

第 00 章作业 - Linux 知识补充 - 程序的静态与动态编译

1652817 钟钰琛 计算机科学与技术

2018 年 10 月 10 日

1 Linux 下的动态编译

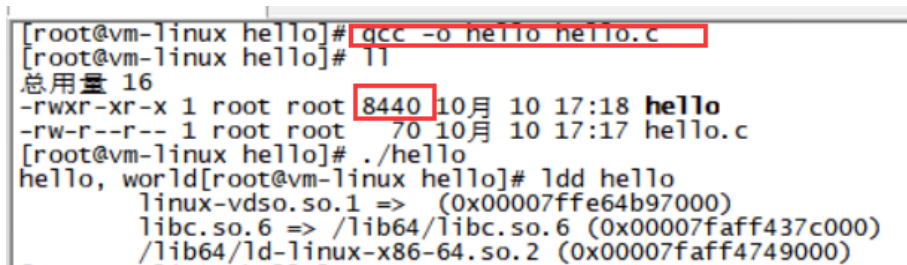
1. 动态编译指的是延迟编译直到程序被加载或者运行. 在许多情况下, 只有编译器后端(生成目标代码和优化)才会被延迟. 例如: JIT: just-in-time compilation

这是动态编译的原始意思. 但 c/c++ 是 ahead of time(AOT) 编译, 所以我觉得这里动态编译指的是动态链接?

动态链接是指编译的时候使用动态链接库. 但动态链接库不会直接放入可执行文件中, 而是在需要的时候被加载. 所以执行的时候必须要有依赖的动态链接库.

2. 直接 gcc 即可

```
gcc -o hello hello.c
```



```
[root@vm-linux hello]# gcc -o hello hello.c
[root@vm-linux hello]# ll
总用量 16
-rwxr-xr-x 1 root root 8440 10月 10 17:18 hello
-rw-r--r-- 1 root root 70 10月 10 17:17 hello.c
[root@vm-linux hello]# ./hello
hello, world[root@vm-linux hello]# ldd hello
        linux-vdso.so.1 => (0x00007ffe64b97000)
        libc.so.6 => /lib64/libc.so.6 (0x00007faff437c000)
        /lib64/ld-linux-x86-64.so.2 (0x00007faff4749000)
```

图 1: hello.c 动态编译

大小是 8440

3. 直接 g++ 即可

```
g++ -o hello hello.cpp
```

```
[root@vm-linux hello]# g++ -o hello hello.cpp
[root@vm-linux hello]# ll
总用量 20
-rwxr-xr-x 1 root root 8888 10月 10 17:22 hello
-rw-r--r-- 1 root root 70 10月 10 17:17 hello.c
-rw-r--r-- 1 root root 91 10月 10 17:22 hello.cpp
[root@vm-linux hello]# ./hello
hello, world[root@vm-linux hello]# ldd hello
linux-vdso.so.1 => (0x00007fff18bf9000)
libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f10b9d23000)
libm.so.6 => /lib64/libm.so.6 (0x00007f10b9a21000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f10b980b000)
libc.so.6 => /lib64/libc.so.6 (0x00007f10b943e000)
/lib64/ld-linux-x86-64.so.2 (0x00007f10ba02a000)
```

图 2: hello.cpp 动态编译

大小是 8888

注意到, cpp 生成的可执行文件, 相比 c 生成的, 依赖的动态库多了几个, 导致文件字节数变大.

4. mysql_demo.cpp

```
192.168.159.22 x
[root@vm-linux ~]# g++ demo.cpp -o demo -I /usr/include/mysql/ -L /usr/lib64/mysql/ -lmys
qlclient
[root@vm-linux ~]# ll
总用量 32
drwxr-xr-x 5 root root 52 10月 10 01:27 1652817-000103
drwxr-xr-x 2 root root 94 10月 10 02:52 1652817-000104
-rwxrwxrwx. 1 root root 1410 9月 22 03:19 anaconda-ks.cfg
-rwxr-xr-x 1 root root 14192 10月 10 02:54 demo
-rwxrwxrwx. 1 root root 2223 9月 22 06:00 demo.cpp
drwxrwxrwx. 2 root root 4096 9月 22 06:17 doc
-rw-r--r-- 1 root root 76 9月 28 23:41 student.conf
[root@vm-linux ~]#
```

图 3: demo.cpp

这时候用 ldd 查看依赖的动态库, 有一大串, 导致文件大小猛增.

