**实验报告五 常用开发工具**

姓名：叶倩琳

班级：软工1706班

学号：201706061330

提交日期：2019年12月28日

[实验环境：Ubuntu]

1. **实验目的**

1.掌握C语言编译系统gcc的基本用法。

2.掌握gdb调试工具的基本用法。

3. 理解make程序维护工具的功能，学会编制makefile文件。

1. **实验内容**

1.利用gce命令编译C语言程序，使用不同选项，观察并分析显示结果。

2.用gdb命令调试一个编译后的C语言程序。

3.编写一个由多个文件构成的C语言程序，编制makefile，然后运行make工具进行维护。

**三、主要实验步骤**

1.改写例6.1，使用下列选项对它进行编译:-I，-D，-E，-c，-o，-l。

（1）vi p180.c

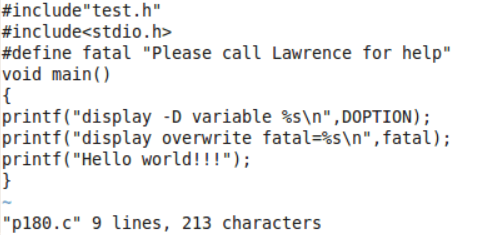


图6-1-1 源程序

1. gcc p180.c

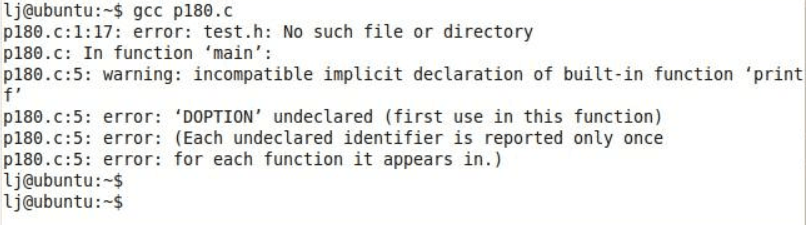


图6-1-2 gcc p180.c

（3）test.h放在/home/lj/c目录下，用gcc –I /home/lj/c p180.c结果如下：

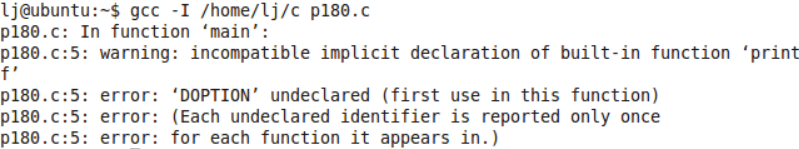


图6-1-3 gcc –I /home/lj/c p180.c

（4）gcc –I /home/lj/c –D DOPTION=’"testing -D"’–D fatal –E p180.c

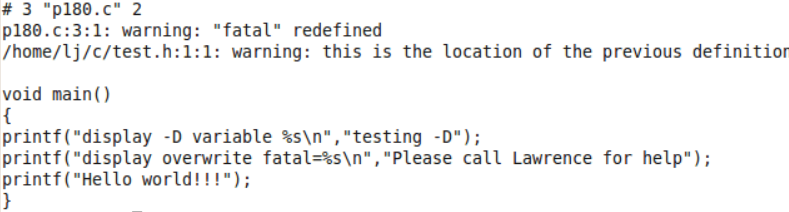


图6-1-4 gcc–I/home/lj/c–D DOPTION=’"testing -D"’–D fatal –E p180.c

（5）用注释符括起文件包含行（即#include”test.h”）,gcc –D DOPTION=’”testing -D”’ p180.c –o abc

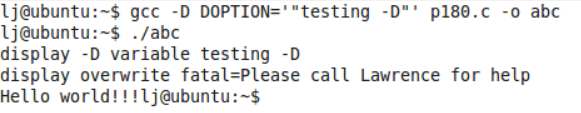


图6-1-5 cc –D DOPTION=’”testing -D”’ p180.c –o abc

2.完成对习题6.5的调试。

3.完成对习题6.6的调试。

4.完成对习题6.9的编制，并使用make命令进行维护。

**四、实验过程**

6.1 gcc编译过程一般分为那几个阶段？各阶段的主要工作是什么？

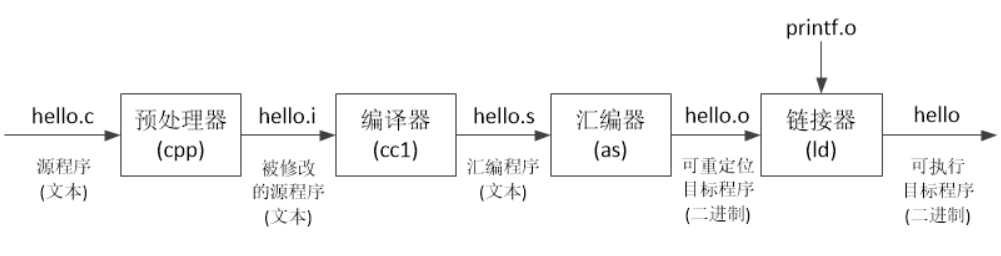
答：

图6-2-1 gcc编译的四个过程

gcc -E hello.c -o hello.i

gcc –S hello.i –o hello.s

gcc –c hello.s –o hello.o

gcc hello.o –o hello

（1）预处理阶段：读取C语言源文件，对其中以#开头的指令（伪指令）和特殊符号进行。

（2）编译阶段：对预处理之后的输出文件进行词法分析和语法分析，在确定各成分都符合语法规则后，将其“翻译”为功能等价的中间代码表示或汇编代码。

（3）汇编过程：把汇编语言代码翻译成目标机器代码。

（4）连接阶段：将一个文件中引用的符号（如变量或函数调用）与该符号在另一个文件中的定义连接起来，从而使有关的目标文件连成一个整体，最终成为可被操作系统执行的可执行文件。



图6-2-2 gcc命令的工作过程

6.2 对C语言程序进行编译时，针对以下情况应使用的编译命令行是什么？

（1）只生成目标文件，不进行连接。

（2）在预处理后的输出中保留源文件中的注释。

（3）将输出写到file指定的文件中。

（4）指示编译程序在目标代码中加入供调试程序gdb使用的附加信息。

（5）连接时搜索由library命名的库。

答：（1） -c

（2） -C

（3） -o file

（4） -g

（5） -llibrary

表6-1-1 gcc常用参数

| **选项名** | **作用** |
| --- | --- |
| E | 激活预处理；头文件、宏等展开（.i文件） |
| S | 激活预处理、编译；生成汇编代码（.s文件） |
| c | 激活预处理、编译、汇编；生成目标文件（.o文件） |
| o | 生成目标 |
| Wall | 打开编译告警（所有） |
| g | 嵌入调试信息，方便gdb调试 |
