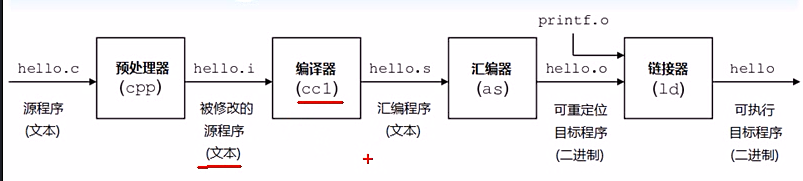
一.gcc

gcc 是一个可移植的支持多平台和跨平台的交叉编译程序

二.编译过程



三.gcc常用选项

-o 产生目标（可执行文件）

-c 通知gcc取消连接步骤，即编译源码并在最后生成目标文件

-E 只运行C预编译器，（头文件，宏的展开）

-S 告诉编译器产生汇编语言文件后停止编译，产生的汇编语言文件扩展名为.s

-Wall 使gcc对源文件的代码有问题的地方发出警告

-Idir 将dir目录搜索头文件的目录路径

-Ldir 将dir目录加入搜索库的目录路径

-llib 链接lib库

-g 在目标文件中嵌入调试信息，以便gdb之类的调试程序调试

-pedantic 对程序中不符合标准C的代码发出警告

-Werror 将警告作为错误

-x 指定源代码语言

-O0/O1/O2/O3 优化等级，缺省O1.（O0不优化）

注：文件类型说明

.h c语言源代码头文件

.c 预处理前的C语言源代码文件

.i 预处理后的C语言源代码文件

.s 汇编语言文件

.o 目标文件

.a 静态库文件

.so 共享库（动态库）文件

例： 预编译 gcc -E hello.c -o hello.i ,此阶段会将头文件和宏定义进行替换

编译 gcc -S hello.i -o hello.s ，此阶段生成汇编文件

汇编 gcc -c hello.s -o hello.o 将汇编文件编译为二进制文件

链接 gcc hello.o -o hello 将库和.o文件生成可执行文件

gcc -wall hello.c -o hello 添加警告信息

编译多个源程序

一次性编译

gcc [选项参数] a.c b.c c.c d.c -o abcd

独立编译

gcc -Wall -c a.c -o a.o

gcc -Wall -c b.c -o b.o

gcc -Wall a.o b.o -o ab

三.预处理命令

#include

#define PI 3.14

#undef PI 取消宏定义

#if 判定条件编译

#ifdef PI 判定宏PI是否定义，如果有定义

#ifndef 如果没有定义

#else

#endif//结尾

## 连接两个字符

#

#error 产生错误，结束并终止编译过程

#warning 产生警告

预定义宏

\_BASE\_FILE\_ //正在被编辑的源文件名

\_FILE\_ //所在的文件名

\_LINE\_ //所在的行号

\_FUNCTION\_ //所在函数名

\_func\_ //所在函数名

\_DATE\_ //编译日期

\_TIME\_ //编译时间

\_INCLUDE\_LEVEL\_ //包含层数，基0

\_cplusplus//是否是C++编译器，是C++编译器此宏为1，不是C++此宏无定义

详见 /Source/Linux工具使用/预定义宏

环境变量

C\_INCLUDE\_PATH C头文件搜索路径

CPATH C头文件附加搜索路径

CPLUS\_INCLUDE\_PATH C++头文件附加搜索路径

LIBRARY\_PATH 链接时库文件搜索路径

LD\_LIBRARY\_PATH 运行时库文件搜索路径

四.使用外部库

1.头文件

作用：声明外部变量和函数

extern int a;

int add(int ,int);

定义宏，类型别名，自定义类型

#define PI 3.1415926

typedef unsigned int UINT

struct Student{};