查找某个可执行文件所依赖的动态链接库: 用 ldd 命令

```
[root@vm-linux ~]# ldd demo
linux-vdso.so.1 => (0x00007fff6c565000)
libmysqlclient.so.18 => /usr/lib64/mysql/libmysqlclient.so.18 (0x00007f526104a000)
)
libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f5260d42000)
libm.so.6 => /lib64/libm.so.6 (0x00007f5260a40000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f526082a000)
libc.so.6 => /lib64/libc.so.6 (0x00007f526045d000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f5260241000)
libz.so.1 => /lib64/libz.so.1 (0x00007f526002b000)
libssl.so.10 => /lib64/libssl.so.10 (0x00007f525fdb9000)
libcrypto.so.10 => /lib64/libcrypto.so.10 (0x00007f525f958000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f525f754000)
/lib64/ld-linux-x86-64.so.2 (0x00007f526154a000)
libgssapi_krb5.so.2 => /lib64/libgssapi_krb5.so.2 (0x00007f525f507000)
libkrb5.so.3 => /lib64/libkrb5.so.3 (0x00007f525f21f000)
libcom_err.so.2 => /lib64/libcom_err.so.2 (0x00007f525f01b000)
libk5crypto.so.3 => /lib64/libk5crypto.so.3 (0x00007f525ede8000)
libkrb5support.so.0 => /lib64/libkrb5support.so.0 (0x00007f525ebda000)
libkeyutils.so.1 => /lib64/libkeyutils.so.1 (0x00007f525e9d6000)
libresolv.so.2 => /lib64/libresolv.so.2 (0x00007f525e7bd000)
libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f525e596000)
libpcre.so.1 => /lib64/libpcre.so.1 (0x00007f525e334000)
[root@vm-linux ~]#
```

图 4: ldd 命令

2 Linux 下的 gcc 静态编译

1. 静态编译就是编译器在编译可执行文件的时候, 将可执行文件需要调用的对应静态库 (.a) 中的部分提取出来, 链接到可执行文件中去, 使可执行文件在运行的时候不依赖于动态链接库.

2. 静态编译 hello.c

需要安装 glibc-static, 安装步骤在下文

输入命令

```
gcc -static -o hello hello.c
```

```
[root@vm-linux hello]# gcc -static -o hello hello.c
[root@vm-linux hello]# ./hello
hello, world[root@vm-linux hello]# ll
总用量 852
-rwxr-xr-x 1 root root 861072 10月 10 17:43 hello
-rw-r--r-- 1 root root 70 10月 10 17:17 hello.c
-rw-r--r-- 1 root root 91 10月 10 17:22 hello.cpp
[root@vm-linux hello]# ldd hello
不是动态可执行文件
```

图 5: hello.c 静态编译

大小达到惊人的 861072，比其动态编译的 8440 大了 100 余倍！这是因为把依赖的库都链接进去了，所以才会这么大。

3 Linux 下的 c++/g++ 静态编译

需要 libstdc++-static, 安装步骤在下文.

输入命令

```
g++ -static -o hello hello.cpp
```

```
[root@vm-linux hello]# g++ -static -o hello hello.cpp
[root@vm-linux hello]# ./hello
hello, world[root@vm-linux hello]# ll
总用量 1580
-rwxr-xr-x 1 root root 1608232 10月 10 17:48 hello
-rw-r--r-- 1 root root 70 10月 10 17:17 hello.c
-rw-r--r-- 1 root root 91 10月 10 17:22 hello.cpp
[root@vm-linux hello]# ldd hello
不是动态可执行文件
```

图 6: hello.cpp 静态编译

大小比 c 的静态编译还要大一倍. 因为链接的库更多.

4 按要求写出下列几种常用情况的静态编译测试样例

4.1 子目录 01

为了静态编译, 需要安装 glibc-static

首先从官网下载 [下载地址](#)

然后用 ftp 传到 linux 上

最后用 rpm 命令安装

```
[root@vm-linux lib]# rpm -iv glibc-static-2.17-222.el7.x86_64.rpm
警告: glibc-static-2.17-222.el7.x86_64.rpm: 头v3 RSA/SHA256 Signature, 密钥 ID f4a80eb5:
NOKEY
软件包准备中...
glibc-static-2.17-222.el7.x86_64
[root@vm-linux lib]#
```

图 7: 安装 glibc-static

makefile 如下:

```
CC=gcc

test: test.o
    $(CC) -static test.c -o test

clean:
    rm test *.o
```

```
[root@vm-linux 01]# make
gcc -c -o test.o test.c
gcc -static test.c -o test
[root@vm-linux 01]# ll
总用量 856
-rw-r--r-- 1 root root    72 10月 10 04:09 makefile
-rwxr-xr-x 1 root root 861072 10月 10 04:15 test
-rw-r--r-- 1 root root    81 10月 10 03:17 test.c
-rw-r--r-- 1 root root  1504 10月 10 04:15 test.o
[root@vm-linux 01]# ./test
1652817 钟钰琛[root@vm-linux 01]# ldd test
不是动态可执行文件
```

图 8: 静态编译 test.c

4.2 子目录 02

需要 libstdc++-static

一开始也是在官网下载然后放到 linux 上, 安装的时候发现还需要一堆依赖...

