## Lesson1\_GNU开发环境基础

**课程目标**

掌握GCC和GDB的使用 ， 掌握Eclipse Linux下的使用

    通过研究GCC源代码，分析GCC程序的关键结构 进而将来扩展编译器的编写 。

**重点难点**

GCC编译过程

GCC常规使用

GDB常规使用

**考核目标**

如何通过GCC编译动态链接库然后调用

GCC的常规编译选项有哪些？

如果要用GDB调  试，GCC如何编译？

**课后练习**

将课堂上的练习都做一遍

### gcc开发环境

### 简介

GCC （GNU C Compiler) 是GNU项目中符合ANSI C标准的编译系统 ， 能够编译用C ， C++，Object C等语言编写的程序 ， 同时gcc也是一个交叉编译器 ，特别适用于不同平台的嵌入式开发。

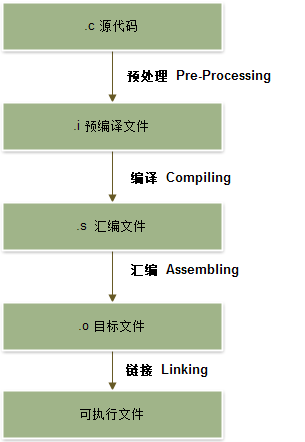
### GCC Helloworld

sample:gcc helloworld

gcc XXX.c -o XXX //(将.c文件编译成目标文件) 。-o指的是目标文件object

./XXX //执行目标文件

**GCC程序编译过程**



预处理把所有include嵌套进来成 .i文件，编译成汇编文件.s，汇编过程生成机器码二进制目标文件object .o文件，多个目标文件打包就是链接成最终可执行文件。

GCC支持的后缀名

|  |  |
| --- | --- |
| **后缀名** | **对应的语言** |
| .c | C原始程序 |
| .C/.cc/.cxx | C++原始程序 |
| .m | Objective-C原始程序 |
| .i | 已经预处理的C原始程序 |
| .ii | 已经预处理的C++原始程序 |
| .s/.S | 汇编语言原始程序 |
| .h | 预处理文件（头文件） |
| .o | 目标文件 |
| .a(archive)/.so | 编译后的库文件，静态库和动态库 |

**后缀名对应的语言**

.c C原始程序 .C/.cc/.cxx C++原始程序 .m Objective-C原始程序.i已经预处理的C原始程序.ii已经预处理的C++原始程序.s/.S汇编语言原始程序.h预处理文件（头文件）.o目标文件.a(archive)/.so编译后的库文件，静态库和动态库

**研究GCC参数 ： 1.看帮助 2.看源码**

GCC使用参数分类

通用参数

C语言参数

C++参数

Object C参数

语言无关参数

警告参数

调试参数

优化参数

编译过程

预编译参数

汇编参数

链接参数

机器依赖参数

比如ARM平台编译

最常用参数：

gcc [-c|-S|-E] [-std=standard]

[-g] [-pg] [-Olevel]

[-Wwarn...] [-pedantic]

[-Idir...] [-Ldir...]

[-Dmacro[=defn]...] [-Umacro]

[-foption...] [-mmachine-option...]

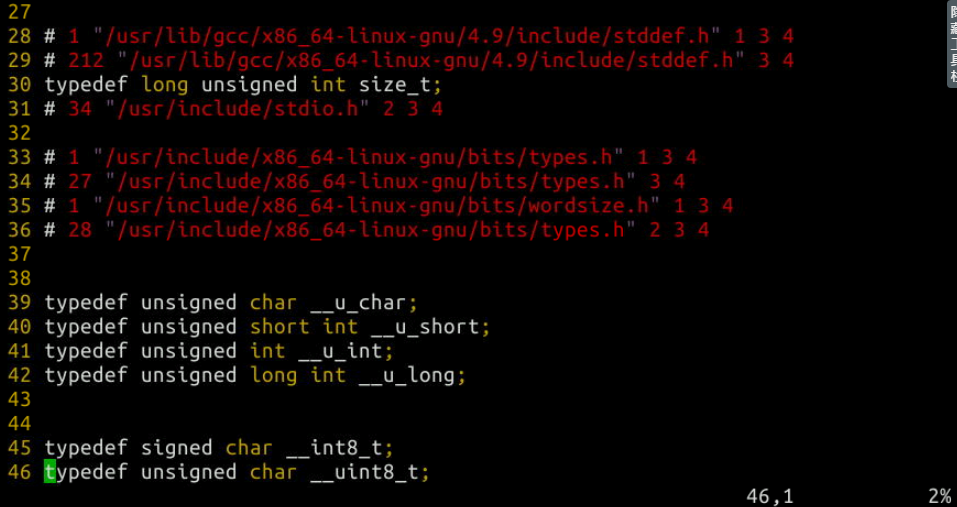
[-o outfile] [@file] infile...

