Linux C语言开发工具链

Version 1.0

西安電子科技大學

本地开发 vs 交叉平台开发

- **本地开发**：
  一般软件的开发属于本地开发，也就是说开发软件的系统与运行软件的系统是相同的。
- **交叉平台开发**：
  嵌入式系统开发属于交叉平台开发，也就是说开发软件的系统与运行软件的系统不同。
  - **宿主机**：开发软件的平台，称为宿主机，往往是通用电脑
  - **目标机**：运行软件的平台，称为目标机，在这里是嵌入式系统。

西安電子科技大學

（嵌入式）开发工具链

- 掌握嵌入式开发工具链的使用是进行嵌入式开发的前提条件之一
- 与主流开发工具类似，嵌入式交叉开发工具也包括
  1. **编辑器**：常用编辑器vim，gvim, emacs, gedit, eclipse
  2. 交叉编译器，编译器能够把一个源程序编译生成一个由机器语言构成的可执行程序。虽然交叉编译器本身也在主机上运行，但编译生成的不是主机认识的机器语言，而是目标机能够识别的机器语言。
  3. 调试工具，即能够对执行程序进行源码或汇编级调试的软件
  4. 项目管理工具，用于协助多人开发或大型软件项目的管理的软件

GNU tools

- GNU tools和其他一些优秀的开源软件可以完全覆盖上述类型的软件开发工具。为了更好的开发嵌入式系统，需要熟悉如下一些软件
  - vi，vim
  - gcc
  - gdb
  - make
  - binutils—辅助GCC的主要软件
  - Cvs
- 善于调试代码，解决Bug是优秀程序员的必备能力

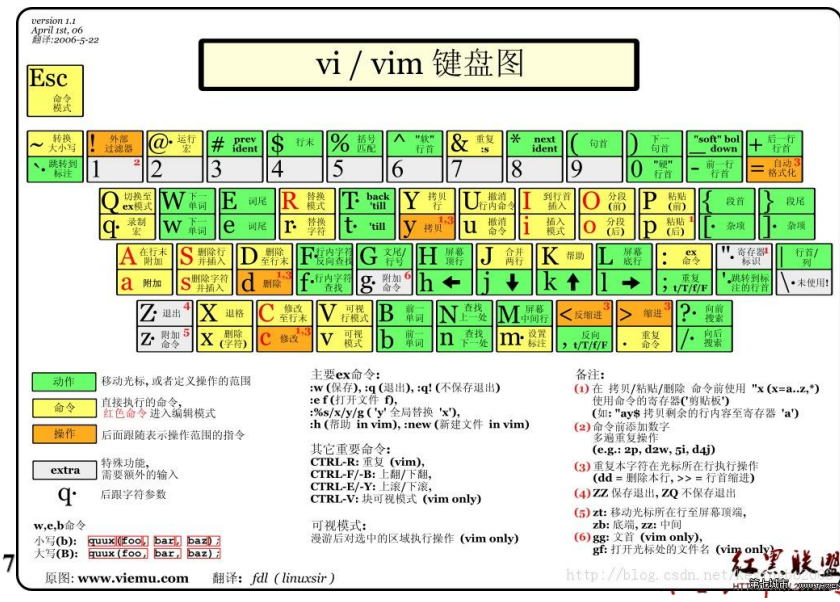
vim编辑器

- **Vim** is a highly configurable text editor built to enable efficient text editing.
- "programmer's editor "
- charityware. GPL-compatible
- 纯控制台方式的编辑器。(只有6KB大小)
- gVim是图形用户界面版的vim，有windows版
- 高手中的vim：手不离键盘。

vim编辑器

- 标准模式(一般模式)：键盘输入作为命令对待，不回显。
- 编辑模式(插入模式)：键盘输入作为文本正文。
- 命令行模式(底行模式)：键盘输入作为命令对待，在最底行回显，输入“Enter”才有效。
- 可视模式：可以通过移动光标选择文本。

