同济大学计算机网络
实验报告

姓名：涂远鹏-1652262
题目：动态链接库的编译与使用

1.Linux 下的动态链接库

(1)动态链接库的基本概念

答：动态链接库（英语：Dynamic-link library，缩写为 DLL）是微软公司在微软视窗操作系统中实现共享函数库概念的一种实现方式。这些库函数的扩展名是.DLL、.OCX（包含 ActiveX 控制的库）或者.DRV（旧式的系统驱动程序)。

所谓动态链接，就是把一些经常会共用的代码（静态链接的 OBJ 程序库）制作成 DLL 档，当可执行文件调用到 DLL 档内的函数时，Windows 操作系统才会把 DLL 档加载内存内，DLL 档本身的结构就是可执行文件，当程序有需求时函数才进行链接。通过动态链接方式，内存浪费的情形将可大幅降低。静态链接库则是直接链接到可执行文件.

(2)动态链接库的好处

答: 使用动态链接库可以节省内存和减少交换操作。很多进程可以同时使用一个 DLL，在内存中共享该 DLL 的一个副本。 相反，对于每个用静态链接库生成的应用程序，Windows 必须在内存中加载库代码的一个副本。使用动态链接库可以节省磁盘空间。 许多应用程序可在磁盘上共享 DLL 的一个副本。 相反，每个用静态链接库生成的应用程序均具有作为单独的副本链接到其可执行图像中的库代码。升级到 DLL 更为容易。 当 DLL 中的函数发生更改时，只要函数的参数和返回值没有更改，就不需重新编译或重新链接使用它们的应用程序。 相反，静态链接的对象代码要求在函数更改时重新链接应用程序。

(3)动态链接库的编译方法

答:

首先将.c 文件编译成二进制文件：# gcc -c hello.c

随后将.o 文件转化为静态链接库的的.a 文件：#ar rcs libmyhello.a hello.o

(4)动态链接库的使用实例

答：实例:用 vs2015 等开发工具建立工程时，可以在应用程序设置中的应用程序类型设置 DLL 表示建立动态链接库，之后可以在 DLL 项目中添加导出函数以便在编程时在附加的动态库中调用。

2. 按要求写出下列几种常用情况的动态链接库的测试样例

2.1 创建文件夹：

2.2 在子目录 01 下建立两个源程序文件 test1.c,test2.c（分别打印自己的学号）并写出满足要求的 makefile 文件：

(1)编写的 makefile 文件内容如下:

```
[root@RHEL74-SVR 1652262-990101]# cd /home/1652262-000105
[root@RHEL74-SVR 1652262-000105]# cd 01
[root@RHEL74-SVR 01]# cat makefile
test2:test1.so test1.o
        cc -o test2 test2.c test1.so
test1.so:test1.c
        cc -c -fpic test1.c
        cc -shared -fpic -o test1.so test1.o
clean:
        rm test2 *.o *.so

[root@RHEL74-SVR 01]#
```

(2)在这里我把文件名改成了 libtest1.so

这样-ltest1 就可以找到，有可能连接的文件找不到

```
[root@RHEL74-SVR 01]# ldconfig
[root@RHEL74-SVR 01]# ./test2
./test2: error while loading shared libraries: test1.so: cannot open shared object file: N
o such file or directory
[root@RHEL74-SVR 01]#
```

此时暂时给一个 export，就可以成功运行出预期结果

```
[root@RHEL74-SVR 01]# export LD_LIBRARY_PATH=/usr/lib:$LD_LIBRARY_PATH
[root@RHEL74-SVR 01]# ./test2
1652262涂远鹏[root@RHEL74-SVR 01]#
```

修改 makefile 使其不用加临时输出也可以运行:

```
 test2.c  9L, 106C written
[root@RHEL74-SVR 01]# vi makefile
test2:libtest1.so test1.o
        cc -o test2 test2.c -L. -ltest1 -Wl,-rpath .
libtest1.so:test1.c
        cc -c -fpic test1.c
        cc -shared -fpic -o libtest1.so test1.o
clean:
        rm test2 *.o *.so
```

显示修改结果:

```
[root@RHEL74-SVR 01]# make
cc -o test2 test2.c -L. -ltest1 -Wl,-rpath .
[root@RHEL74-SVR 01]# make clean
rm test2 *.o *.so
[root@RHEL74-SVR 01]# make
cc -c -fpic test1.c
cc -shared -fpic -o libtest1.so test1.o
cc -o test2 test2.c -L. -ltest1 -Wl,-rpath .
[root@RHEL74-SVR 01]# ./test2
1652262涂远鹏[root@RHEL74-SVR 01]#
```

(3)修改了 test1.c 文件 改了其输出 并使其重新输出.so 文件覆盖之前的文件

运行 test2 可执行文件达到了预期效果

```
"test1.c" 7L, 87C written
[root@RHEL74-SVR 01]# gcc -c -fpic test1.c
[root@RHEL74-SVR 01]# gcc -shared -fpic -o test1.so test1.o
[root@RHEL74-SVR 01]# ./test2
1652262涂远鹏[root@RHEL74-SVR 01]#
```