**编译过程参数**

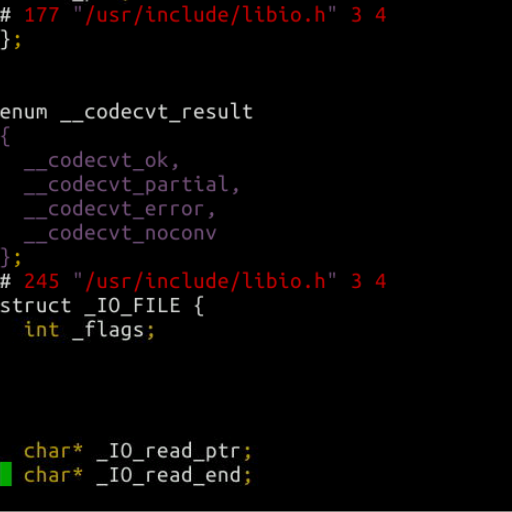
|  |  |
| --- | --- |
| -c | 只编译不链接，生成目标文件.o |
| -S | 只编译不汇编 ， 生成汇编代码 |
| -E | 只预编译 |
| -g | 包含调试信息 |
| -o file | 指定目标输出文件 |
| -I[dir] | 搜索头文件路径 |

gcc -E mengleihello.c -o mengleihello.i //预处理成 .i 预编译文件

vi mengleihello.i



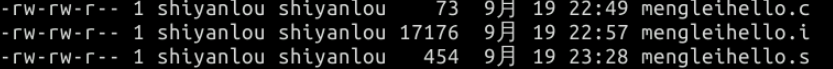
如图可见，在.i中头文件被层层展开了。



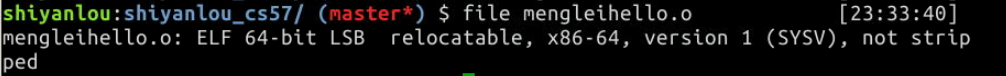
如图177和245的文件被展开使用



gcc -S mengleihello.i -o mengleihello.s //编译成 .s 汇编文件

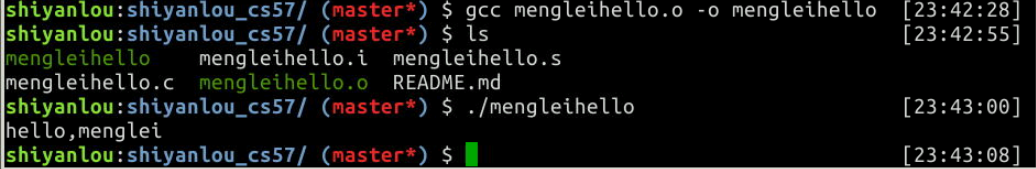


gcc -c mengleihello.s -o mengleihello.o //将.s汇编文件，汇编成为.o目标文件。



gcc mengleihello.o -o mengleihello //最后一步链接文件,使其可执行。

./mengleihello 执行

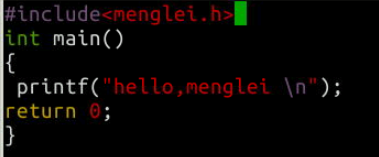


gcc -g mengleihello.c -o mengleihello2 //加入调试信息

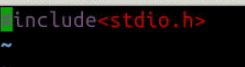


mengleihello2因为添加了调试信息所以要比mengleihello大一些，下面的GDB会使用到。

menglei.c中的内容



menglei.h中的内容



出错信息



whereis stdio.h //这就是头文件所在位置



gcc -I. menglei.c -o menglei // -I. 即为在当前路径下搜索头文件编译

gcc -I~/shiyanlou/Code/shiyanlou\_CS menglei.c -o menglei //-I 后边紧跟文件目录



**库选项**

静态链接库是指编译的时候把库文件完全链接到可执行文件中了，将来发布可执行文件的时候，类库就可以不用了。因为类库的内容已经完全包含到可执行文件中了。这样的特点是发布很方便，不好的地方在于有时候就为了一点简单的类库调用，就要把库文件完全包含到可执行文件中。造成程序臃肿，而且万一修改类库，则可执行文件都要重新再编译一遍。不适合扩展。

动态链接库在运行时调用，在运行时用到哪个类库，再去加载哪个类库，并且调用它的相关函数。可执行文件比较精简，也比较灵活。但是在发布时除了要有可执行文件，还要包含其中的类库文件。类库文件版本发生了变化，也要及时进行更新。去维护类库版本。

以上两种库的最大区别在于链接的时候

**linux下静态链接库和动态链接库格式**

.a [libname.a]

.so [libname.so.[主版本号.次版本号.发行号]]