vim编辑器



vim的进入、保存和退出

- 进入vim可以直接在命令终端下键入vim < 文件名 >，vim可以自动载入所要编辑的文件或是开启一个新的文件。如在shell中键入vim hello.c（新建文件）则可进入vim环境。进入vi后屏幕左方会出现波浪符号，凡是具有该符号就代表此列目前是空的。此时进入的是命令行模式。
- 要退出vim可以在命令行模式下键入“:q”（不保存并退出）或“:q!”（不保存并强制退出）或“:wq”（保存并退出）指令则是保存之后再离开（注意冒号）。

vim的进入、保存和退出

命令类别	命令	说明
编辑	:e *filename*	编辑文件名为*filename*的文件。若这个文件不存在，则会开启一个名为*filename*的新文件的编辑
保存	:w	保存文件，文件应已有名字
	:w *filename*	以文件名*filename*保存文件
退出	:q	退出，如果文件已修改则不能退出
	:q!	不保存强行推出，无论文件是否被修改
	:wq	保存后退出

vim光标的移动

命令类别	命令	说明
基本操作	h, j, k, l	分别等同于左方向键、下方向键、上方向键、右方向健
字操作	w	移至下一个单词的字首
	e	移至下一个单词的字尾
	b	移至上一个单词的字首
行操作	0	移至行首
	$	移至行尾
	G	移至文件尾部
	gg	移至文件首部
	H	移至当前屏幕顶部
	M	移至当前屏幕中间行的行首
	L	移至当前屏幕底部最后一行的行首
	n-	向上移动n行
	n+	向下移动n行
	nG	移至第n行
页操作	Ctrl + f	屏幕往“上”翻动一页，等同于PageUp
	Ctrl + b	屏幕往“下”翻动一页，等同于PageDown
	Ctrl + u	屏幕往“上”翻动半页
	Ctrl + d	屏幕往“下”翻动半页

vim文本编辑

命令类别	命令	说明
修改	r	修改光标所在的字符，键入r后直接键入待修改字符
	R	进入取代状态，在光标所指定的位置修改字符，该替代状态直到按下ESC键才结束
复制	yy	复制光标所在行
	*n*yy	复制光标所在行开始的*n*行，如3yy表示复制三行
	y^	复制光标至行首
	y$	复制光标至行尾
	yw	复制一个字（单词）
	yG	复制光标文件尾
	y1G	复制光标文件首
粘贴	p	粘贴至光标后
	P	粘贴至光标前
删除	x	删除光标所在位置的一个字符
	X	删除光标所在位置的前一个字符
	s	删除光标所在的字符，并进入输入模式
	S	删除光标所在的行，并进入输入模式
	dd	删除光标所在的行
	*n*dd	从光标所在行开始向下删除n行
	D	删除至行尾，等同于d$
	dG	删除至文件尾部
	d1G	删除至文件首部，等同于dgg
恢复	u	撤销上一步的操作，可以多次撤销
	U	在光标离开之前，恢复所有的编辑操作
	Ctrl + r	返回至撤销操作之前的状态

Vim查找与替换

vim的查找和替换功能都支持正则表达式，可以匹配非常复杂的关键字，功能非常强大。

类别	命令	说明
查找	/<要查找的字符>	向下查找要查找的字符
	?<要查找的字符>	向上查找要查找的字符
	n	继续查找
	N	反向查找
替换	:[range]s/pattern/string/[c,e,g,i]	range：指定查找的范围。例如1,$指替换范围从第0行到最后一行； s：指转入替换模式； pattern:指要被替换的字符串，可以用正则表达式； string:指替换的字符串； c：每次替换前询问； e：不显示错误； g：强制整行替换； i：不区分大小写

Vim的使用技巧重在积累

GCC

- 很多人认为GCC只是一个C编译器，
  其实GCC = GNU Compiler Collection
- 目前，GCC可以支持多种高级语言，如
  - C、C++
  - ADA
  - Object C
  - JAVA
  - Fortran
  - PASCAL

