从glibc升级中学到了什么

1. 为什么要交叉编译？

Host是x86架构的，使用的是x86-64的指令集。而BMC运行在arm上，使用的是armv6的指令集。CPU不同，能识别的指令集也不同。所以arm上面只能识别arm架构的指令集。

1. 那么要怎么编译出来arm架构的指令集呢？

我们使用的源码是一样的，对于glibc来说，使用的就是普通的C代码。编译器通过编译链接把源码转换成机器可以识别执行的文件。

Gcc -S main.c //生成汇编文件main.s

Gcc -c main.c //可重定位文件main.o

Objdump -d main.o // 查看机器代码文件内容

Gcc -o main main.c //生成可执行文件

源码是一样的，所以这里的差别就是编译器不一样。在x86-64上执行gcc的时候使用的是x86-64的编译器，在arm上执行gcc的时候使用的就是arm的gcc（比如arm-vendor-os-gnueabi-gcc）。

1. 为什么不直接在arm上面编译源码呢？

Arm上面的资源较少，光是运行它自己的操作系统就占用了大部分资源，所以通常不在arm上面直接编译代码。（open: arm上操作系统占用了多少资源，编译需要多少资源？）

1. 因此我们只能在本地x86-64的机器上编译跑在arm上的程序。Glibc共享链接库虽然不是一个独立的程序，但它会被arm上其他程序调用，所以也需要用交叉编译器编译。
2. Glibc共享链接库是什么？

Glibc共享链接库包含多个以.so结尾的文件，这些文件是已经编译但没有链接的共享文件。Glibc提供了众多接口，系统调用API，（如malloc）通过动态链接的方式被链接和加载。

1. 既然交叉编译这么好，那哪里能买到交叉编译工具呢？

幸运的是我们并不需要买，只需要自己下载安装可以生成交叉编译工具链的工具就可以了。（有点绕是不是？这里可以插一张图解释说明）我用的是crosstool-ng，因此根据前面说的，交叉编译工具和指令集相关，因此我们需要告诉crosstool-ng我们需要的指令集，（比如armv5），运行的操作系统（linux3.2.0），系统线程模型等。