所以换 yum 直接安装. 首先把之前设置的只用 iso 作为 yum 源去掉, 让 yum 上网.

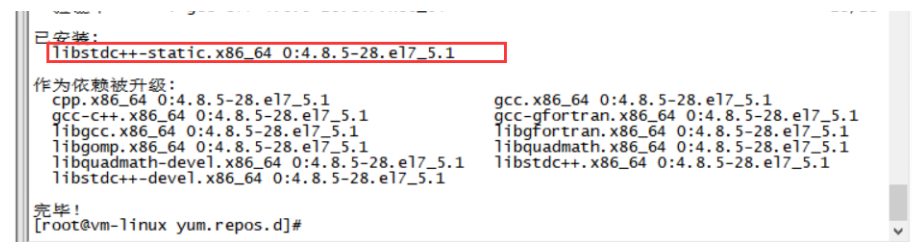
```
[root@vm-linux yum.repos.d]# ls
CentOS-Base.repo  CentOS-Debuginfo.repo  CentOS-ISO.repo  CentOS-Sources.repo
CentOS-CR.repo    CentOS-fasttrack.repo   CentOS-Media.repo  CentOS-vault.repo
```

图 9: /etc/yum.repo.d/

然后运行

```
yum clean all
```

就可以联网使用了.



```
已安装:
libstdc++-static.x86_64 0:4.8.5-28.el7_5.1

作为依赖被升级:
cpp.x86_64 0:4.8.5-28.el7_5.1          gcc.x86_64 0:4.8.5-28.el7_5.1
gcc-c++.x86_64 0:4.8.5-28.el7_5.1      gcc-gfortran.x86_64 0:4.8.5-28.el7_5.1
libgcc.x86_64 0:4.8.5-28.el7_5.1        libgfortran.x86_64 0:4.8.5-28.el7_5.1
libgomp.x86_64 0:4.8.5-28.el7_5.1        libquadmath.x86_64 0:4.8.5-28.el7_5.1
libquadmath-devel.x86_64 0:4.8.5-28.el7_5.1  libstdc++.x86_64 0:4.8.5-28.el7_5.1
libstdc++-devel.x86_64 0:4.8.5-28.el7_5.1

完毕!
[root@vm-linux yum.repos.d]#
```

图 10: 安装 libstdc++-static

makefile 如下:

```
CC=g++

test:test.cpp
$(CC) -static test.cpp -o test

clean:
rm test
```

4.3 总目录

这个 makefile 和 000103 的一模一样. 不再赘述.

```

[root@vm-linux 02]# ls
makefile test.cpp
[root@vm-linux 02]# make
g++ -static test.cpp -o test
[root@vm-linux 02]# ls
makefile test test.cpp
[root@vm-linux 02]# ./test
1652817 钟钰琛[root@vm-linux 02]# ldd test
不是动态可执行文件
[root@vm-linux 02]#

```

图 11: 静态编译 test.cpp

```

192.168.159.22 x
[root@vm-linux 1652817-000104]# make
for dir in `ls -d */`; \
do \
    make -C $dir; \
done
make[1]: 进入目录"/root/1652817-000104/01"
gcc -c -o test.o test.c
gcc -static test.c -o test
make[1]: 离开目录"/root/1652817-000104/01"
make[1]: 进入目录"/root/1652817-000104/02"
g++ -static test.cpp -o test
make[1]: 离开目录"/root/1652817-000104/02"
[root@vm-linux 1652817-000104]# ldd 01/test
不是动态可执行文件
[root@vm-linux 1652817-000104]# ldd 02/test
不是动态可执行文件
[root@vm-linux 1652817-000104]# make clean
for dir in `ls -d */`; \
do \
    make clean -C $dir; \
done
make[1]: 进入目录"/root/1652817-000104/01"
rm test *.o
make[1]: 离开目录"/root/1652817-000104/01"
make[1]: 进入目录"/root/1652817-000104/02"
rm test
make[1]: 离开目录"/root/1652817-000104/02"
[root@vm-linux 1652817-000104]# ls 01/
makefile test.c
[root@vm-linux 1652817-000104]# ls 02/
makefile test.cpp
[root@vm-linux 1652817-000104]#

```

图 12: makefile