**1静态库和共享库**

\*本节就如何创建和使用程序库进行论述。所谓“程序库”，简单说，就是包含了数据

和执行码的文件。其不能单独执行，可以作为其它执行程序的一部分来完成某些功能。库的

存在，可以使得程序模块化，可以加快程序的再编译，可以实现代码重用，可以使得程序便

于升级。程序库可分静态库(static library)和共享库(shared object)。

1. 库由专门的团队管理, 这个团队仅仅关心库相关功能的逻辑. 更加专业.
2. 库一般都是二进制的形式提供给别人, 这样其他不需要编译相关代码, 只是最后连接
3. 代码公用, 不同的团队不用实现相同的逻辑.

对于维护特定库的工作人员, 要开阔自己的眼界, 不能将自己的能力, 限制在工作局限的范围里面.

**A:静态库**

是在可执行程序运行前就已经加入到执行码中，成为执行程序的一部分；共享库，是在

执行程序启动时加载到执行程序中，可以被多个执行程序共享使用。

静态库的地址, 是在连接器时候, 由链接器确定的.

动态库的地址, 是在程序加载的时候, 动态链接器确定的.

总之, 库的地址, 在运行的时候是要确定下来的. 这样其他模块的功能, 才能确定库中各个符号的位置, 实现正常的调用.

建议库开发人员创建共享库，比较明显的优势在于库是独立的，便于维护和更新；而静

态库的更新比较麻烦，一般不做推荐。然而，它们又各有优点，后面会讲到。

节所讲述的执行程序和库都采用ELF(Executable and Linking Format)格式,尽管GNU

GCC工具可以处理其它格式，但不在本节的讨论范围。

静态库可以认为是一些目标代码的集合。按照习惯，一般以“.a”做为文件后缀名。使

用ar(archiver)命令可以创建静态库。因为共享库有着更大的优势，静态库已经不经常使

用。但静态库使用简单，仍有使用的余地，并会一直存在。有些Unix系统，如Solaris 10，

已经基本废弃了静态库。

静态库在应用程序生成时，可以不必再编译，节省再编译时间。但在编译器越来越快的

今天，这一点似乎已不重要。如果其他开发人员要使用你的程序，而你又不想给其源码，提

供静态库是一种选择。从理论上讲，应用程序使用了静态库，要比使用动态加载库速度快

1-5%，但实际上可能并非如此。由此看来，除了使用方便外，静态库可能并非一种好的选

择。

要创建一个静态库，或要将目标代码加入到已经存在的静态库中，可以使用以下命令：

ar rcs libmylib.a file1.o

file2.o以上表示要把目标码file1.o和file2.o加入到静态库libmylib.a中(ar的参数

r)。若libmylib.a不存在，会自动创建(ar的参数c)。然后更新.a文件的索引，使之包含新

加入的.o文件的内容(ar的参数s)。

静态库创建成功后，需要链接到应用程序中使用。使用gcc的-l选项来指定静态库，使

用-L参数来指定库文件的搜索路径。比如上述例子应指定-lmylib，所有库文件名都以lib开

头，开头的lib在指定参数时应省略。-l和-L之后都直接带参数而不跟空格。

静态库的连接, 目前的 Ide 里面都可以自动完成, 开发人员仅仅是在项目管理界面, 配置一下需要连接的库的路径.

这点在 Qt 的 pro 文件里面, 可以表示出来.

在最后打包连接的时候, 链接器需要把库和项目文件打包到一起, 所以, 需要表明到底哪些库要进行连接. IDE 会取调用各种连接命令.