GCC下的工具

- gcc — 符合ISO等标准的C编译器
- g++ — 基本符合ISO标准的C++编译器
- gcj — GCC的java前端
- gnat — GCC的GNU ADA 95前端

GNU Tools—gcc

- gcc是一个强大的工具集合，它包含了预处理器(cpp)、编译器(cc)、汇编器(as)、链接器(ld)等组件。它会在需要的时候调用其他组件。输入文件的类型和传递给gcc的参数决定了gcc调用具体的哪些组件。
- 对于开发者，它提供的足够多的参数，可以让开发者全面控制代码的生成，这对嵌入式系统级的软件开发非常重要。

GCC

- **GCC又是一个交叉平台编译器，支持的硬件平台很多，如alpha、arm、avr、hppa、i386、m68k、mips、powerpc、sparc、vxworks、x86_64、MS Windows、OS/2等等。它能够在当前CPU平台上为多种不同体系结构的硬件平台开发软件，因此尤其适合在嵌入式领域的开发编译。**

GCC的简介

- GCC使用的基本语法为：

  gcc [option] filename

GCC 编译参数

- -E　只预处理，不编译汇编和连接
- -S　只编译，不汇编和连接
- -c　只编译和汇编，不连接
- -o　指定输出文件
- -lname　链接名称为libname.a/ libname.so的库
- -I　　指定头文件路径
- -L　指定链接时的函数库的查找路径
- -w　关闭显示警告
- -Wall　启动所有编译警告

西安電子科技大學

GCC 编译参数

- -g　　编译时加入调试参数（-ggdb）
- -pg　　产生gprof所用的信息
- -On　　优化
- -ansi
- -std=
  - c89 / c99 /gnu89 /gnu99
- -static　　只链接静态库
- -shared　　尽可能链接动态库

gcc使用举例（1）

```
//gcctest.c

#include <stdio.h>

int main()
{
        int i,j;
        i=0;
        j=0;
        i=j+1;
        printf("Hello World!\n");
        printf("i=j+1=%d\n",i);
}
```

gcc使用举例（2）

- 编译和运行

```
[donger@donger gcctest]$ ls
gcctest.c
[donger@donger gcctest]$ gcc -o gcctest gcctest.c
[donger@donger gcctest]$ ls
gcctest  gcctest.c
[donger@donger gcctest]$ ./gcctest
Hello World!
i=j+1=1
[donger@donger gcctest]$
```

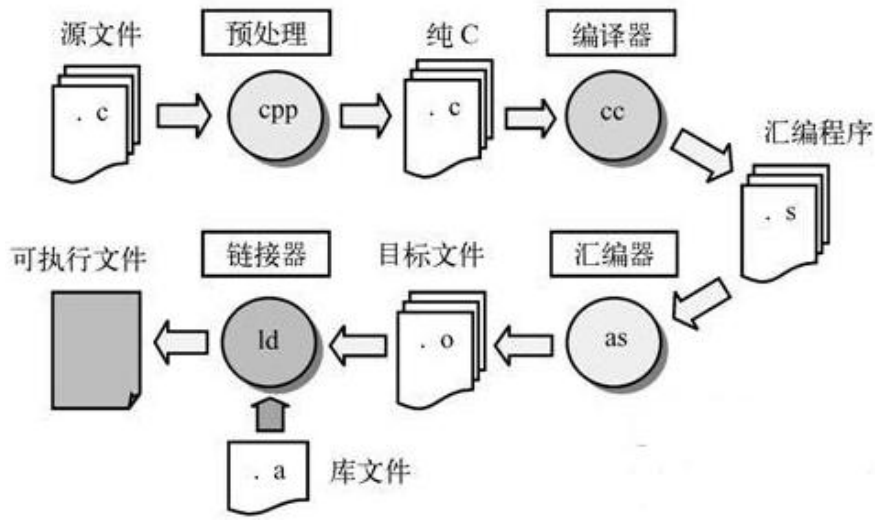
编译

运行

gcc的编译过程

- 一般情况下，c程序的编译过程为
  1、预处理（Pre-Processing）
  2、编译（Compiling）
  3、汇编（Assembling）
  4、链接（Linking）

