

交叉编译缺动态链接库的解决方法

龙芯中科 嵌入式

June 11, 2012

Contents

1 察看elf文件依赖的动态链库	2
2 解决办法	3
2.1 静态编译	3
2.2 从工具链里面拷贝需要的动态链接库	3
2.3 单独的动态链接库	3
3 编译busybox	4

1 察看elf文件依赖的动态链库

利用ldd和readelf都能察看elf文件依赖的动态链接库，但ldd不能跨平台使用，因此我们这里介绍的是用readelf察看用户程序依赖的动态链接库的办法。

```

1 readelf -d mount
2
3 Dynamic section at offset 0x7b8 contains 23 entries:
4   Tag          Type              Name/Value
5   0x0000000000000001 (NEEDED)      Shared library: [libgcc_s.so.1]
6   0x0000000000000001 (NEEDED)      Shared library: [libc.so.6]
7   0x000000000000000c (INIT)        0x120003010
8   0x000000000000000d (FINI)        0x120010e80
9   0x0000000000000004 (HASH)        0x120000978
10  0x0000000000000005 (STRTAB)       0x120002470
11 ...

```

可以看到程序依赖libgcc_s.so.1和libc.so.6两个动态链接库。

```

1 $readelf -l mount
2
3 Elf file type is EXEC (Executable file)
4 Entry point 0x1200030a0
5 There are 7 program headers, starting at offset 64
6
7 Program Headers:
8   Type           Offset             VirtAddr           PhysAddr
9                   FileSiz             MemSiz             Flags Align
10  PHDR            0x0000000000000040 0x0000000012000040 0x0000000012000040
11                  0x0000000000000188 0x0000000000000188 R E 8
12  INTERP          0x000000000000136e0 0x000000001200136e0 0x000000001200136e0
13                  0x000000000000000f 0x000000000000000f R 1
14    [Requesting program interpreter: /lib64/ld.so.1]
15  LOAD            0x0000000000000000 0x00000000120000000 0x00000000120000000
16                  0x00000000000013710 0x00000000000013710 R E 10000
17  LOAD            0x00000000000013710 0x00000000120023710 0x00000000120023710
18                  0x0000000000001e68 0x00000000000003040 RW 10000
19  DYNAMIC          0x00000000000007b8 0x000000001200007b8 0x000000001200007b8
20                  0x00000000000001c0 0x00000000000001c0 RWE 8
21  NOTE            0x000000000000136f0 0x000000001200136f0 0x000000001200136f0
22                  0x0000000000000020 0x0000000000000020 R 4
23  NULL            0x0000000000000000 0x0000000000000000 0x0000000000000000
24                  0x0000000000000000 0x0000000000000000 8

```

可以看出还依赖/lib64/ld.so.1。

```

1 $readelf --dyn-syms mount
2
3 Symbol table '.dynsym' contains 218 entries:
4   Num:   Value              Size Type  Bind  Vis      Ndx Name
5   0: 0000000000000000 0 NOTYPE LOCAL DEFAULT UND
6   1: 0000000000000001 0 SECTION GLOBAL DEFAULT ABS _DYNAMIC_LINKING
7   2: 00000000120010ed0 4 OBJECT GLOBAL DEFAULT 12 _IO_stdin_used
8   3: 00000000120023730 0 OBJECT LOCAL HIDDEN 17 __DTOR_END__
9   4: 00000000120024e50 0 OBJECT GLOBAL DEFAULT 21 __RLD_MAP
10  5: 0000000000000000 0 OBJECT GLOBAL DEFAULT UND __environ@GLIBC_2.0 (2)
11  6: 00000000120010e60 0 FUNC GLOBAL DEFAULT UND geteuid@GLIBC_2.0 (2)
12  7: 000000001200077c8 188 FUNC GLOBAL DEFAULT 9 getfs_by_uuid
13  8: 00000000120009cc4 340 FUNC GLOBAL DEFAULT 9 xstrconcat4
14  9: 0000000012000a788 768 FUNC GLOBAL DEFAULT 9 fsprobe_procfsloop_mount
15 10: 00000000120003010 0 FUNC GLOBAL DEFAULT 8 _init