在使用gcc时，要注意其参数的顺序。-l是链接器选项，一定要放在被编译的文件名称

之后；若放在文件名称之前则会连接失败，并会出现莫名其妙的错误。这一点切记。

**B:共享库**

共享库的创建比较简单，基本有两步。首先使用-fPIC或-fpic创建目标文件，PIC或

pic表示位置无关代码，然后就可以使用以下格式创建共享库了： gcc -share -Wl,-

soname,your\_soname -o library\_name file\_list library\_list 下面是使用a.c和b.c创建

库的示例：

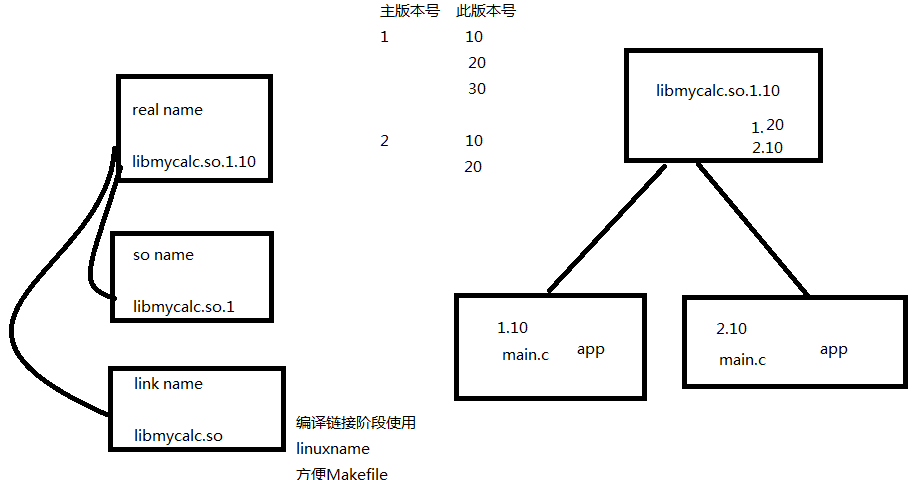
所谓位置无关的代码, 就是在库的内部, 访问变量, 访问函数, 不在使用绝对地址, 而是使用偏移量进行.

这个可以用绝对路径和相对路径来理解.

如果一个模块, 使用的都是相对路径, 那么这个模块只要能够保证整体进行迁移, 那么这个模块的内部逻辑就不会被进行破坏.

各个项目, 都是使用相对路径管理自己项目的资源, 代码文件, 库文件.

而需要暴露给外界的接口, API, 则需要特殊的标明. 而模块内的其他功能, 则使用相对路径, 封装在模块的内部.



相关命令：

gcc –fPIC –c XX.c

gcc –fpic –c XX.c

gcc -shared -Wl,-soname,libmyab.so.1 -o libmyab.so.1.0.1 a.o b.o

按照共享库的命名惯例，每个共享库有三个文件名：real name、soname和linker

name。真正的库文件（而不是符号链接）的名字是real name，包含完整的共享库版本号。

soname是一个符号链接的名字，只包含共享库的主版本号，主版本号一致即可保证库函

数的接口一致，因此应用程序的.dynamic段只记录共享库的soname，只要soname一致，这个

共享库就可以用。如libmyab.so.1和libmyab.so.2是两个主版本号不同的libmyab，有些应

用程序依赖于libmyab.so.1，有些应用程序依赖于libmyab.so.2，但对于依赖libmyab.so.1

的应用程序来说，真正的库文件不管是libmyab.so.1.10还是libmyab.so.1.11都可以用，所

以使用共享库可以很方便地升级库文件而不需要重新编译应用程序，这是静态库所没有的优点。

注意libc的版本编号有一点特殊，libc-2.8.90.so的主版本号是6而不是2或2.8。

linker name仅在编译链接时使用，gcc的-L选项应该指定linker name所在的目录。有

的linker name是库文件的一个符号链接，有的linker name是一段链接脚本。例如上面的

libc.so就是一个linker name，它是一段链接脚本：

**C:共享库加载**

在所有基于GNUglibc的系统中，在启动一个ELF二进制执行程序时，一个特殊的

程序“程序装载器”会被自动装载并运行。在linux中，这个程序装载器就是/lib/ldlinux.

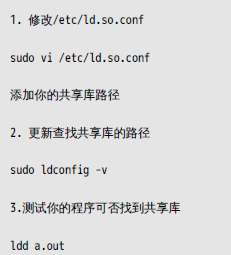
so.X(X是版本号)。它会查找并装载应用程序所依赖的所有共享库。被搜索的目录保存在/etc/ld.so.conf文件中。当然，如果程序的每次启动，都要去搜索一番，势必效率不