| l lib | 链接 lib 库 （这里是小写 L ） 相当于 C++ #pragma comment(lib, “xxx.lib”) |
| I dir | 增加 include 目录 (这里是大写 i ) 头文件路径 |
| L Dir | 增加 lib 目录 （编译静态库和动态库） |

6.3 通常，程序中的错误按性质分为哪三种？

答：

（1）编译错误，即语法错误。这是在编译阶段发生的错误，主要是程序代码中有不符合所用编程语言语法规则的错误。

（2）运行错误。这种错误在编译时发现不了，只在运行时才显现出来。

（3）逻辑错误。这种错误即使在运行时也不显示出来，程序能正常运行，但结果不对。

6.4 gdb主要帮助用户在调试程序时完成哪些工作？

答：

（1）启动程序，按用户要求影响程序的运行行为。

（2）使运行程序在指定条件处停止。

（3）当程序停止时，检查它出现了什么问题。

（4）动态改变程序的执行环境，这样就可以先纠正一个错误，然后再纠正其他错误。

6.5 调试下面的程序：

yql@DESKTOP-TRMDDFQ:~$ vi test5.c

i

/badprog.c错误地访问内存/

#include <stdio.h>

#include<stdlib.h>

int main(int argc,char \*\*argv)

{

char \*p;

int i;

p=malloc(30); **//加类型转换 p=(char\*)malloc(30)**

strcpy(p,”not 30 bytes”);

printf(“p=<%s>\n”,p);

if(argc==2){

if(strcmp(argv[1],”-b”)==0)

p[50]=’a’;

**//不要漏小括号，下标最大29，这里用p[0]=’a’**

else if(strcmp(argv[1],”-f”)==0){

free(p);

p[0]=’b’;

}

}

**//printf("%s\n",p); 输出p  
//free(p); 释放p**

/free(p);/

return 0;

}

6.6 调试下面的程序：

/callstk.c有3个函数调用深度的调用链/

yql@DESKTOP-TRMDDFQ:~$ vi callsk.c

i

#include <stdio.h>

#include <stdlib.h>

int make\_key(void);

int get\_key\_num(void);

int number(void);

int main(void)

{

int ret=make\_key();

printf(“make\_key returns %d\n”,ret);

exit(EXIT\_SUCCESS);

}

int make\_key (void)

{

int ret=get\_key\_num();

return ret;

}

int get\_key\_num(void)

{

int ret=number();

return ret;

}

int number(void)

{

return 10;

}

Enter esc

:wq

使用gdb调试程序callstk.c得到可运行的callstk文件。

yql@DESKTOP-TRMDDFQ:~$ gcc -g -o callstk.c

yql@DESKTOP-TRMDDFQ:~$ ls

a.out callstk.c ex3.sh callstk ex3\_2sh sort

yql@DESKTOP-TRMDDFQ:~$ gdb

GNU gdb (GDB) Fedora (7 .3.50.20191227-16. fc16)

Copyright (C) 2019 Free Software Foundation， Inc。

L .1cense GPLv3+: GNU GPL version 3 or Later <http:/ /gnu. org/11c enses/ gpl. html>

This is free software: you are free to change and redist ribute it.

There is NO WARRANTY， to the extent pe rmitted by 1 aw.

Type show copying"

and" show war ranty" for details.

