Linux内核-实战

源自**技术简说**第2讲
编译器,交叉编译器和大小端

讲师：杨文川

内容

- 1 编译器和交叉编译器的介绍
- 2 交叉编译器的命名规则
- 3 如何交叉编译C代码
- 4 大端、小端的通俗讲解

引言

- 要学习linux内核开发，编译器或者交叉编译器是绕不过去的，本讲内容包括：
  - 编译器和交叉编译器的介绍
  - 交叉编译器的命名规则
  - 如何交叉编译C代码
  - 大端、小端的通俗讲解
  - 如何判断你的系统是大端系统还是小端系统

1 编译器和交叉编译器

- 编译器的作用，是把人类容易标识的程序代码，转换为机器所理解的机器代码
  - **编译器 编译出来的二进制文件，能在编译器所在硬件平台上运行，**
- 交叉编译器也是实现类似的功能，只不过：
  - **交叉编译器生编译出来的二进制文件，不能在自己所在的硬件平台上运行，需要移植到对应的平台上运行。**

为什么会有交叉编译器？

- 因为早期的嵌入式硬件(比如arm) 的性能较低，不适合在上面进行程序编译这种耗费cpu的任务
- 所以，交叉编译器应用而生了：我们在pc机上(x86架构)安装arm的交叉编译器，并在pc机上编译出能在arm体系上运行的二进制代码，这就是交叉编译器所做的事情。

- 了解这个历史原因之后，可能会有人问：现在arm的性能已经有了大幅提升，还需要交叉编译器吗？
- 答案是：需要。

- 有些arm的Linux系统中，已经可以使用gcc，可以编译一些比较上层的应用，但是这是有一个前提的：那就是Linux系统已经运行起来了。
- 那如果arm的板子连Linux内核都没有，你怎么使用gcc？
- 另外Linux内核是需要提前做好的，如果arm系统都没有起来，怎么编译Linux内核？ 你只能通过在PC机上进行交叉编译。
- 以上就是关于交叉编译器的作用的介绍，接下来我们看一下**交叉编译器的命名规则**。

2交叉编译器的命名规则

- 交叉编译工具链的命名规则一般为：
- $arch [-$vendor] -$os [-[gnu][eabi][hf]]-gcc
  - 例如： arm-linux-gnueabi-gcc
- arch - 体系架构，如arm，mips等，不可省略
- vendor - 工具链提供商，可省略
- os - 目标操作系统，不可省略
- gnu: 加gnu表示编译器使用的是gnu glibc的库
- eabi - 嵌入式应用二进制接口(Embedded Application Binary Interface)，可选的参数包括：
  - abi: 二进制应用接口。
  - eabi: 嵌入式二进制应用接口，主要针对嵌入式平台。

- el：表示使用软浮点处理单元(softfp)。其实在armel中，关于浮点数计算的约定有三种。
  - 以gcc为例，对应的-mfloat-abi参数值有三个：soft,softfp,hard
  - soft是指所有浮点运算全部在软件层实现，效率当然不高，会存在不必要的浮点到整数、整数到浮点的转换，只适合于早期没有浮点计算单元的ARM处理器；
  - softfp是armel的默认设置，它将浮点计算交给FPU处理，但函数参数的传递使用通用的整型寄存器而不是FPU寄存器；
  - hard则使用FPU浮点寄存器将函数参数传递给FPU处理。
  - 注意：在兼容性上，soft与后两者是兼容的，但softfp和hard两种模式不兼容，armel使用softfp，因此将hard模式的armel单独作为一个abi，称之为armhf
- hf: 表示使用硬件浮点处理单元(hard)

- 以上是交叉编译器的命名规则，下面来看几个例子：
  - arm-none-eabi-gcc： 针对arm的裸机程序的gcc编译器，如果你在arm板子上直接运行c程序，可能需要这个。
  - arm-uclinux-gcc： 针对arm的uclinux的交叉编译器
  - arm-linux-gcc ： 针对arm的linux的交叉编译器
  - arm-linux-gnueabi-gcc：针对arm的linux的交叉编译器，此编译器使用gnu glibc的库
  - arm-linux-gnueabihf-gcc： 针对arm的linux的交叉编译器，此编译器使用硬件浮点处理浮运算。
  - arm-linux-gnueabiel-gcc： 针对arm的linux的交叉编译器，此编译器使用软浮点处理浮点运算。

3编译一个最简单的程序

- 了解了交叉编译器之后，我们用它来编译一个最简单的程序。代码如下：
  - #include <stdio.h>
  - void main()
  - {
  - printf("hello arm");
  - }
- 在pc上编译：
  - # arm-linux-gnueabi-gcc -o main main.c -static
- 在pc上运行：
  - root@ubuntu:/home/jinxin/app# ./main
  - bash: ./main: cannot execute binary file: Exec format error

- 运行出错了，为什么呢？
- 因为当前pc是x86架构，而刚编译出来的二进制文件是arm格式的：
    - root@ubuntu:/home/jinxin/app# file main
    - main: ELF 32-bit **LSB** executable, **ARM**, EABI5 version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=ec9b455a1c1df94060c7364d0efc42975207fca9, not stripped
- 如果想要正常运行，需要把编译出来的二进制代码拷贝到arm的板子上去运行。
- 以上的输出，有一个需要强调的地方：**LSB**
- **这里的LSB是干什么的呢？接下来我们就来看看大小端。**

4大小端

- **LSB(Least Significant Bit)最低有效位,表示可执行文件是小端格式的。**
- 大端和小端主要描述了数据在内存里的存储的顺序。
- 大端模式，是指数据的高字节保存在内存的低地址中，而数据的低字节保存在内存的高地址中；
- 小端模式，是指数据的高字节保存在内存的高地址中，而数据的低字节保存在内存的低地址中；

比如：

0x123456在内存中的存储方式

- 大端模式

```
低地址 -----> 高地址
0x12 | 0x34 | 0x56
```

- 小端模式

```
低地址 -----> 高地址
0x56 | 0x34 | 0x12
```

- x86是小端模式，而网络字节序是大端模式。所以，专门出了网络字节序和主机字节序之间的转换接口(ntohs、htons、ntohl、htonl)。
- 而arm则支持大小端模式，但是同一时刻只能支持一种，一般情况下，arm配置为大端模式，这也是跟x86分庭抗礼的意思。
- **针对arm，注意：如果arm硬件配置为大端，那么编译器要使用大端配置，内核、文件系统、应用程序也要编译为大端的格式。**
- 那如何知道你的系统是大端还是小端呢？

5一个简单的测试

- 一个测试你系统是大端还是小端的程序：
  - #include <stdio.h>
  - void main()
  - {
  - int a = 0x1234;
  - char b = *(char*)&a;
  - if(0x34 == b){
  - printf("your system is little endian\n");
  - } else {
  - printf("your system is big endian\n");
  - }
  - }
- 在某pc上执行，输出：
  - root@ubuntu:/home/jinxin/app# ./endian
  - your system is little endian

谢　谢