堪忍受。Linux系统已经考虑这一点，对共享库采用了缓存管理。ldconfig就是实现这一功

能的工具，其缺省读取/etc/ld.so.conf文件，对所有共享库按照一定规范建立符号连接，

然后将信息写入/etc/ld.so.cache。/etc/ld.so.cache的存在大大加快了程序的启动速

度。



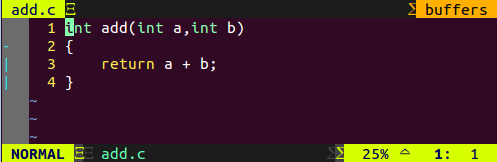
D:案例说明：

D1:创建一个目录，mycal

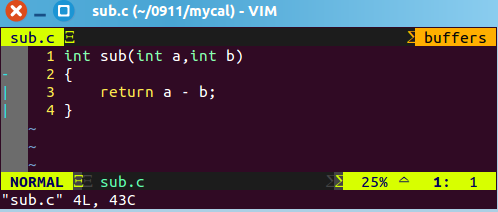
mkdir mycal

D2:创建4个c文件盒1个.h,分别实现加减乘除

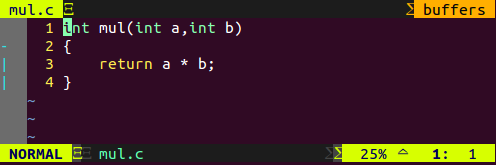
add.c



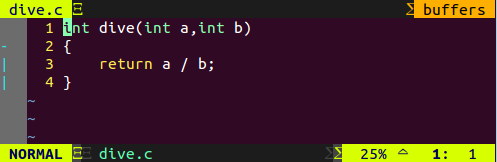
sub.c



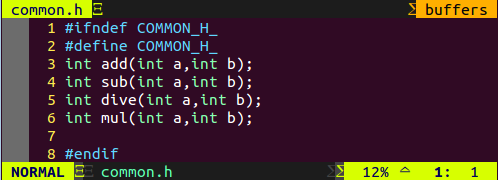
mul.c



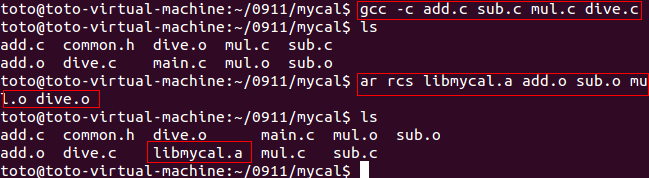
dive.c



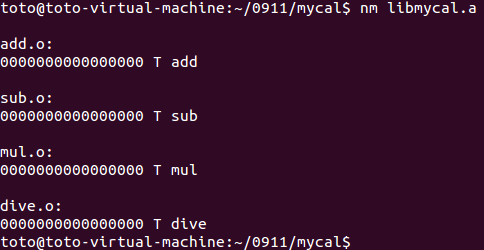
common.h



D3制作静态库



查看静态库信息：



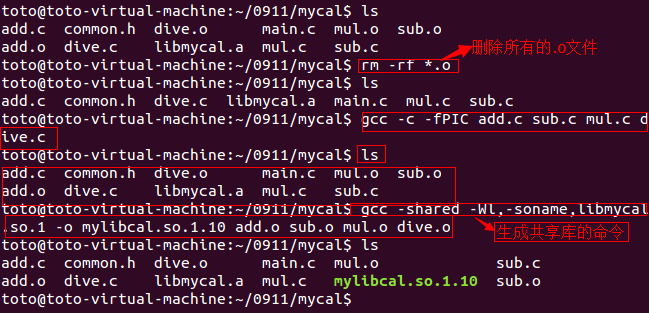
使用静态库(程序扔到任何电脑都可以运行，因为静态库相当于包含到了静态库中)：

gcc -Isrc main.c libmycal.a -o app

ldd app命令检测的是共享库，不检测静态库

上面的过程相当于libmycal.a和main.c进行组合，生成app

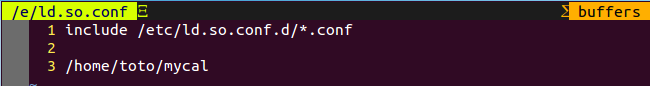
D4制作共享库



D5设置共享库加载路径

打开共享库路径配置文件

sudo vi /etc/ld.so.conf

最后一行添加mycal路径,截图如下 

sudo ldconfig –v

gcc main.c mylibcal.so.1.10 -o app

ldd app查看依赖信息

注意：共享库和main.c共同使用的时候，在这个过程中相当于生成了main.o,符号记录表。app运行的时候依赖共享库版本。