```

```

16 11: 0000000120005e18 3316 FUNC GLOBAL DEFAULT 9 main
17 12: 0000000120010e50 0 FUNC GLOBAL DEFAULT UND getgid@GLIBC_2.0 (2)
18 13: 0000000120010e40 0 FUNC GLOBAL DEFAULT UND fchmod@GLIBC_2.0 (2)
19 14: 0000000120009f30 140 FUNC GLOBAL DEFAULT 9 die
20 15: 0000000120010e30 0 FUNC GLOBAL DEFAULT UND exit@GLIBC_2.0 (2)

```

可以看到依赖的函数glibc至少什么版本。

2 解决办法

解决找不到动态链接看的办法有静态编译或拷贝对应动态链接库。

2.1 静态编译

gcc加-static参数

```

1 mipsel-linux-gcc -static ...
2 make CROSS_COMPILE=mipsel-linux- CC='mipsel-linux-gcc -static'

```

2.2 从工具链里面拷贝需要的动态链接库

附件中的ldd-pack1利用readelf得到程序需要的动态链接库，然后到用户提供的指定的几个目录里面搜索动态链接库和动态链接库又依赖的动态链接库，最后将这些动态链接库打包到/tmp/1.tar里面。这样只需要到ramdisk目录里面解压/tmp/1.tar则需要的动态链接库就安装好了。ldd-pack1用法如下：

```
1 perl ldd-pack1 elffile searchdir1 searchdir2 ...
```

其中elffile就是需要查找依赖动态链接库的程序，searchdir1,searchdir2 ... 是指定的搜索目录。

如果要到工具链目录里面搜索，如/opt/gcc-4.3-ls232/中，写法如下

```

1 rm -f /tmp/1.tar
2 cd /opt/gcc-4.3-ls232/sysroot/
3 perl ldd-pack1 elffile lib/ usr/lib/
4 cd /mnt/ramdisk
5 tar xvf /tmp/1.tar

```

cd /opt/gcc-4.3-ls232/sysroot/的目的是打包出的1.tar的目录是lib和usr/lib，正好可以直接解压到ramdisk里面。

2.3 单独的动态链接库

如果只有某个程序少动态链接库，且库的版本和系统的有冲突，如libc的版本不同等。这个时候直接覆盖系统库代价太大。比较合适的办法是给这个程序单独提供一套动态链接库，然后指定应用程序使用单独提供的这套动态链接库。上节里面的ldd-pack1程序在打包的时候会产一个runme文件，如elffile是/bin/bash的时候，runme内容如下：

```

1 D=$(dirname $0)
2 $D/lib/ld-linux.so.2 --library-path $D/lib:$D/usr/lib /bin/bash $@

```

上面脚本的目的就是当动态库不是系统目录的时候，通过这个脚本来调用elf文件使用单独提供的动态链接库。也就是说/tmp/1.tar不要在ramdisk目录根解压，而是建立一个子目录解压

```

1 mkdir ramdisk/root/newbin
2 cd ramdisk/root/newbin
3 tar xvf 1.tar
4 cd -

```

```
5 #注意runme内容的elf文件路径要正确
6 ln -sf root/newbin/runme ramdisk/bin/newbash
```

3 编译busybox

龙芯给客户提供的ramdisk文件本身是由busybox编译出来的，如果里面的动态库版本不匹配，客户可以自己编译，方法如下：

1. 下载busybox

```
1 wget http://busybox.net/downloads/busybox-1.18.5.tar.bz2
2 tar jxvf busybox-1.18.5.tar.bz2
3 cd busybox-1.18.5
```

2. 配置busybox

```
1 make menuconfig
2 Busybox Settings --->
3 Build Options --->
4 [*] Build BusyBox as a static binary (no shared libs)
5 [*] Build with Large File Support (for accessing files > 2 GB)
```

3. 编译安装busbox

```
1 make
2 make install
3 cd _install
4 mkdir dev
5 cp -a /dev/null dev
6 cp -a /dev/console dev
7 cp -a /dev/tty* dev
8 mkdir etc
9 cp ../examples/inittab etc
```