静态库和共享库的制作和使用

一、基础：gcc 的编译过程

1、编译工具链：

1）预处理器（cpp）：头文件展开、宏替换、去掉注释，命令：gcc **–E** hello.c –o hello.i

2）编译器（gcc）：c 文件变成汇编文件，gcc **–S** hello.i –o hello.s

3）汇编器（as）：汇编文件变成二进制文件，命令：gcc **–c** hello.s –o hello.o

4）链接器（ld）：将函数库中相应的代码组合到目标文件中，命令：gcc hello.o –o hello

在这四步编译过程中，如果直接执行后面的过程则会默认调用前面的过程。也就是说，gcc hello.c –o hello 会自动调用前三步。如果直接执行 gcc hello.c 则生成的可执行文件名称默认为 a.out。

2、gcc 参数说明：

-I（大写 i）：后面跟头文件目录，在旧版本中-I 和目录直接不能有空格，新版本中加不加空格都行。

-D：后面跟宏名，在编译的所有.c 文件中加入指定的宏定义。

-O：优化程序。分为四个等级：-O0、-O1、-O2、-O3，把程序中冗余代码作出优化。

-o：指定生成文件的名字。

-Wall：输出警告信息。比如定义了一个变量但没有使用等警告信息。

-g：生成包含调试信息的可执行程序，在 gdb 调试时使用。

例：gcc hello.c –o hello –I./include –D DEBUG –O3 –Wall –g

二、静态库的制作和使用

1、命名规则：lib+库的名字+.a　　例：libMytest.a

2、制作步骤：

1）生成对应的.o 文件：gcc **–c** *.c

2）将生成的.o 文件打包成静态库：ar rcs + 静态库的名字（libMytest.a）+ 生成的所有的.o 文件

3、发布和使用静态库：

1）发布静态库和头文件。

2）使用时包含头文件并链接静态库即可。-L：后面跟链接库的目录；-l（小写 L）：后面跟链接库的名称（Mytest）。

***在链接时并不会将整个静态库组合到可执行文件中，而是以静态库中的.o 文件为单位，用到哪个组合那个。**

4、实例：

1）准备工作：

include 目录存放头文件 head.h；lib 目录存放生成的库文件；src 目录存放源文件 add.c、sub.c、mul.c、div.c；main.c 是测试程序。

//head.h

```
#ifndef _HEAD_H_
#define _HEAD_H_
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);
int div(int a, int b);
#endif
```

//main.c

```c
#include <stdio.h>
#include "head.h"

int main()
{
    int sum = add(2, 3);
    printf("sum = %d\n",sum);
    return 0;
}
```

//src 文件

```c
int div(int a, int b)
{
    return a / b;
}
```

../src/div.c

```c
int mul(int a, int b)
{
    return a * b;
}
```

../src/mul.c

```c
int sub(int a, int b)
{
    return a - b;
}
```

../src/sub.c

```c
int add(int a, int b)
{
    return a + b;
}
```

../src/add.c

2）制作静态库并将制作好的静态库放到 lib 目录下：

```
runsir@ubuntu:~/libtest/src$ ls
add.c  div.c  mul.c  sub.c
runsir@ubuntu:~/libtest/src$ gcc -c *.c
runsir@ubuntu:~/libtest/src$ ls
add.c  add.o  div.c  div.o  mul.c  mul.o  sub.c  sub.o
runsir@ubuntu:~/libtest/src$ ar rcs libMytest.a *.o
runsir@ubuntu:~/libtest/src$ ls
add.c  add.o  div.c  div.o  libMytest.a  mul.c  mul.o  sub.c  sub.o
runsir@ubuntu:~/libtest/src$ mv libMytest.a ../lib
runsir@ubuntu:~/libtest/src$ cd ../lib
runsir@ubuntu:~/libtest/lib$ ls
libMytest.a
```

