## 编译流程

gcc编译程序时分为4个阶段：

1. 预处理(Pre-processing)， 将文件包括的.h文件，宏定义等在文件中展开。

2. 编译(compiling)，在这个阶段中，Gcc首先要检查代码的规范性、是否有语法错误等，以确定代码的实际要做的工作，在检查无误后，Gcc把代码翻译成汇编语言。

3. 汇编(Assembling)，汇编阶段是把编译阶段生成的”.s”文件转成二进制目标代码，.o文件

4. 链接阶段(Link)，在成功编译之后，就进入了链接阶段，将各个文件互相连接，检查外部变量函数等，生成目标文件。

静态库与动态库通常的编译器实现遵守ILP32或LP64规范：

ILP32缩写：int（I）、long（L）和指针（P）类型都占32位，通常32位计算机的C编译器采用这种规范，x86平台的gcc也是如此。

LP64是指：long（L）和指针占64位，通常64位计算机的C编译器采用这种规范。

## 操作命令

预编译操作：

1.把用户定义的所有的预定义#define展开

2.把所有的#include的文件内容加载进来。

3.处理条件编译，将不符合条件的代码段删除。

预编译：gcc -E hello.c -o hello.i hello.c #文件经过预编译后生成hello.i文件。

汇编：gcc -S hello.i -o hello.s 或者

gcc -S hello.c -o hello.s hello.c

文件经过预编译+编译后生成hello.s文件，即汇编代码。

编译机器代码：gcc -c hello.s -o hello.o 或者

gcc -c hello.c -o hello.o hello.c

文件经过预编译+编译+汇编后生成hello.o目标文件，即机器代码。

可执行文件：gcc hello.o -o hello 或者

gcc hello.c -o hello hello.c

文件经过预编译+编译+汇编+连接后生成hello执行文件。

### 参数：

-o ------输出文件的文件名

-O -----优化代码

-O2 ----优化代码，比-O更加优化 -O3最快，-O2次之

-I -------#include<>这样的头文件，就在-I后面这个目录中找。#include"  " 这样的头文件，先在当前目录下寻找，如找不到再到-I目录下寻找。

-L -------指定需要的库文件（.a  .so）文件所在的目录

- static -----静态编译，不使用各种库。 编译出的文件 较大

-D ------------宏定义。 eg：gcc -DDEBUG   使用这样的命令，相当于在每个文件中都添加了 #define DEBUG  这个代码，DDEBUG 定义DEBUG宏

-g ------------编译进去调试信息，gdb输出debug调试信息

-fPIC："Generate position-independent code (PIC)" gcc 编译.o文件时附带

表示编译为位置独立的代码，不用此选项的话编译后的代码是位置相关的所以动态载入时是通过代码拷贝的方式来满足不同进程的需要，而不能达到真正代码段共享的目的。

-dn -dy　参数来控制使用的是动态库还是静态库，-dn表示后面使用的是静态库，-dy表示使用的是动态库

-shared 该选项指定生成动态连接库

-Wl,option 此选项传递option给连接程序;如果option中间有逗号,就将option分成多个选项,然后传递给会连接程序.

-Wl,E 此"-E"根本就不是GCC的选项，而是LD的选项，因为GCC的"W1"选项就是给LD指定参数的。然后查看LD文档

###**lua5.2's error: multiple Lua VMs detected** Don't link your C module with liblua.a when you create a .so from it. For examples, see my page of Lua libraries: http://www.tecgraf.puc-rio.br/~lhf/ftp/lua/ . You can link liblua.a statically into your main program but you have to export its symbols by adding -Wl,-E at link time. That's how the Lua interpreter is built in Linux.

## Makefile

.PHONY : all clean distclean

#all, clean, distclean不发生歧义，以Makefile为主。

uname\_S := $(shell sh -c 'uname -s 2>/dev/null || echo not')

#2>/dev/null 2号文件描述符转移到/dev/null, uname没有结果就赋值not

src/red.o : src/red.c

gcc $(CFLAGS) -o $@ $^ -Isrc $(LDFLAGS)

#$@:目标集(src/red.o) $^:依赖目标集(src/red.c)

.c.o:

gcc -c $<

  #表示所有的 .o文件都是依赖与相应的.c文件的。

SRCS = $(shell echo src/\*.c)

OBJS = $(SRCS:.c=.o)

#变量OBJS的值就是将SRCS里面所有.c文件编译出的.o目标文件

Make –j 4 同时4核编译

## 链接顺序

gcc中库的链接顺序是从右往左进行，所以要把最基础实现的库放在最后，这样左边的lib就可以调用右边的lib中的代码。同时，当一个函数的实现代码在多个lib都存在时，最左边的lib代码最后link，所以也将最终保存下来。

需要注意的是：使用方式中静态库出现的顺序，libtest2.a必须在libtest1.a之前，否则会有"undefined reference to \*\*\*”的链接错误，其中\*\*\*就是libtest2.a中使用的libtest1.a的接口。

如果libtest1.a在前，为什么会有链接错误呢？

原因是gcc在链接的时候，对于多个静态库或者.o文件是从前往后依次处理的，如果当前的静态库或.o文件中没有使用的符号，则往后继续寻找，而不会再往前查找。