Linux静态库动态库以及简单makefile的编写

1. 库

库是可以通用的代码的集合(本质上是一种二进制形式的可执行代码).比如我们常用的glibc库等,这样我们就不用编写重复的代码,只关心自身的业务逻辑即可.

1. 库的分类

一般情况下,库分为静态库和动态库,动态库也称为共享库.

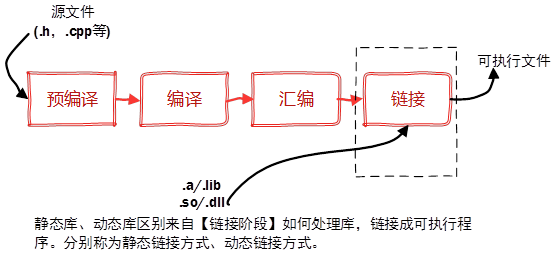
Linux下静态库一般为 libxxx.a, 动态库一般为 libxxx.so

Windows下的静态库一般为xxx.lib, 动态库一般为 xxx.dll

库的分类是按照链接方式的不同来进行区分的,链接分为静态链接和动态链接.

1. 编译过程

一个可执行文件的生成过程如下图:



1. 静态库

静态库在链接过程中会被整体打包进可执行程序,所以利用静态库生成的文件一般偏大.可执行文件对静态库的链接是在生成可执行文件之前完成的.

优点:编译生成的程序不需要依赖外部的函数库的支持,可以独立运行.

缺点:更新静态库时,同时需要更新整个应用程序,即重新编译生成可执行文件.

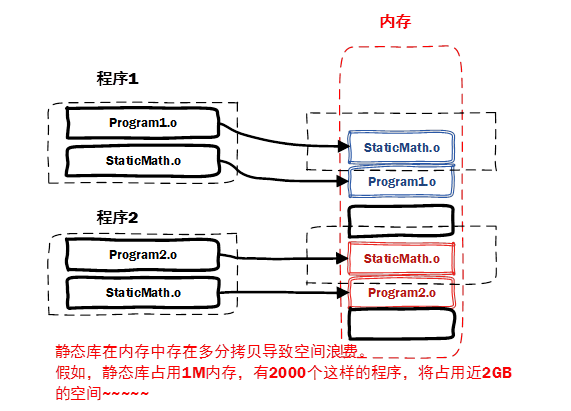
1. 动态库

动态库在编译的时候并没有被编译进可执行文件,只有当程序执行到动态库中的函数的时候,动态库才会加载到内存中,并且可以多个程序共享,所以动态库也被称为共享库.

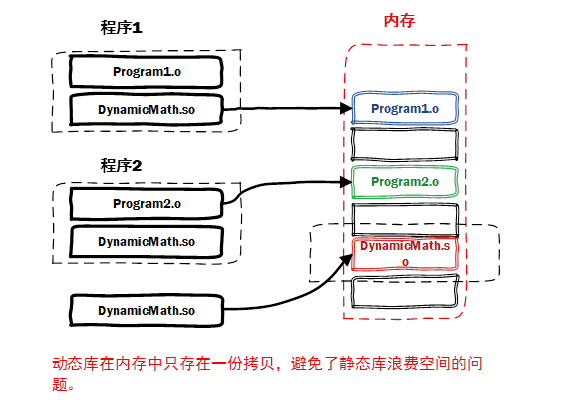
优点:可以实现进程之间库的共享,同时节省内存资源,而且程序的更新更方便,只需要更新动态库即可.

缺点:连接共享库会增加程序耗时,执行速度相对于静态库降低. DLL Hell问题.

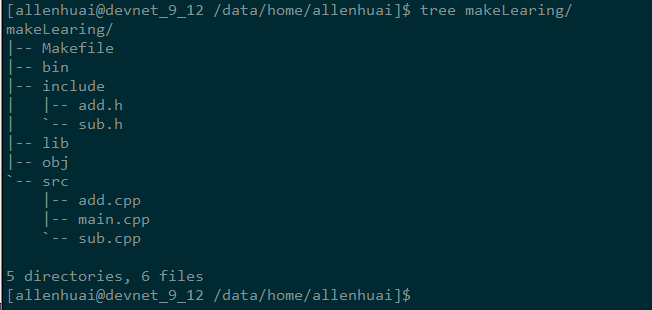
1. 静态库与动态库运行对比
   1. 静态库



* 1. 动态库



1. 演示
   1. 目录



Makefile文件

Bin目录用来存放可执行程序

Include目录用来存放头文件

Lib目录用来存放生成的库文件

Obj目录用来存放编译生成的.o文件

Src目录用来存放具体的代码文件

* 1. 文件内容

Add.h

#ifndef \_\_ADD\_H\_\_

#define \_\_ADD\_H\_\_

int add(int a, int b);

