### 编译过程

1. **预处理**：处理所有以’#’开头的代码，包括头文件、宏定义、条件编译

gcc -E hello.c -o hello.i

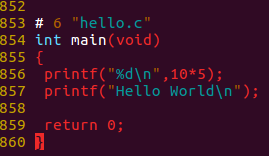


同时还会把注释的内容给去掉，只保留没有注释的东西。还会将宏定义进行替换例如：

hello.c:



hello.i:按shift + g键命令跳到最后一行



可以看到注释没了，同时宏定义的X也替换成了10。

1. **编译**：语法检查以及将C语言变成汇编语言

gcc -S hello.i -o hello.s



1. **汇编**：将汇编语言变成二进制文件

gcc -c hello.s -o hello.o



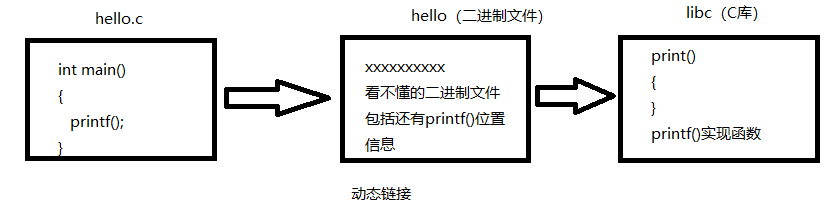
1. **链接**：链接代码需要用到的其他文件（库文件等）

gcc hello.o -o hello



### 二、静态链接和动态链接

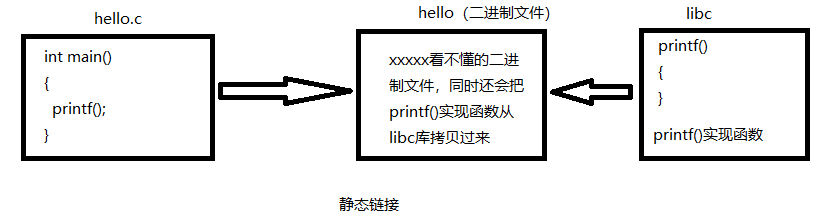
链接默认的方法都是动态的链接。大致意思就是当我们执行代码执行到printf();的时候会把我们printf()在哪的信息链接进来，在运行的时候会根据你链接进来的位置信息找到这个printf()函数进行执行。

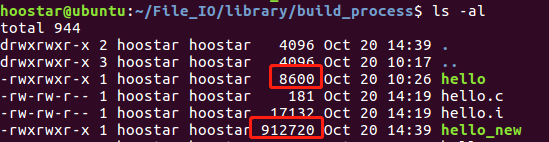


链接方法还有一个静态链接：



我们可以对比一下看看两种方法生成的可执行文件的大小:



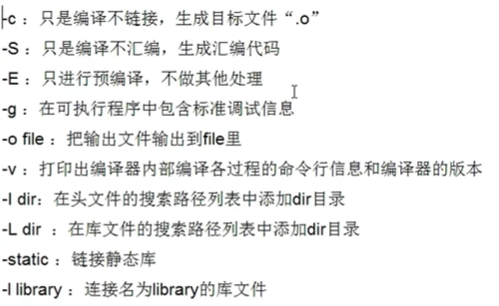


所以静态链接方法比动态链接方法得到的可执行二进制文件大的多。

两种方法的对比：

1. 动态链接时，生成的可执行文件体积小，静态链接时生成的可执行文件体积大
2. 动态链接时，若后面链接的库没了，则可执行文件将会无法执行，而静态链接的时候就算你后面不小心把库删了，可执行文件照样可以正常执行。

### 三、gcc编译常用的参数



### 动态库和静态库的制作

上面我们讲了静态链接和动态链接，那么这是各自需要库的。现在我们就来讲解一下静态库和动态库的制作。

为什么我们需要知道这个呢？举个例子，有一家公司要处理某一项业务，但这方面的业务却不是他们擅长的，那么公司就需要将这项业务外包出去，那么外包出去的业务在对方做完之后，对方是不会直接将源码发给你的，否则很可能白给你们干活了。他们在做完之后会编译好并制作成库的形式给你们。

#### 静态库制作

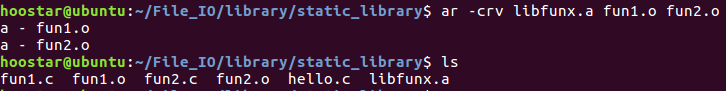
1. 首先写三个源文件：hello.c、fun1.c、fun2.c
2. 将这三个源文件编译成二进制文件。gcc -c命令



