Linux下的lib文件故障解决实例

说到这个LIB文件，先从一个小故障说起。某日开发说，一台测试用虚机可以PING通SSH不能连了。运维同学就赶紧去查，SSHD_CONFIG配置文件都正确啊，一点错误都没有，那为什么呢？

测试下，不管连自己还是其他机，都是报错

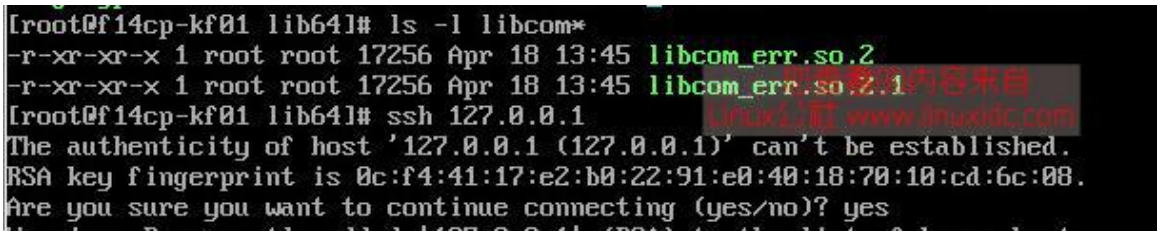
这里注意看，提示你有个 libcom_err.so.2 共享库文件找不到。

询问开发，才了解他们测试一个软件，意外删除了某个库文件。那么在正常的相同虚机的机器查看下，再和出错的虚机比对下，发现

少了 2 个库文件

挂载系统光盘或从正常的虚机上把这个两个文件拷贝过来，放到 lib64 下就可以了

再试正常了



这个小故障很容易解决，那么你怎么理解 Linux 中的库文件呢？学习下也不误工作。

Linux 下的库文件分为共享库和静态库两大类，它们两者的差别仅在程序执行时所需的代码是在运行时动态加载的，还是在编译时静态加载的。

Linux 的库一般在/lib 或/usr/lib 目 录下，如果是 64 位的系统则会有 lib64 目录。lib 是库（Library）的英文缩写，它主要存放系统的链接库文件，没有该目录则系统就无法正常运行。/lib 目录中存储着程序运行时使用的共享库。通过共享库，许多程序可以重复使用相同的代码，并且这些库可以存储在一个公共的位置上，因此能减小运行程序的大小。这个目录包含程序在链接时使用的 各种库。

库的知识

1.库的命名

库的命名比较简单，第一个特点是所有的库以 lib 开头，GCC 命令在在－l 选项所指定的文件名前会自动加入 lib。

第二个特点文件名以.a 结尾的库是静态库。

第三个特点文件名是.so 的库为共享库(共享库是在运行的时候动态加载的)。默认情况下，GCC 在链接时优先使用共享库，只有当共享库不存在时才考虑使用静态库。

2、库的编号

库的编号格式如下：

library_name .major.num .minor_.min .pathch_num
例如，笔者 Red Hat Linux 9.0 的 GUN 数据库是 libgdbm.so.0.0.2，详细表述如下：

◆library_name 是 libc.so(标准 C 库)；

◆major_num 是 2（主版本号）；

◆minor_.min 是 0（次版本号）；

◆pathch_num 是 0（补丁级别号又称发行号）。

3、库的操作命令

Linux 库操作可以使用命令完成，目前常用的命令是 ldd 和 ldconfig。

ldd 是 Library Dependency Display 缩写，它的作用是显示一个可执行程序必须使用的共享库 。

（1）命令格式

ldd [选项] 文件名

（2）主要参数

－d 执行重定位并报告丢失的函数。

－r 执行对函数和数据对象的重定位，并报告丢失的函数和数据对象。

（3）应用举例

比如查询 Perl 语言有哪些共享库，则可以首先使用 find 命令查询这个程序的绝对路径，然后使用 ldd 命令：

＃find -name perl
ldd /usr/bin/perl

$ ldd test
执行 test，可以看到它是如何调用动态库中的函数的。

2.ldconfig

ldconfig 命令的作用是决定位于目录/usr/lib 和/lib 下的共享库所需的运行链接。这些链接保存在的 Libs 保存在/et/ld.so.conf 文件中。搜 索出可共享的动态链接库(格式如前介绍,lib*.so*),进而创建出动态装入程序(ld.so)所需的链接和缓存文件。缓存文件默认为/etc /ld.so.cache,此文件保存已排好序的动态链接库名字列表。

（1）命令格式

ldconfig [选项] [libs]