#endif

Sub.h

#ifndef \_\_SUB\_H\_\_

#define \_\_SUB\_H\_\_

int sub(int a, int b);

#endif

Add.cpp

#include "add.h"

int add(int a, int b)

{

return a + b;

}

Sub.cpp

#include "sub.h"

int sub(int a, int b)

{

return a - b;

}

Main.cpp

#include "add.h"

#include "sub.h"

#include <iostream>

using namespace std;

int main()

{

cout << add(1, 2) << endl;

cout << sub(2, 4) << endl;

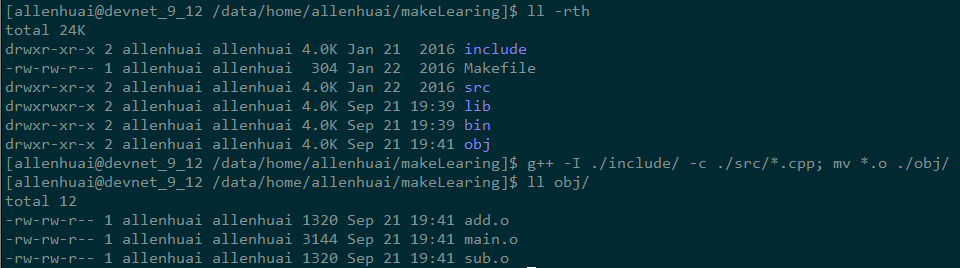
cout << "hello allen" << endl;

return 0;

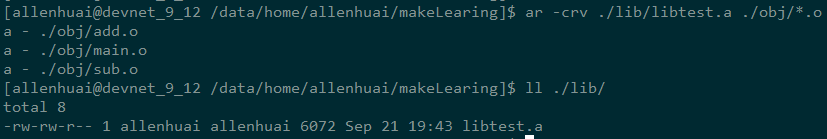
}

* 1. 静态库演示

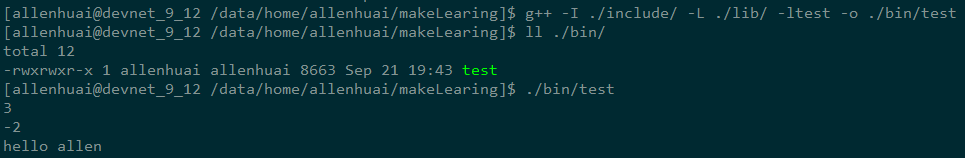
g++ -I ./include/ -c ./src/\*.cpp; mv \*.o ./obj/



