

RPATH和RUNPATH区别



c4d2c002d954 [关注](#)

2021.05.07 09:28:09 字数 482 阅读 819

RPATH和RUNPATH都可以用来在运行时搜索动态库。下面用一个简单的例子说明二者的区别。

一个小工程中，有1个头文件和3个源文件。

头文件sub.h中的内容如下：

```
#ifndef TESTRPATH_SUB_H
#define TESTRPATH_SUB_H

void f1(void);

void f2(void);

#endif
```

源文件a.c中的内容如下：

```
#include "sub.h"

void f1(void)
{
}
```

源文件b.c中的内容如下：

```
#include "sub.h"

void f2(void)
{
    f1();
}
```

源文件main.c中的内容如下：

```
#include "sub.h"

int main(void)
{
    f2();
}
```

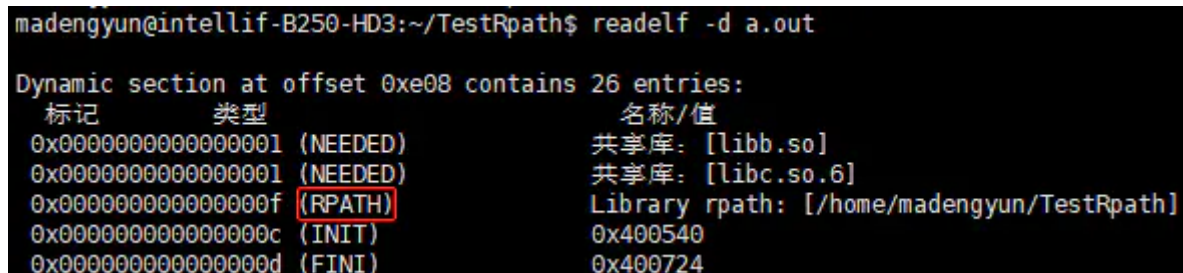
在Ubuntu16.04 + gcc5.4中编译，命令及结果如下：

```
madengyun@intellif-B250-HD3:~/TestRpath$ gcc a.c -fPIC -shared -o liba.so
madengyun@intellif-B250-HD3:~/TestRpath$ gcc b.c -fPIC -shared -L$PWD -la -o libb.so
madengyun@intellif-B250-HD3:~/TestRpath$ gcc main.c -L$PWD -Wl,-rpath,$PWD -lb
madengyun@intellif-B250-HD3:~/TestRpath$ ldd a.out
    linux-vdso.so.1 => (0x00007ffe00568000)
    libb.so => /home/madengyun/TestRpath/libb.so (0x00007f8baa94c000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f8baa582000)
    liba.so => /home/madengyun/TestRpath/liba.so (0x00007f8baa380000)
```

在Ubuntu18.10 + gcc5.4中编译，命令及结果如下：

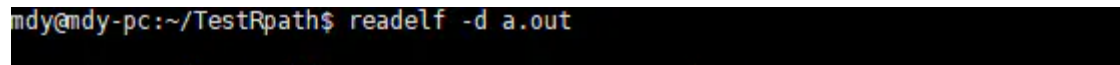
```
mdy@mdy-pc:~/TestRpath$ gcc a.c -fPIC -shared -o liba.so
mdy@mdy-pc:~/TestRpath$ gcc b.c -fPIC -shared -L$PWD -la -o libb.so
mdy@mdy-pc:~/TestRpath$ gcc main.c -L$PWD -Wl,-rpath,$PWD -lb
mdy@mdy-pc:~/TestRpath$ ldd a.out
linux-vdso.so.1 (0x00007fff635a1000)
libb.so => /home/mdy/TestRpath/libb.so (0x00007f59d326f000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f59d2e7e000)
/lib64/ld-linux-x86-64.so.2 (0x00007f59d3673000)
liba.so => not found
```

在Ubuntu18.10中编译生成的app，ldd显示找不到liba.so。但其实liba.so和libb.so在同一目录下，ldd可以找到libb.so，却找不到liba.so。用readelf -d分析两种环境下生成的app，可以看出Ubuntu16.10中生成的是RPATH，而Ubuntu18.10中生成的是RUNPATH。



```
madengyun@intellif-B250-HD3:~/TestRpath$ readelf -d a.out
Dynamic section at offset 0xe08 contains 26 entries:
 标记      类型      名称/值
0x0000000000000001 (NEEDED)   共享库: [libb.so]
0x0000000000000001 (NEEDED)   共享库: [libc.so.6]
0x000000000000000f (RPATH)    Library rpath: [/home/madengyun/TestRpath]
0x000000000000000c (INIT)     0x400540
0x000000000000000d (FINI)     0x400724
```

image.png



```
andy@andy-pc:~/TestRpath$ readelf -d a.out
```



image.png

在这个例子中，RPATH和RUNPATH的区别可用下面两段话解释：

1. 来自ld.so(8) — Linux manual page:

Using the directories specified in the DT_RUNPATH dynamic section attribute of the binary if present. Such directories are searched only to find those objects required by DT_NEEDED (direct dependencies) entries and do not apply to those objects' children, which must themselves have their own DT_RUNPATH entries. This is unlike DT_RPATH, which is applied to searches for all children in the dependency tree.

2. 来自Stack Overflow

The ld dynamic linker does not search DT_RUNPATH locations for transitive dependencies, unlike DT_RPATH。

简单地说，在搜索app的间接依赖库时，RPATH起作用，但RUNPATH不起作用。在使用RUNPATH的情况下，很可能还要再配合LD_LIBRARY_PATH一块使用。

其他问题：

1. app中默认使用RPATH还是RUNPATH是由什么决定的？

目前看是跟Linux的发行版本有关。例如，在Fedora34 + gcc10.2生成的也是RPATH。

2. 如何控制生成RPATH还是RUNPATH？

链接时使用--enable-new-dtags可以固定生成RUNPATH，使用--disable-new-dtags可以固定生成RPATH。