后 缀 名	所对应的语言	编 译 流 程
.c	C原始程序	预处理、编译、汇编
.C/.cc/.cxx	C++原始程序	预处理、编译、汇编
.m	Objective-C原始程序	预处理、编译、汇编
.i	已经过预处理的C原始程序	编译、汇编
.ii	已经过预处理的C++原始程序	编译、汇编
.s/.S	汇编语言原始程序	汇编
.h	预处理文件（头文件）	（不常出现在指令行）
.o	目标文件	链接
.a/.so	编译后的库文件	链接



1、预处理

- 预处理：使用-E参数
  输出文件的后缀为“.i”

gcc –E –o gcctest.i gcctest.c

关于预处理

- **文件包含**：#include是最为常见的预处理，主要将指定的文件代码组合到源程序代码之中。
- 条件编译：#if、#ifdef、#ifndef、#if defined、#endif、#undef等也是常用的预处理，主要是在编译时进行选择性编译，有效地控制版本和编译范围，防止对文件的重复包含等重要功能。
- **布局控制**：#progma功能因后面的参数不同而不同，例如使用#progma pack(1)可以内存变量的1个字节对齐，使得结构变量的成员分配到连续的内存块，等价于__attribute__((packed))。
- **宏替换**：#define的主要功能是定义符号常量、函数功能、重新命名、字符串符号的拼接等各种功能。
- **其他的预处理有**：#line是用于修改预定义宏__LINE__（当前所在的行号）和__FILE__（当前源文件的文件名）；#error / #warning分别用于输出一个错误/警告信息；等等。

2、编译成汇编代码

- 可以**直接编译到汇编代码**

输出文件的后缀为“.s”

gcc –S gcctest.c

- 利用预处理阶段生成的结果

gcc –S –o gcctest.s gcctest.i

3、编译成目标代码

- 直接编译成目标代码

  gcc –c gcctest.c

- 利用预处理阶段生成的结果

  gcc –c –o gcctest.o gcctest.i

- 使用汇编器生成目标代码

  as –o gcctest.o gcctest.s

4、编译成执行代码

- 直接生成执行代码

  gcc –o gcctest gcctest.c

- 利用汇编阶段生成的结果

  gcc –o gcctest gcctest.o

- 执行代码

  ./gcctest

- 最简单的程序也用到了系统函数库（C运行时库）
- 理解这个编译的过程是撰写Makefile的基础

优化编译

- 优化编译选项有：
  - -O0
    缺省情况，不优化
  - -O1
  - -O2
  - -O3

  } 不同程度的优化
  - 等等

gcc的优化编译举例（1）

```
//mytest.c
#include <stdio.h>
#include <math.h>

int main()
{
        int i,j;
        double k,k1,k2,k3;
        k=0.0; k1=k2=k3=1.0;
        for (i=0;i<50000;i++)
                for (j=0;j<50000;j++)
                {
                        k+=k1+k2+k3;
                        k1 += 0.5;
                        k2 += 0.2;
                        k3 = k1+k2;
                        k3 -= 0.1;
                }
        return 0;
}
```

```
[donger@donger gcctest]$ ls
gcctest.c  mytest.c
[donger@donger gcctest]$ gcc -O0 -o m0 mytest.c
[donger@donger gcctest]$ gcc -O1 -o m1 mytest.c
[donger@donger gcctest]$ gcc -O2 -o m2 mytest.c
[donger@donger gcctest]$ gcc -O3 -o m3 mytest.c
[donger@donger gcctest]$ ls
gcctest.c  m0  m1  m2  m3  mytest.c
[donger@donger gcctest]$
```