公共头文件

#include <stdio.h>

#include <stdlib.h>

头文件卫士：防止重定义

#ifndef \_HEAD\_H\_

#define \_HEAD\_H\_

#endif

gcc -l ：编译时指定头文件的附加搜索路径

#include <> 包含系统指定目录的头文件

详细说明见代码 /头文件/add.h ,add.c main.c

2.库文件：预先编译好的函数集合

静态库(.a)：程序在编译连接的时候把库代码链接到可执行文件中，程序运行的时候不再需要动态库

动态库（.so或.sa）：程序在运行的时候才去链接共享库代码，多个程序共享使用库的代码

一点简单说明：

一个共享库链接的可执行文件仅仅包含它所用到的函数入口地址的一个表，而不是外部函数所在目标文件的整个机器码

在可执行文件开始运行以前，外部函数的机器码由操作系统从磁盘上的该共享库中复制到内存中，这个过程称之为动态链接

共享库可以在多个程序间共享，所以动态链接使得可执行文件更小，节省了磁盘空间，操作系统采用虚拟内存机制允许物理内存中的一份共享库被用到该库的所有进程共用

生成静态库

ar rcs abc.a hello.o

ar是gnu归档文件，rcs表示（replace and create）

使用静态库

ar工具的使用

ar [选项] 静态库名 目标文件列表

-r 将目标文件插入到静态库中，若已经存在则更新

-q 将目标文件追加到静态库尾部

-d 从静态库中删除指定的目标文件

-t 列表显示静态库中的目标文件

-x 将静态库展开为目标文件

调用静态库：

gcc -Wall main.c libhello.a -o main

gcc -L. -Wall main.o -o main -lhello 链接库文件的方式生成可执行文件

-L.指定库文件文件目录为当前目录

库搜索路径

C\_INCLUDE\_PATH（头文件路径）,LIBRARY\_PATH（库文件路径）

从左到右搜索-I -L指定的目录

由环境变量指定的目录

由系统指定的目录

生成共享库

shared:生成共享库格式

fPIC:产生位置无关码

库名规则：libxxx.so

如：gcc -shared -fPIC hello.o -o libhello.so

PIC：位置无关码，可执行程序加载共享库时，可将其映射到其地址空间的任何位置

-fPIC ：大模式，代码量大，速度慢，所有平台都支持

-fpic : 小模式，代码量小，速度快，仅一部分平台支持

例：1.编译

gcc -c -fpic calc.c

gcc -c -fpic show.c

2.生成动态库

gcc -shared calc.o show.o -o libmath.so

加载动态库

使用共享库

编译选项

-l 链接共享库，只要库名，去掉lib和版本号

-L 链接库所在的路径

gcc main.o -o main -L. -lhello

动态加载

#include <dlfcn.h>

void \* dlopen(const char\* filename ,int flag);

filename:若只给共享库文件名，则通过LD\_LIBRARY\_PATH环境变量搜索共享库，如给共享库路径，则按照路径加载，不使用环境变量

flag : RTLD\_LAZY:延迟加载，什么时候使用共享库，再什么时候加载

RTLD\_NOW:立即加载，成功返回库句柄，失败返回NULL

获取函数地址

void dlsym(void\* handle,//共享库句柄

const char\* symbol//函数名）

成功返回函数地址，失败返回NULL

卸载共享库

int dlclose(

void\* handle//共享库句柄

);

成功返回0，失败返回非零

获取错误信息

char\* dlerror(void);

返回错误信息字符串，没有错误返回NULL

详细用法见程序lcoal.c

常用辅助工具

nm 查看目标文件，可执行文件，静态库，共享库中符号列表

ldd 查看可执行文件或共享库的动态依赖

ldconfig 事先把共享库的路径信息写到/etc/ld.so.conf配置文件中，ldconfig根据

配置文件生成/etc/ld.so.cache缓冲文件，并将该缓冲读入内存，提高动态库的加载效率

strip:通过删除符号表和调试信息，给目标文件，可执行文件，库文件减肥

objdump:对机器指令做反汇编

错误处理

1.通过函数的返回值表示错误

1）返回合法值表示成功，返回非法值表示失败

2）返回有效指针表示成功，返回空指针表示失败

3）返回0表示成功，返回-1表示失败，如果有需要返回给调用者的数据，可以通过指针型参数输出

4）如果一个函数永远不会失败，也没有数据提供给调用者，可以没有返回值，如exit函数

2.通过错误码返回函数失败的原因，需要包含头文件#include <errno.h>

1)直接通过errno全局变量获取错误原因

2)将errno转换成一个字符串（错误信息）

a.包含#include<string.h> 头文件，使用char\* strerror(int errnum)函数输出

printf(“%s\n”,strerror(error));所有的错误码都不为0，所以没有错误，错误码为0

b.包含#include<stdio.h> 头文件，使用void perror（const char\* s）函数输出

perror(“malloc”)

环境变量

1.环境表

每个进程都会接受到一张环境表，是一个以NULL指针结尾的字符指针数组。

全局变量environ保存了环境表的首地址

main函数的第三个参数就是环境表的

ens 名令可以显示全部环境变量

2.环境变量函数

头文件：#include <stdlib.h>

getenv 根据name获得value

putenv 以<name>=<value>形式设置环境变量

如果name不存在，就添加，存在就修改原理来的值

setenv 根据name设置value，若name已存在，根据参数决定是否覆盖原value

unsetenv 删除环境变量

clearen 清空环境变量，environ == NULL