3）测试：

```
runsir@ubuntu:~/libtest$ ls
include  lib  main.c  src
runsir@ubuntu:~/libtest$ gcc main.c lib/libMytest.a -o mytest -Iinclude
runsir@ubuntu:~/libtest$ ls
include  lib  main.c  mytest  src
runsir@ubuntu:~/libtest$ ./mytest
sum = 5
runsir@ubuntu:~/libtest$
```

gcc main.c lib/libMytest.a -o mytest –Iinclude

= gcc main.c -o mytest -Iinclude -L lib -l Mytest

5、优缺点：

1）优点：

①静态库中相应的代码会组合到可执行程序中，所以发布程序的时候不需要提供对应的库。

②加载库的速度快。

2）缺点：

①应用程序的体积很大。

②库发生改变，应用程序需要重新编译。

三、共享库（动态库）的制作和使用

1、命名规则：lib+库的名字+.so　　例：libMytest.so

2、制作步骤：

1）生成与位置无关的代码（.o 文件）：gcc **–fPIC** –c *.c（生产的代码中没有绝对路径，只有相对路径）

2）将生成的.o 文件打包成共享库：gcc **–shared** -o libMytest.so *.o

3、发布和使用共享库：

1）发布共享库和头文件。

2）使用时包含头文件并链接共享库。

4、实例：

1）准备工作（同静态库）

2）制作共享库并将制作好的共享库放到 lib 目录下：

```
runsir@ubuntu:~/libtest/src$ ls
add.c  div.c  mul.c  sub.c
runsir@ubuntu:~/libtest/src$ gcc -fPIC -c *.c
runsir@ubuntu:~/libtest/src$ ls
add.c  add.o  div.c  div.o  mul.c  mul.o  sub.c  sub.o
runsir@ubuntu:~/libtest/src$ gcc -shared -o libMytest.so *.o
runsir@ubuntu:~/libtest/src$ ls
add.c  add.o  div.c  div.o  libMytest.so  mul.c  mul.o  sub.c  sub.o
runsir@ubuntu:~/libtest/src$ mv libMytest.so ../lib
runsir@ubuntu:~/libtest/src$ cd ../lib/
runsir@ubuntu:~/libtest/lib$ ls
libMytest.so
```

3）测试：

```
runsir@ubuntu:~/libtest$ ls
include  lib  main.c  src
runsir@ubuntu:~/libtest$ gcc main.c lib/libMytest.so -o mytest -Iinclude
runsir@ubuntu:~/libtest$ ls
include  lib  main.c  mytest  src
runsir@ubuntu:~/libtest$ ./mytest
sum = 5
```

注：此时使用 gcc main.c -o mytest -Iinclude -Llib –lMytest 生成的文件将无法执行，可以用 ldd 命令查看可执行文件在运行中所依赖的共享库。

```
runsir@ubuntu:~/libtest$ ls
include  lib  main.c  src
runsir@ubuntu:~/libtest$ gcc main.c -o mytest -Iinclude -Llib -lMytest
runsir@ubuntu:~/libtest$ ls
include  lib  main.c  mytest  src
runsir@ubuntu:~/libtest$ ./mytest
./mytest: error while loading shared libraries: libMytest.so: cannot open shared object file:
runsir@ubuntu:~/libtest$ ldd mytest
        linux-vdso.so.1 =>  (0x00007ffd5db6c000)
        libMytest.so => not found
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fb028492000)
        /lib64/ld-linux-x86-64.so.2 (0x0000559755413000)
runsir@ubuntu:~/libtest$
```

解决方案：可执行程序在运行时是通过动态链接器来加载动态库的。

①可以将自己所编写的动态库放到/lib 目录下，这样动态链接器就能够找到该动态库了。极其不推荐这样做。

②配置 **LD_LIBRARY_PATH** 环境变量：export LD_LIBRARY_PATH=自己所编写的动态库的路径，动态链接器会根据该环境变量找到动态库。也可以将该环境变量配置到~/.bashrc 文件中达到永久配置效果。

③gcc 编译时-L 参数只在编译时有效，所以在运行时应用程序是不知道该目录的，但是可以在编译时加上**-Wl,--rpath=your_lib_dir1** 参数来指明运行应用程序时动态库的位置。--rpath 是 ld 编译选项，所以前面要加上-Wl（小写 L）标明。

④将自己编写的动态库的路径添加到动态链接器的配置文件（/etc/ld.so.conf）中。

注：动态库的路径应该写绝对路径，否则改变应用程序

5、优缺点：

1）优点：

①应用程序的体积小。

②库发生改变，应用程序不需要重新编译。

2）缺点：

①发布程序的时候需要提供对应的库。
②加载库的速度慢。