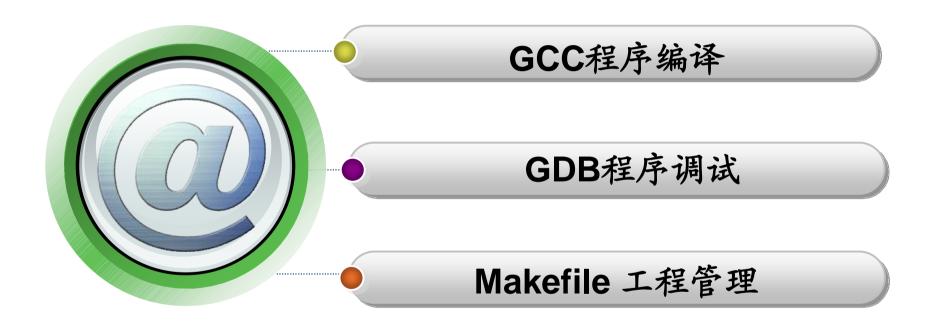


# LINUX 应用程序设计基础

版权声明:本课件及其印刷物、视频的版权归成都国嵌信息技术有限公司所有,并保留所有权力:任何单位或个人未经成都国嵌信息技术有限公司书面授权,不得使用该课件及其印刷物、视频从事商业、教学活动。已经取得书面授权的,应在授权范围内使用,并注明"来源:国嵌"。违反上述声明者,我们将追究其法律责任。

### **Contents**

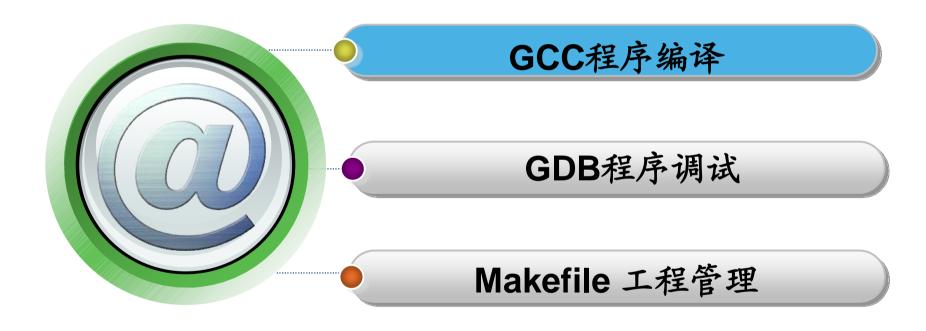






#### **Contents**









### Linux系统下的gcc(GNU C

Compiler)是GNU推出的功能强大、性能优越的多平台编译器,是GNU的代表作之一。gcc可以在多种硬体平台上编译出可执行程序,其执行效率与一般的编译器相比平均效率要高20%~30%。





GCC编译器能将C、C++语言源程序、 汇编程序编译、链接成可执行文件。 在Linux系统中,可执行文件没有统一的 后缀,系统从文件的属性来区分可执行 文件和不可执行文件。





使用GCC编译程序时,编译过程可以被细分为四个阶段:

- ∨预处理(Pre-Processing)
- ∨编译(Compiling)
- ∨汇编(Assembling)
- ∨链接(Linking)



## 文件类型



#### Gcc通过后缀来区别输入文件的类别:

- V.C为后缀的文件: C语言源代码文件
- V .a为后缀的文件: 是由目标文件构成的库文件
- V.C,.cc或.cxx 为后缀的文件: 是C++源代码文件
- ∨ .h为后缀的文件: 头文件
- ∨ .i 为后缀的文件: 是已经预处理过的C源代码文件
- ∨ .ii为后缀的文件: 是已经预处理过的C++源代码文件
- ∨ .o为后缀的文件: 是编译后的目标文件
- ∨ .s为后缀的文件: 是汇编语言源代码文件
- ∨ .S为后缀的文件: 是经过预编译的汇编语言源代码文件。



## 起步 (演示)



```
hello.c:
  #include <stdio.h>
  int main(void)
  printf (Hello world!\n);
  return 0;
  编译和运行这段程序:
  # gcc hello.c -o hello
  # ./hello
  输出: Hello world!
    嵌入式Linux技术咨询QQ号: 550491596
     嵌入式Linux学习交流QQ群: 65212116
```



## 基本用法



gcc最基本的用法是:

gcc [options] [filenames]

options: 编译器所需要的编译选项

filenames: 要编译的文件名。







gcc编译器的编译选项大约有100多个, 其中多数我们根本就用不到,这里只介 绍其中最基本、最常用的参数。

V-o output\_filename:确定可执行文件的名称为output\_filename。如果不给出这个选项,gcc就给出预设的可执行文件a.out。(演示)





- V-c: 只编译,不连接成为可执行文件,编译器只是由输入的.c等源代码文件生成.o为后缀的目标文件。
- V-g:产生调试工具(GNU的gdb)所必要的符号信息,要想对编译出的程序进行调试,就必须加入这个选项。