不同的优化编译选项

使用time命令统计程序的运行

```
[donger@donger gcctest]$ ls
gcctest.c  m0  m1  m2  m3  mytest.c
[donger@donger gcctest]$ time ./m3

real    0m2.756s
user    0m2.658s
sys     0m0.042s
[donger@donger gcctest]$ time ./m2

real    0m2.733s
user    0m2.643s
sys     0m0.037s
[donger@donger gcctest]$ time ./m1

real    0m1.829s
user    0m1.767s
sys     0m0.022s
[donger@donger gcctest]$ time ./m0

real    0m40.808s
user    0m39.632s
sys     0m0.337s
[donger@donger gcctest]$
```


gcc警告和出错选项

选　项	含　义
-ansi	支持符合ANSI标准的C程序
-pedantic	允许发出ANSI C标准所列的全部警告信息
-pedantic-error	允许发出ANSI C标准所列的全部错误信息
-w	关闭所有警告信息
-Wall	允许发出gcc提供的所有有用的报警信息
-werror	把所有的警告信息转化为错误信息，并在警告发生时终止编译过程

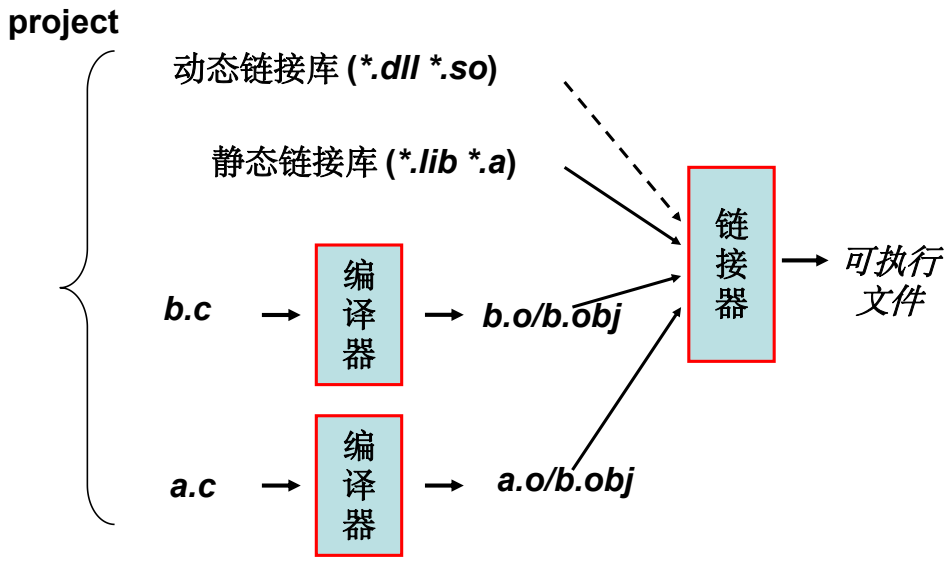
gcc警告和出错选项

选　项	含　义
-mcpu=type	针对不同的CPU使用相应的CPU指令。可选择的type有i386、i486、pentium及i686等
-mieee-fp	使用IEEE标准进行浮点数的比较
-mno-ieee-fp	不使用IEEE标准进行浮点数的比较
-msoft-float	输出包含浮点库调用的目标代码
-mshort	把int类型作为16位处理，相当于short int
-mrtd	强行将函数参数个数固定的函数用ret NUM返回，节省调用函数的一条指令

编译链接多个文件

- **gcc –o program t1.c t2.c t3.c**
- **gcc –o program t1.o t2.o t3.o**

分别编译与链接



链接器的主要工作

1. 将分散的数据和机器代码**收集并合成**一个单一的可加载并可执行的文件；
2. **符号解析**：由多个程序模块(源程序)构建一个可执行程序时，模块之间的相互引用通过符号进行。程序也可以通过符号来引用代码库(lib库)中的功能。符号解析就是将符号引用和符号定义关联起来。
3. **地址重定位**：编译器产生的各个目标文件(obj文件)中数据和代码的地址一般都是从0开始。因此如果一个程序包含多个目标文件时就会产生地址重叠。重定位就是为每个目标文件重新定义加载地址，并修改相应的代码和数据以反映这种变化。