（2）主要选项

-v 或--verbose ldconfig 将显示正在扫描的目录、搜索到的动态链接库，以及它所创建的连接的名字。

-f CONF 指定动态链接库的配置文件为 CONF，系统默认为/etc/ld.so.conf。

-C CACHE 指定生成的缓存文件为 CACHE，系统默认的是/etc/ld.so.cache,文件存放已排好序的可共享的动态链接库的列表。

-p 或--print-cache 让 ldconfig 打印出当前缓存文件所保存的所有共享库的名字。

-r ROOT 改变应用程序的根目录为 ROOT。

－n ldconfig 仅扫描命令行指定的目录，不扫描默认目录(/lib、/usr/lib),也不扫描配置文件/etc/ld.so.conf 所列的目录。

运行没有选项的 ldconfig 命令时，用于更新高速缓冲文件。这个命令主要用于高速缓冲 DNS 服务器(Caching DNS Server)。高速缓冲 DNS 服务器的原理是提供查询的历史记录，并且利用这些记录来提高查询的效率。

当某个查询是第一次被发送到高速缓冲 DNS 服务器时，高速缓冲 DNS 服务器就将此查询的整个过程记录下来，在一定的时期内用它来回答所有相同的查询，从而减少整个 DNS 系统的负担并且提高查询速度。

（3）应用实例

如果用户想知道系统中有哪些动态链接库，或者想知道系统中有没有某个动态链接库时，可用-p 选项让 ldconfig 输出缓存文件中的动态链接库列表，从而查询得到。 例如：

```
ldconfig -p
998 libs found in cache `/etc/ld.so.cache'
libzvt.so.2 (libc6) => /usr/lib/libzvt.so.2
libzvt.so (libc6) => /usr/lib/libzvt.so
……
```

补充：

静态链接库*.a 的编译和使用
创建.a 库文件和.o 库文件：

```
[yufei@localhost perl_c2]$ pwd
/home/yufei/perl_c2

[yufei@localhost perl_c2]$ cat mylib.c
#include <stdio.h>
#include <string.h>
void hello(){
      printf("success call from perl to c library\n");
}
[yufei@localhost perl_c2]$ cat mylib.h
extern void hello();


[yufei@localhost perl_c2]$ gcc -c mylib.c
[yufei@localhost perl_c2]$ dir
mylib.c  mylib.h  mylib.o
[yufei@localhost perl_c2]$ ar -r mylib.a mylib.o
ar: 正在创建 mylib.a
[yufei@localhost perl_c2]$ dir
mylib.a  mylib.c mylib.h  mylib.o
```

*.a 的使用方法

最简单的是直接把.a 当成一个普通源代码编译进来.

```
gcc  main.cpp  ./lib/libInfo.a  -o  exec
```

动态链接库*.so 的编译与使用- -

动态库*.so 在 linux 下用 c 和 c++编程时经常会碰到，这里做个笔记，也为其它正为动态库链接库而苦恼的兄弟们提供一点帮助。
1、动态库的编译

下面通过一个例子来介绍如何生成一个动态库。这里有一个头文件：so_test.h，三个.c 文件：test_a.c、test_b.c、test_c.c，我们将这几个文件编译成一个动态库：libtest.so。

so_test.h：

```
#include <stdio.h>
#include <stdlib.h>
void test_a();
void test_b();
void test_c();
```

test_a.c：

```
#include "so_test.h"
void test_a()
{
   printf("this is intest_a...\n");
}
```

test_b.c：

```
#include "so_test.h"
void test_b()
{
   printf("this is intest_b...\n");
}
```

test_c.c：

```
#include "so_test.h"
void test_c()
{
   printf("this is intest_c...\n");
}
```

将这几个文件编译成一个动态库：libtest.so

```
$ gcc test_a.c test_b.c test_c.c -fPIC -shared -o libtest.so
```

2、动态库的链接

在 1、中，我们已经成功生成了一个自己的动态链接库 libtest.so，下面我们通过一个程序来调用这个库里的函数。程序的源文件为：test.c。
test.c：

```
#include "so_test.h"
int main()
{
   test_a();
   test_b();
   test_c();
   return 0;

}
```

l 将 test.c 与动态库 libtest.so 链接生成执行文件 test：

```
$ gcc test.c -L. -l test -o test
```

l 测试是否动态连接，如果列出 libtest.so，那么应该是连接正常了

```
$ ldd test
```

l 执行 test，可以看到它是如何调用动态库中的函数的。

总结：

1、共享库特别适合多个程序共享代码，升级程序部分功能模块，实现程序“插件”功能的情况；
而静态库是一劳永逸，编译后不需要带一堆库文件跑，而且不管放置到哪里都可正常运行。

2、当搜索的库文件目录下同时存在该库的静态版本和共享版本时，链接器优先使用共享版本.so，此时你可以使用-static 链接选项指定链接静态版本.a。
3、动态库可以导出两个特殊的函数：_init 和_fini，前者在动态库被加载后调用，后者在动态库被卸载前调用，我们可以使用这两个函数做些特别的工作。需要注意的是：在定义这两个函数后编译时，需要使用 -nostartfiles 选项，否则编译器报重复定义错误。
4、ldd 命令用来查看程序所依赖的共享库，同时也方便我们判断共享库是否被找到；
nm 命令查看 obj 文件（.so 也是一个 obj）中的标识（函数、变量）。