**ar -crv ./lib/libtest.a ./obj/\*.o**

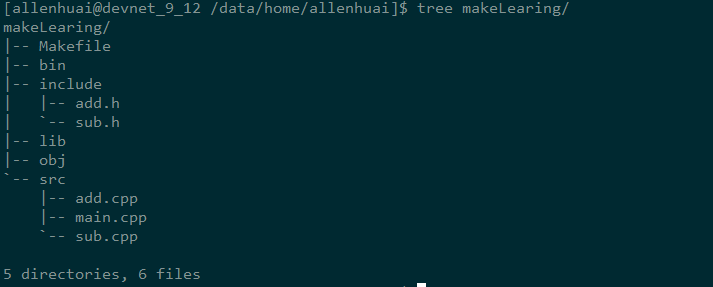


**生成可执行文件**

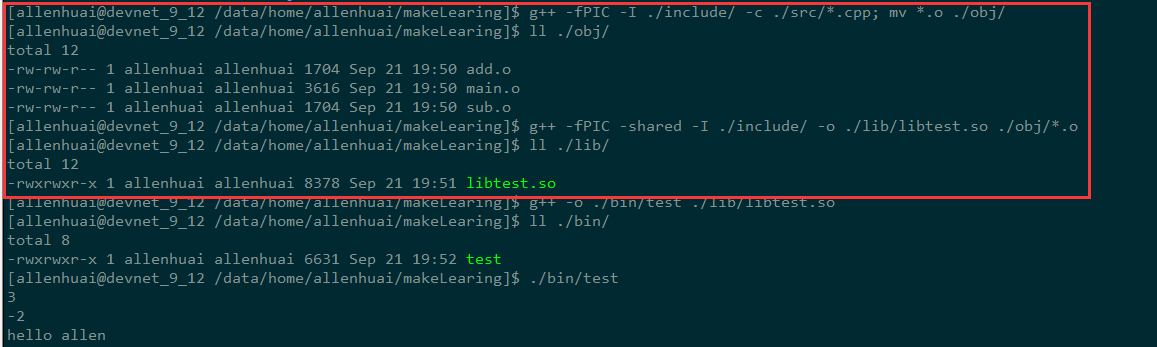


* 1. 动态库演示

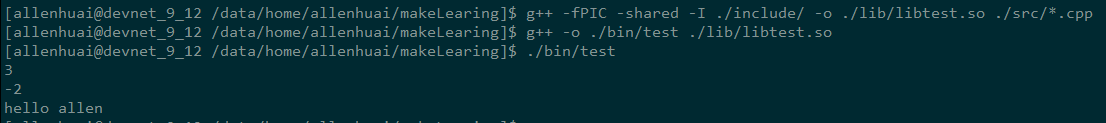
恢复之前的目录



操作和生成静态库类似,只是需要编译的时候需要-fPIC参数,用来生成地址无关的代码,因为动态库需要在程序运行的时候动态加载.



红框内的两步操作可以合并成一步



动态库的加载也可以指定路径

g++ -o ./bin/test -L ./lib/ -ltest

此时直接运行可执行文件会报错,提示./bin/test: error while loading shared libraries: libtest.so: cannot open shared object file: No such file or directory

当系统加载可执行代码时候，能够知道其所依赖的库的名字，但是还需要知道绝对路径。此时就需要系统动态载入器(dynamic linker/loader)。

对于elf格式的可执行程序，是由ld-linux.so\*来完成的，它先后搜索elf文件的 DT\_RPATH段—环境变量LD\_LIBRARY\_PATH—/etc/ld.so.cache文件列表—/lib/,/usr/lib 目录找到库文件后将其载入内存。