This GDB was configured as " 1686- redhat- 1 inux- gnu" .

For bug. reporting inst ructions， please see:

<http: / 1 www. gnu. org/ sof twa re/ gdb/ bugs/> .

( gdb) file ./callstk

Reading symbols from / root/ex/callstk. . . done.

( gdb) l1st callstk.c:1,30

1 #include<stdio. h>

2 #include<stdlib. h>

3

4 int make\_ key( void) ;

5 int get\_ key\_ num( void) ;

6 int number( vo1d) :

7

8 int main( )

9 {

11 printf("make\_ key returns%d\n" , ret);

12 exit( EXIT\_ SUCCESS);

13 }

14

15 int make\_ key( vo1d)

16 {

17 int ret=get\_ key\_ num();

18 return ret;

19 }

20 int get\_ key\_ num(void)

21 {

22 int ret=number();

23 return ret;

---Type <return> to continue, or q <return> to quit---

24

25 int number( void)

26 {

27 return 10;

28 }

(gdb) break 10

Breakpoint 1 at 0x80483fd: file callstk.c, line 10.

(gdb) run

Starting program: / root/ex/callstk

Breakpoint 1, main () at callstk.c:10

10 int ret=make\_ key();

Missing separate debuginfos, use: debuginfo- install glibc-2.14.90-24. fc16.9. 1686

(gdb) step

make\_ key () at callstk.c:17

17 int ret=get\_ key\_ num();

(gdb) step

get\_ key\_ num () at callstk.c:22

22 int ret=number();

(gdb) step

number () at callstk.c:27

27 return 10;

(gdb) step

28 }

(gdb) kill

Kill the program being debugged? (y or n) y

(gdb) exit

Undefined command: "exit". Try "help".

(gdb) quit

6.7 GNU make的工作过程是怎样的？

答：

1. 依次读入各makefile文件；
2. 初始化文件中的变量；
3. 推导隐式规则，并分析所有规则；
4. 为所有的目标文件创建依赖关系链；
5. 根据依赖关系和时间数据，确定哪些目标文件要重新生成；
6. 执行相应的生成命令。

6.8 makefile的作用是什么？其书写规则是怎样的？

答：

作用：

定义了一系列规则，记录了文件之间的依赖关系及在此依赖关系基础上所应执行的命令序列，即定义了一系列规则来指定哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译等，此外，还可以有变量定义、注释等。

书写规则：

目标文件:[相依文件…]

<tab>命令1[#注释]

…

<tab>命令n[#注释]

在这种形式中，依赖行从一行的开头开始书写。如果依赖行中的目标文件或相依文件较多，在同一行写不下，此时可以用续行符“\”作为该行的结尾，而在下一行接着输入相应内容。各命令行单独占一行，每个命令行的第一个字符必须是制表符<tab>，而不能使用8个空格。#号后的内容为注释。

6.9 设某个正在开发的程序由以下内容组成：

① 4 个C语言源文件：a.c，b.c，c.c和d.c。设b.c和d.c都使用了defs.h中的声明。

② 汇编语言源文件assmb.s被某个C语言源文件调用。

③ 使用了在/home/user/lib/libm.so中的一组例程。

设最后生成的可执行文件名为prog。试编写相应的makefile文件。

答：利用make工具可以自动完成编译工作，这些工作包括：

如果修改了某几个源文件，则只重新编译这几个源文件；

如果某个头文件被修改了，则重新编译所有包含该头文件的源文件。

利用这种自动编译可以大大简化开发工作，避免不必要的重新编译。make工具通过一个称为Makefile的文件来完成并自动维护编译工作，Makefile文件描述了整个工程的编译、连接规则。

prog:a.o b.o c.o d.o assmb.o

gcc a.o b.o c.o d.o assmb.o -L /home/user/lib -lm -o prog

a.o:a.c

gcc -c a.c

b.o:b.c defs.h

gcc -c b.c

c.o:c.c

gcc -c c.c

d.o:d.c defs.h

gcc -c d.c

assmb.o:assmb.s

as -o assmb.o assmb.s

clean:

rm prog \*.o

**五、实验体会**

* 本次实验是在Linux环境下编辑c程序，通过函数的编写测试，我对gcc编译和gdb调试文件的功能有了进一步的了解。GDB（GNU Debugger）是GCC的调试工具，其功能非常强大：
* （1）启动程序后，可以按照自定义的要求随心所欲的运行程序。
* （2）可让被调试的程序在你所指定的调置的断点处停住。（断点可以是条件表达式）
* （3）当程序被停住时，可以检查此时程序中所发生的事。
* （4）可以动态的改变程序的执行环境。

我也对make工具有了自己的理解，make首先读取makefile文件,然后再激活编译器,汇编器,资源编译器和连接器以便产生最后的输出,最后输出并生成的通常是可执行文件。利用这种自动编译可以大大简化开发工作，避免不必要的重新编译。make工具通过一个称为Makefile的文件来完成并自动维护编译工作，Makefile文件描述了整个工程的编译、连接规则。