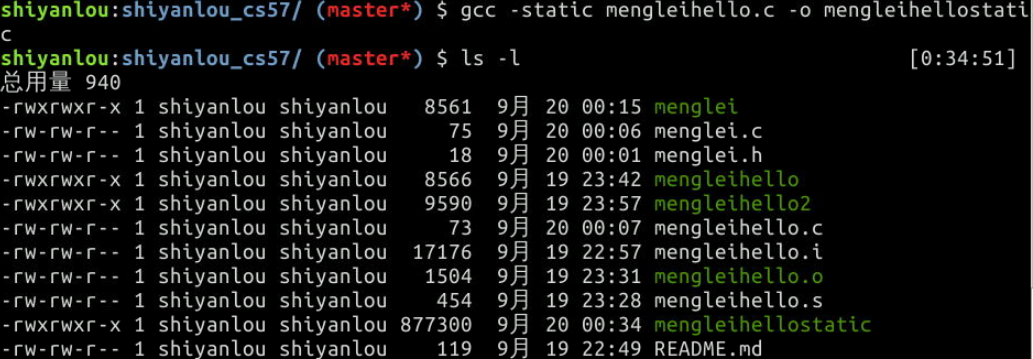
以上都在linux中的 /usr/lib 中

|  |  |
| --- | --- |
| -static | 静态编译 |
| -shared | 1，生成动态库文件  2，进行动态编译 |
| -L dir | 库文件搜索中添加路径 |
| -fPIC | 生成使用相对位置无关的目标代码(Position Independent Code)，然后通常用于使用gcc的-static选项从该PIC目标文件生成动态库文件 |

gcc默认的链接是动态链接

gcc -static mengleihello.c -o mengleihellostatic

可以看出链接出的文件明显很大



、

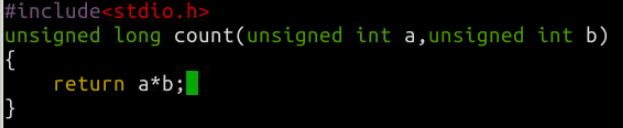
编译静态链接库

1，先生成目标文件 .o

2, ar crv [\*.a] [\*.o] //a是用来打包的工具，将.o 目标文件打包成.a 的文件，crv是参数

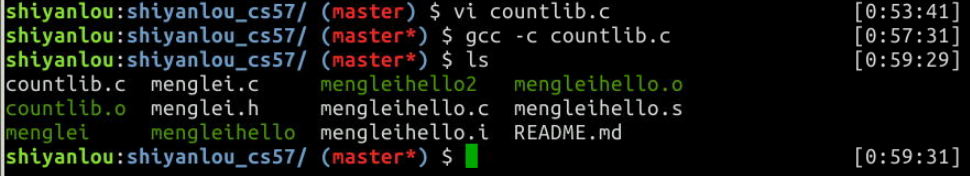
//r就是插入一些file member 数据，c为creat建立一个归档文件archive

建立一个countlib.c文件,内容如下：

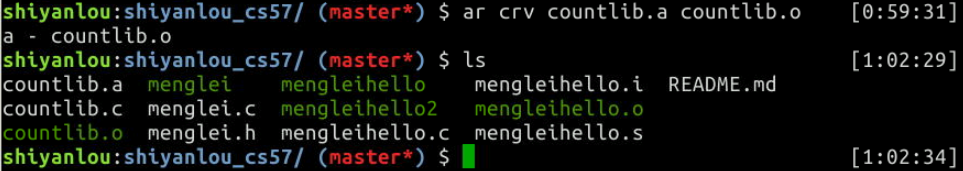


将countlib.c文件编译成一个静态链接库，

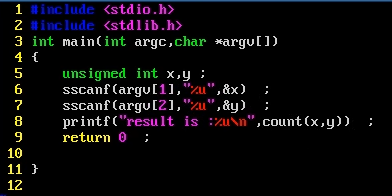
gcc -c countlib.c //先生成一个目标文件



再打包：ar crv countlib.a countlib.o //countlib.a 为生成的静态库文件名称



写一个count.c类型的c语言文件，内容如下：

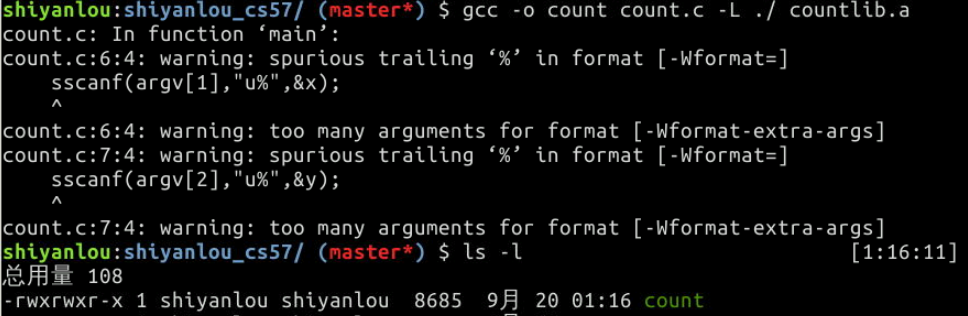


编译时指定其动态链接库

调用静态链接库

gcc -o [file] [file.c] -L. [file.a]

gcc -o count count.c -L ./ countlib.a



count即为编译好的文件,执行count文件，别忘了加参数。



编译动态链接库

1，生成位置无关的目标代码 gcc-fPIC -c [\*.c]

2，gcc -shared -o [\*.so] [\*.o]