2.3 在子目录 02 下建立两个源程序文件 test1.cpp,test2.cpp（分别打印自己的学号）并写出满足要求的 makefile 文件：

(1)将上一题的 makefile 文件中的 gcc 改为 g++　.c 改成.cpp 即可

```
[root@RHEL74-SVR 02]# vi makefile
test2:libtest1.so test1.o
        g++ -g -Wall -o  test2 test2.cpp -L. -ltest1 -Wl,-rpath .
libtest1.so:test1.cpp
        g++ -c -fpic test1.cpp
        g++ -shared -fpic -o libtest1.so test1.o
clean:
        rm test2 *.o *.so
```

(2)运行可执行文件 test2 达到预期效果

```
[root@RHEL74-SVR 02]# make
g++ -c -fpic test1.cpp
g++ -shared -fpic -o libtest1.so test1.o
g++ -g -Wall -o  test2 test2.cpp -L. -ltest1 -Wl,-rpath .
[root@RHEL74-SVR 02]# ./test2
1652262涂远鹏[root@RHEL74-SVR 02]# makeclean
```

(3)修改 test1.cpp 的学号,重新编译并修改动态库文件,运行 test2 达到预期效果

```
"test1.cpp" 8L, 110C written
[root@RHEL74-SVR 02]# g++ -cpp -fpic test1.cpp
/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64/crt1.o：在函数‘_start’中：
(.text+0x20)：对‘main’未定义的引用
collect2: 错误：ld 返回 1
[root@RHEL74-SVR 02]# g++ -c -fpic test1.cpp
[root@RHEL74-SVR 02]# g++ -shared -fpic -o test1.so test1.o
[root@RHEL74-SVR 02]# ./test2
1652262涂远鹏[root@RHEL74-SVR 02]#
```

2.4 在 1652262-000105 目录下写一个满足要求的 makefile 文件：

(1)编写的 makefile 文件内容如下，与之前编写的内容一样:

```
[root@RHEL74-SVR test]# cd ..
[root@RHEL74-SVR home]# vi makefile
SUB = `ls -d */`

subdirs:
        for dir in $(SUB); do \
$(MAKE) -C $$dir; \
done

clean:
        for dir in $(SUB); do \
$(MAKE) clean -C $$dir; \
done
```

(2)执行 make 操作测试结果如下，结果显示正确:

```
"makefile" 10L, 137C written
[root@RHEL74-SVR 1652262-000105]# make
for dir in `ls -d */`; do \
make -C $dir; \
done
make[1]: 进入目录“/home/1652262-000105/01”
cc -o test2 test2.c test1.so
make[1]: 离开目录“/home/1652262-000105/01”
make[1]: 进入目录“/home/1652262-000105/02”
g++ -o test2 test2.cpp test1.so
make[1]: 离开目录“/home/1652262-000105/02”
[root@RHEL74-SVR 1652262-000105]# cd 01
[root@RHEL74-SVR 01]# ll
总用量 36
-rw-r--r--. 1 root root  157 9月  18 21:34 makefile
-rw-r--r--. 1 root root   87 9月  18 21:41 test1.c
-rw-r--r--. 1 root root 1552 9月  18 21:42 test1.o
-rwxr-xr-x. 1 root root 8104 9月  18 21:42 test1.so
-rwxr-xr-x. 1 root root 8480 9月  30 16:37 test2
-rw-r--r--. 1 root root  106 9月  17 21:27 test2.c
[root@RHEL74-SVR 01]# cd ..
[root@RHEL74-SVR 1652262-000105]# cd 02
[root@RHEL74-SVR 02]# ll
总用量 40
-rw-r--r--. 1 root root  187 9月  18 21:45 makefile
-rw-r--r--. 1 root root  110 9月  18 21:46 test1.cpp
-rw-r--r--. 1 root root 2536 9月  18 21:46 test1.o
-rwxr-xr-x. 1 root root 8544 9月  18 21:47 test1.so
-rwxr-xr-x. 1 root root 8928 9月  30 16:37 test2
-rw-r--r--. 1 root root  127 9月  17 21:27 test2.cpp
[root@RHEL74-SVR 02]#
```

(3)执行 make clean 操作测试结果如下，结果显示正确：

```
"makefile" 10L, 140C written
[root@RHEL74-SVR 1652262-000105]# make clean
for dir in `ls -d */`; do \
make clean -C $dir; \
done
make[1]: 进入目录“/home/1652262-000105/01”
rm test2 *.o *.so
make[1]: 离开目录“/home/1652262-000105/01”
make[1]: 进入目录“/home/1652262-000105/02”
rm test2 *.o *.so
make[1]: 离开目录“/home/1652262-000105/02”
[root@RHEL74-SVR 1652262-000105]# cd 01
[root@RHEL74-SVR 01]# ll
总用量 12
-rw-r--r--. 1 root root 157 9月  18 21:34 makefile
-rw-r--r--. 1 root root  87 9月  18 21:41 test1.c
-rw-r--r--. 1 root root 106 9月  17 21:27 test2.c
[root@RHEL74-SVR 01]# cd ..
[root@RHEL74-SVR 1652262-000105]# cd 02
[root@RHEL74-SVR 02]# ll
总用量 12
-rw-r--r--. 1 root root 187 9月  18 21:45 makefile
-rw-r--r--. 1 root root 110 9月  18 21:46 test1.cpp
-rw-r--r--. 1 root root 127 9月  17 21:27 test2.cpp
[root@RHEL74-SVR 02]#
```