 安装autoconf**

3.7.1 建议直接使用 sudo apt-get install autoconf 指令安装

**3.8 安装 libssl-dev**

3.8.1 建议直接使用 sudo apt-get install libssl-dev 指令安装

**3.9 安装 python**

3.9.1 建议直接使用 sudo apt-get install python 指令安装

**3.10 cctools-port**

3.10.1 直接解压，无需其他操作

**3.11 安装cmake**

3.11.1 下载源码后解压

3.11.2 cd 文件名，执行./bootstrap指令

3.11.3 make -jN（此处的N根据你分配的虚拟机的处理器内核数而定）

3.11.4 前面都没问题之后，执行sudo make install,都完成后你可以执行 cmake --version来验证是否安装成功

**3.12 iOS SDK 打包步骤（在此我是利用Xcode9.1，打包iOS11.1SDK的。如果需要打包iOS8、iOS9、iOS10等SDK，下面的方法同样适用，注意下面的tmp是我们临时创建的路径）**

3.12.1： SDK=$(ls -l /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs | grep " -> iPhoneOS.sdk" | head -n1 | awk '{print $9}')

3.12.2：cp -r /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS.sdk /tmp/$SDK 1>/dev/null

3.12.3：cp -r /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1 /tmp/$SDK/usr/include/c++ 1>/dev/null

3.12.4：pushd /tmp

3.12.5：tar -cvzf $SDK.tar.gz $SDK（至此我们需要的对应版本的iOS SDK已经打包完成）

3.12.6：rm -rf $SDK

3.12.7：mv $SDK.tar.gz ~

3.12.8：popd （至此我们在将要生成iOS工具链前的准备工作已经全部完成）

**4.生成iOS工具链（在此生成的支持armv7、armv7s的工具链，我用的是iOS9.3sdk）**

**4.1 生成arm64工具链**

4.1.1 cd cctools-port

4.1.2 执行 IPHONEOS\_DEPLOYMENT\_TARGET=8.0 usage\_examples/ios\_toolchain/build.sh ~/iPhoneOS11.1.sdk.tar.gz arm64

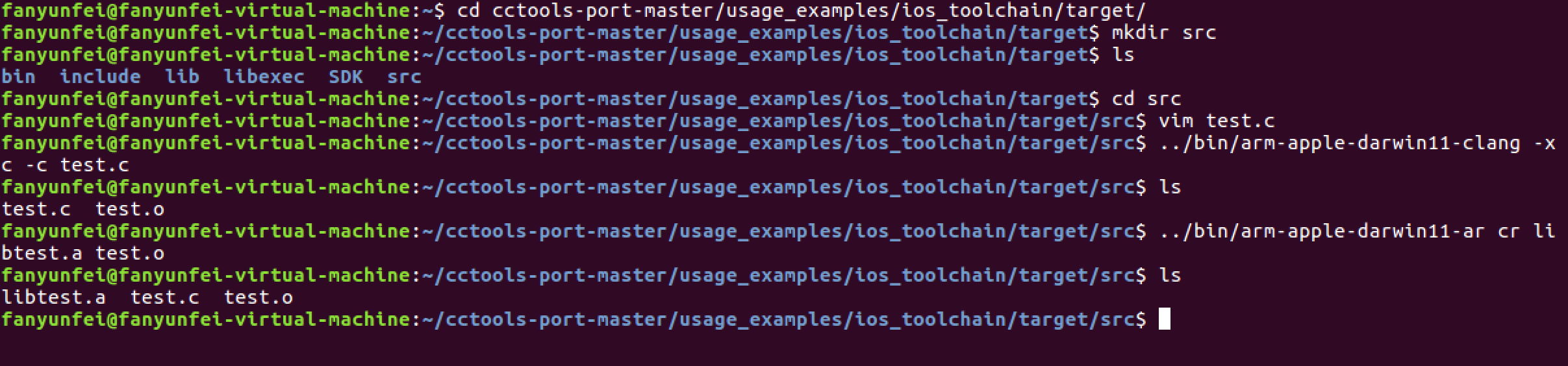
4.1.3 制作工具链成功后会提示\*\*\* all done \*\*\*，此时cctools-port/usage\_examples/ios\_toolchain/下的target 就是我们要的工具链（至此arm64的工具链已经生成）

4.1.4 进入/usr/local,我们创建一个文件，命名为iOS-arm64,然后返回到cctools-port/目录 ，执行 sudo mv usage\_examples/ios\_toolchain/target /usr/local/iOS-arm64

4.1.5 执行sudo cp /usr/local/iOS-arm64/target/lib/libtapi.so /usr/lib/

4.1.6 执行 export PATH=$PATH:/usr/local/iOS-arm64/bin

4.1.7 验证我们的工具链是否可以使用(下面的图片很重要)



4.1.7.1 cd cctools-port-master/usage-examples/ios\_toolchain/target/

4.1.7.2 mkdir src （创建文件src）

4.1.7.3 vim test.c（创建一个.c文件）

4.1.7.4 ../bin/arm-apple-darwin11-clang -xc -c test.c（编译.c文件，生成.o文件）

4.1.7.5 ../bin/arm-apple-darwin11-ar cr libtest.a test.o（把.o文件编译成.a库，我取静态库的名字为libtest.a）

4.1.7.6 如果你顺利的走到这一步的话，Congratulations, you 0.0000001 meters distance from success，下面就是把test.c和libtest.a文件 copy出来，在你的Xcode上验证我们用iOS -arm64工具链生成的静态库，能否正常使用。

**4.2 生成其它工具链方法基本同上**