1. 将二进制文件fun1.o和fun2.o生成静态库

使用命令ar -crv libfunx.a fun1.o fun2.o

其中，静态库固定格式是以lib开头，以.a结尾，中间的funx是自己起的库的名字。



1. 使用静态链接进行编译



gcc hello.c -o hello -static -L . -lfunx

首先-static表示是进行静态链接，-L后面接库的位置，’.’表示当前目录下，-l表示要链接的库的名字，funx就是库的名字。

#### 动态库（共享库）制作

1. 首先写三个源文件
2. 然后可以直接将源文件生成动态库文件。



gcc -fPIC -shared -o libfunx.so fun1.c fun2.c

动态库以lib开头，以.so结尾，中间funx是自己起的库的名字。

（3）动态链接编译

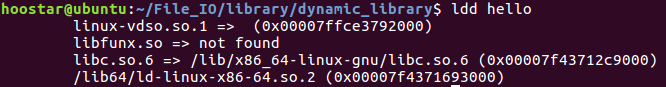


-L后面接库的位置，’.’表示当前目录下，-l表示要链接的库的名字，funx就是库的名字。

执行一下：



发现报错了，是因为系统默认会从指定路径下查找库，我们可以通过ldd命令查看一下：



它找不到对应的库not found。

ldd命令：可以查看所用到的可执行二进制文件要用到的库在哪找

（4）将制作好的动态库放入到指定路径下去。



1. 执行最终的可执行文件



1. 为了安全起见，我们还是把刚刚放进/lib/x86\_64-linux-gnu/目录下的libfunc.so删了。



1. 在执行的时候如何定位动态库文件

当系统加载可执行代码的时候，能够知道其所依赖的库的名字，但还得需要所依赖的库的绝对路径。

动态链接器：ld-linux-x86-64.so.2

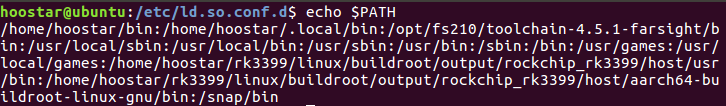
对于ELF格式的可执行文件，是由动态库链接器完成库的链接，首先查找elf文件DAT\_RPATH段（二进制文件），其次再到环境变量LD\_LIBRARY\_PATH查找，再去/etc/ld.so.cache文件列表，最后去/lib，/usr/lib目录下查找。查找到之后将其装载到内存中。若查找不到则会出现上面那种报错的提示：



解决办法:

1. 将做好的动态库放入到/lib目录下去。就像之前上面演示过的那样。（不太建议使用）
2. 通过设置环境变量：

通过echo $PATH查看环境变量PATH的值。

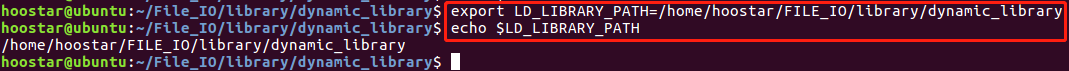


通过echo $LD\_LIBRARY\_PATH查看环境变量LD\_LIBRARY\_PATH的值发现是空的。

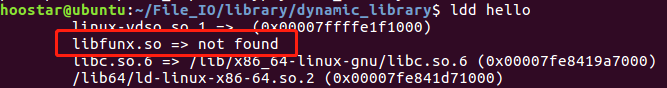


临时设置环境变量

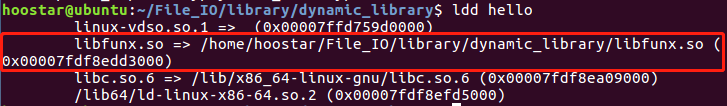
export LD\_LIBRARY\_PATH=动态库的路径（注意：=左右不要敲空格，路径最好是绝对路径）



再通过ldd查看一下发现还是找不到动态库路径：



重启一下终端，再执行一遍操作就有了。

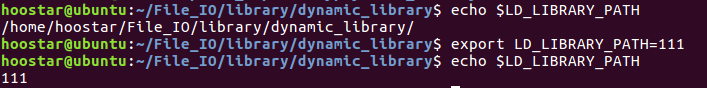


这样就可以正常执行了可执行文件了。

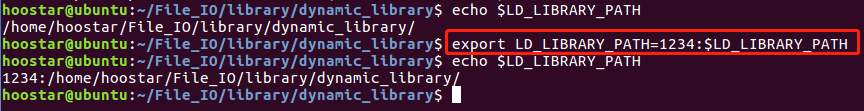


但是临时设置的话，就会导致终端一经关闭设置的环境变量就会消失。

通过直接export LD\_LIBRARY\_PATH xxx的方法会导致后面的将前面的覆盖掉。比如：



我们应该使用拼接的方式：



#### 设置/etc/ld.so.cache文件列表

sudo vi /etc/ld.so.config打开配置文件并添加路径



再执行sudo ldconfig -v



#### 动态库和静态库的区别

静态库：

优点：

1. 静态被打包到对应出现中，加载速度快。
2. 发布后无需静态库，移植方便

缺点：

1. 消耗系统资源大，浪费内存
2. 更新麻烦

动态库：

优点：

1. 方便共享代码
2. 体积小
3. 安全，看不到源代码

缺点：

1. 需要链接到库后执行，加载速度慢。
2. 动态库一旦消失就无法运行可执行文件

区别：

加载时刻不同，静态库是在程序编译时被链接到目标文件中，动态库是在使用到库的时候才会被链接。