- V-O,对程序进行优化编译、链接,采用这个选项,整个源代码会在编译、连接过程中进行优化处理,这样产生的可执行文件的执行效率可以提高,但是,编译、连接的速度就相应地要慢一些。
- V-O2,比-O更好的优化编译、连接,当然整个编译、连接过程会更慢。



## 编译选项(optimize.c)



```
#include <stdio.h>
int main(void)
    double counter:
    double result;
    double temp;
    for (counter = 0; counter < 2000.0 * 2000.0 * 2000.0 / 20.0 + 2020;
     counter += (5 - 1) / 4) {
      temp = counter / 1979;
      result = counter:
    printf(Result is %lf\\n, result);
    return 0;
```





 gcc optimize.c -o optimize time ./optimize

2. gcc –O optimize.c -o optimize time ./optimize

对比两次执行的输出结果不难看出,程序的性能的确得到了很大幅度的改善

嵌入式Linux技术咨询QQ号: 550491596

嵌入式Linux学习交流QQ群: 65212116





-ldirname: 将dirname所指出的目录加入到程序头文件目录列表中。

C程序中的头文件包含两种情况:

- #include <A.h>
- #include "B.h"

对于<>, 预处理程序cpp在系统预设的头文件目录(如 /usr/include)中搜寻相应的文件;而对于"",cpp在当前目录中搜寻头文件。这个选项的作用是告诉cpp,如果在当前目录中没有找到需要的文件,就到指定的dirname目录中去寻找。

例: gcc foo.c -l /home/include -o foo





-Ldirname: 将dirname所指出的目录加入到 库文件的目录列表中。在默认状态下,连接 程序Id在系统的预设路径中(如/usr/lib)寻找所 需要的库文件,这个选项告诉连接程序,首 先到-L指定的目录中去寻找,然后再到系统 预设路径中寻找。





∨ -Iname: 在连接时,装载名字为 "libname.a"的函数库,该函数库位于 系统预设的目录或者由-L选项确定的目录下。例如,-Im表示连接名为 "libm.a"的数学函数库。

例: gcc foo.c -L /home/lib -lfoo -o foo





∨ -static: 静态链接库文件

例: gcc -static hello.c -o hello

库有动态与静态两种,动态通常用.so为后缀,静态用.a为后缀。例如: libhello.so libhello.a。当使用静态库时,连接器找出程序所需的函数,然后将它们拷贝到可执行文件,一旦连接成功,静态程序库也就不再需要了。然而,对动态库而言,就不是这样,动态库会在执行程序内留下一个标记'指明当程序执行时,首先必须载入这个库。由于动态库节省空间,linux下进行连接的 缺省操作是首先连接动态库。

演示: 静态链接与动态链接可执行文件大小比较



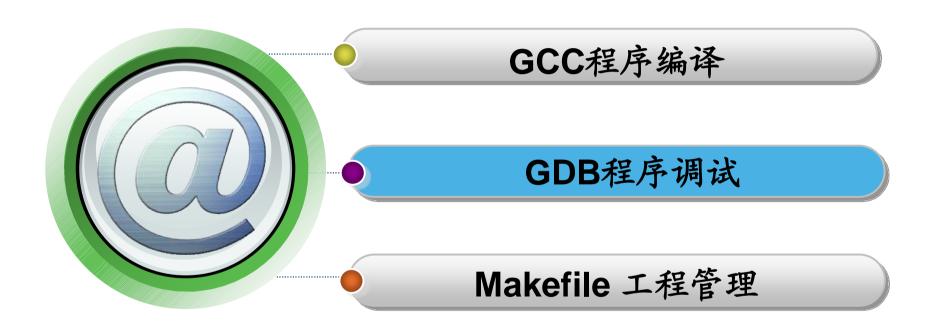


- ∨ -Wall: 生成所有警告信息
- ∨ -w: 不生成任何警告信息
- -DMACRO: 定义 MACRO 宏,等效 于在程序中使用#define MACRO



### **Contents**









GDB是GNU发布的一款功能强大的程序调试工具。GDB主要完成下面三个方面的功能:

- 1、启动被调试程序。
- 2、让被调试的程序在指定的位置停住。
- 3、当程序被停住时,可以检查程序状态(如变量值)。



## 起步(tst.c)



```
#include <stdio.h>
void main()
{
    int i;
    long result = 0;
    for(i=1; i<=100; i++)
    {
       result += i;
    }
    printf("result = %d \n", result );
}</pre>
```



## GDB快速进阶



1.编译生成可执行文件:

gcc -g tst.c -o tst

2.启动GDB

gdb tst

- 3. 在main函数处设置断点 break main
- 4. 运行程序 run



## GDB快速进阶



- 5. 单步运行 next
- 6. 继续运行 continue



## 启动GDB



1. gdb 调试程序名

例: gdb helloworld

2. gdb file 调试程序名



## GDB命令(演示)



- ∨ list(l) 查看程序
- ∨ break(b) 函数名 在某函数入口处添加断点
- ∨ break(b) 行号 在指定行添加断点
- ∨ break(b) 文件名:行号 在指定文件的指定行添 加断点
- ∨ break(b) 行号 if 条件 当条件为真时,指定行号 处断点生效,例b5 if i=10,当i等于10时第5行 断点生效