如何让系统能够找到它：

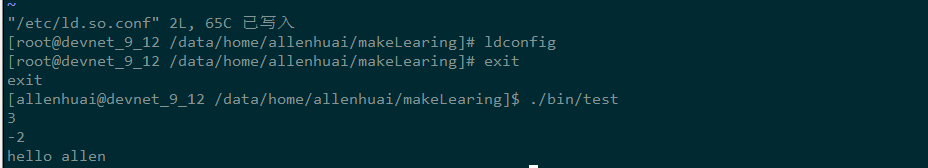
如果安装在/lib或者/usr/lib下，那么ld默认能够找到，无需其他操作。

如果安装在其他目录，需要将其添加到/etc/ld.so.cache文件中，步骤如下：

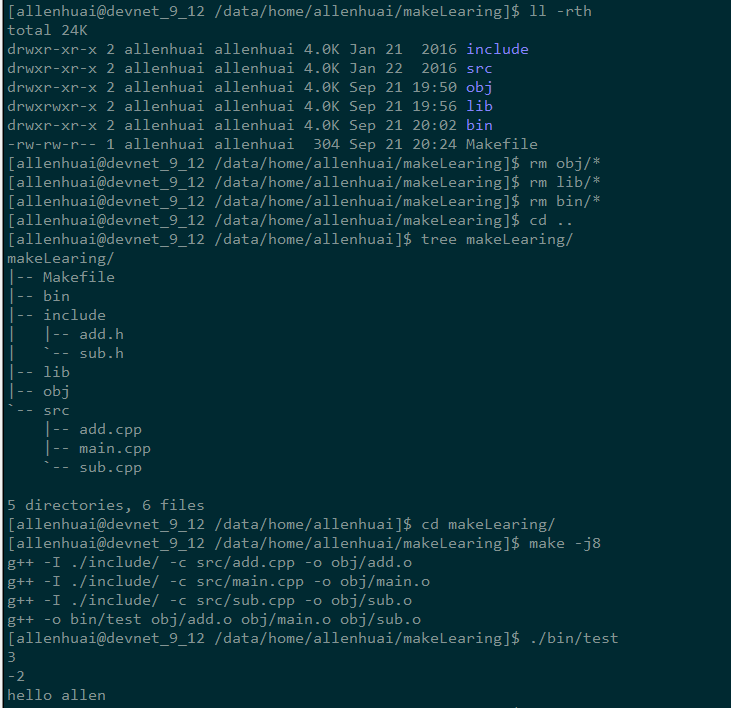
编辑/etc/ld.so.conf文件，加入库文件所在目录的路径

运行ldconfig ，该命令会重建/etc/ld.so.cache文件

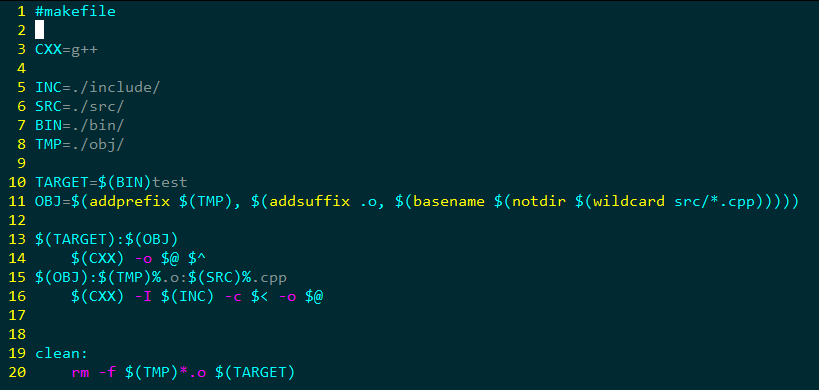




1. 简单makefile的编写



* 1. Makefile文件



Makefile的相关知识后续补上,可以参考陈皓的跟我一起写makefile

1. 附录 相关命令
   1. g++(gcc)编译选项

-shared：指定生成动态链接库

-static：指定生成静态链接库

-fPIC：表示编译为位置独立的代码，用于编译共享库。目标文件需要创建成位置无关码， 念上就是在可执行程序装载它们的时候，它们可以放在可执行程序的内存里的任何地方

-L.：表示要连接的库所在的目录

-l：指定链接时需要的动态库

 -Wall：生成所有警告信息

-ggdb：此选项将尽可能的生成gdb 的可以使用的调试信息

-g：编译器在编译的时候产生调试信息

-c：只激活预处理、编译和汇编,也就是把程序做成目标文件(.o文件)

-Wl,options : 把参数(options)传递给链接器ld 。如果options 中间有逗号,就将options分成多个选项,然后传递给链接程序

* 1. ar命令

在UNIX中,使用ar命令创建或者操作静态库ar->archivefile objfile

**-r : 将objfile文件插入静态库尾或者替换静态库中同名文件**

**-x : 从静态库文件中抽取文件objfile**

**-t : 打印静态库的成员文件列表**

**-d : 从静态库中删除文件objfile**

**-s : 重置静态库文件索引**

**-v : 创建文件冗余信息**

**-c : 创建静态库文件**

* 1. nm

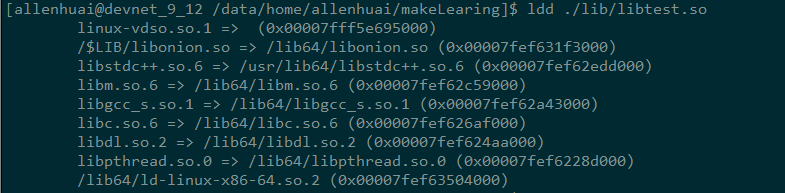
nm命令可以打印出库中的涉及到的所有符号。库既可以是静态的也可以是动态的。

一种是在库中被调用，但并没有在库中定义(表明需要其他库支持)，用U表示

一种是库中定义的函数，用T表示，这是最常见的

* 1. ldd

查看可执行程序所依赖的共享库.



1. 附件