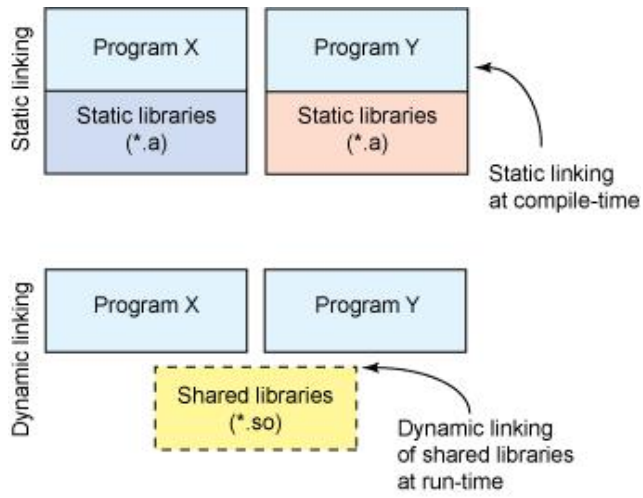
预编译函数库：静态库、动态库

- 预编译的函数库：模块化、可重用性强、编译速度快、保护知识产权
- 静态库（.a后缀）是一系列的目标文件的归档，静态链接时，静态库中的目标函数体会被复制到程序的可执行文件中去
- 动态库（libname.so[.major.minor.release]）不具有“复制”操作，程序运行时根据需要载入内存，且动态地将函数调用与函数体关联到一起。动态库在内存中只有一份拷贝，因此也称为共享库。易于升级。

使用动态链接库的优点

1. 动态链接库文件与可执行文件独立，只要输出接口不变，更换动态链接库文件不会对可执行文件造成任何影响，因而极大地提高了可维护性和可扩展性。
2. 被多个应用程序共享时，在内存中只有一份拷贝，因而更加节省内存
3. 可以在多种编程语言之间共享代码

静态库与动态库



如何使用静态库

```
// hello.c
#include <stdio.h>
void hello(char *name) {
        printf("Hello %s!\n", name);
}
```

如何使用静态库

```
/*share lib test program test.c*/
 #include<stdio.h>
 #include<stdlib.h>
 int main(int argc, char *argv[])
  { if (argc<2) { printf("You MUST input
parameters,ex> %s someword\n",argv[0]);
exit(1);}
 printf("======static======\n");
 hello(argv[1]);
 return 0; }
```


Linux中如何使用静态库

如何生成静态库：

- **gcc -c hello.c**
- **ar -crsv libhello.a hello.o**

如何链接静态库

- **gcc -o test test.c –lhello**
- **gcc -o test test.c -L. –lhello**
- **gcc test.c libhello.a -o test**

Linux中如何使用动态库

如何生成动态库

- gcc -fPIC -c hello.c (生成hello.o文件)
- gcc -shared -o libhello.so hello.o (生成hello.so)

如何链接动态库（加载时链接）

- gcc -o test test.c –lhello

如何找到动态库

- /usr/lib和/lib目录
- 环境变量： LD_LIBRARY_PATH
- 配置文件/etc/ld.so.conf

课后练习 Exercises after class

利用C语言实现阶乘函数f(n) = n!,

•将f(n)封装到静态库中，然后再编写另外一个C程序调用f(n)。

•将f(n)封装到动态库中，然后再编写另外一个C程序调用f(n)。

4月20日前将源码、静态库文件、动态库文件打包发送到：2530182725@qq.com, （助教：何立志）

邮件主题：学号姓名第二次作业，程序打包作为附件发送，附件命名：学号姓名.zip 或 学号姓名.rar