## GDB命令(演示)



- v info break 查看所有设置的断点
- V delete 断点编号 删除断点
- ∨ run(r) 开始运行程序
- ∨ next(n) 单步运行程序(不进入子函数)
- ∨ step(s) 单步运行程序(进入子函数)
- ∨ continue(c) 继续运行程序



## GDB命令(演示)

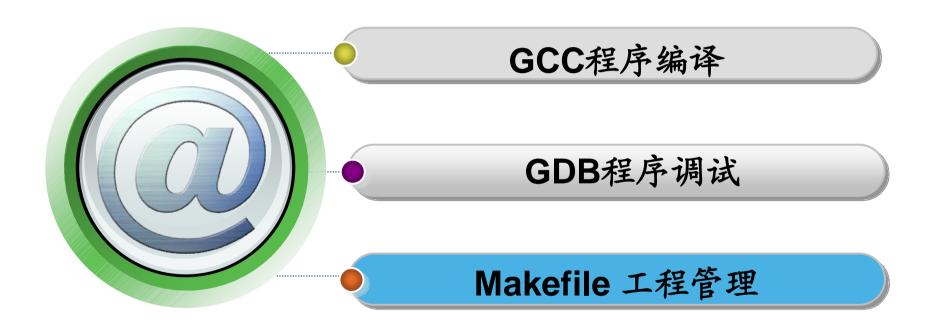


- ∨ print(p) 变量名 查看指定变量值
- V finish 运行程序,直到当前函数结束
- ∨ watch 变量名 对指定变量进行监控
- ∨ quit(q) 退出gdb



#### **Contents**







### **GNU** make



Linux程序员必须学会使用GNU make来构建和管理自己的软件工程。GNU 的make能够使整个软件工程的编译、链接只需要一个命令就可以完成。



### Makefile



make在执行时,需要一个命名为Makefile 的文件。Makefile文件描述了整个工程的 编译,连接等规则。其中包括:工程中的 哪些源文件需要编译以及如何编译;需要 创建那些库文件以及如何创建这些库文件、 如何最后产生我们想要得可执行文件。



## Makefile(例)



hello: main.o func1.o func2.o

gcc main.o func1.o func2.o -o hello

main.o: main.c

gcc -c main.c

func1.o: func1.c

gcc -c func1.c

func2.o: func2.c

gcc –c func2.c

.PHONY: clean

clean:

rm –f hello main.o func1.o func2.o

嵌入式Linux技术咨询QQ号: 550491596

嵌入式Linux学习交流QQ群: 65212116



## Makefile(术语)



规则:用于说明如何生成一个或多个目标文件,规则格式如下:

targets: prerequisites

command

目标 依赖 命令

main.o: main.c

gcc -c main.c

目标?依赖?命令?

\*\*命令需要以【TAB】键开始\*\*

嵌入式Linux技术咨询QQ号: 550491596

嵌入式Linux学习交流QQ群: 65212116



## 目标



在Makefile中,规则的顺序是很重要的,因 为,Makefile中只应该有一个最终目标,其 它的目标都是被这个目标所连带出来的,所 以一定要让make知道你的最终目标是什么。 一般来说,定义在Makefile中的目标可能会 有很多,但是第一条规则中的目标将被确立 为最终的目标。

## 文件名



make命令默认在当前目录下寻找名字为 makefile或者Makefile的工程文件,当名字 不为这两者之一时,可以使用如下方法指 定:

make -f 文件名



## 伪目标



Makefile中把那些没有任何依赖只有执行动作的目标称为"伪目标"(phony targets)。

.PHONY: clean

clean:

rm –f hello main.o func1.o func2.o

".PHONY"将"clean"目标声明为伪目标

嵌入式Linux技术咨询QQ号: 550491596

嵌入式Linux学习交流QQ群: 65212116



### 变量



hello: main.o func1.o func2.o gcc main.o func1.o func2.o -o hello

思考1:如果要为hello目标添加一个依赖,如:func3.o,该如何修改?

答案1:

hello: main.o func1.o func2.o func3.o gcc main.o func1.o func2.o func3.o -o hello

答案2:

obj=main.o func1.o func2.o func3.o

hello: \$(obj)

gcc \$(obj) -o hello

嵌入式Linux技术咨询QQ号: 550491596

嵌入式Linux学习交流QQ群: 65212116



### 变量



#### 在makefile中,存在系统默认的自动化变量

∨ \$^: 代表所有的依赖文件

∨ \$@: 代表目标

V \$<: 代表第一个依赖文件

例:

hello: main.o func1.o func2.o gcc main.o func1.o func2.o -o hello

=>>

hello: main.o func1.o func2.o gcc \$^ -o \$@



## 杂项



Makefile中"#"字符后的内容被视作注释。

hello: hello.c

@gcc hello.c -o hello

@:取消回显 (演示)

