# 实验三 linux编程工具与环境

## 一、实验目的

1．掌握linux环境下C程序的编辑、编译、运行等操作；

2．掌握多文件系统的编译及连接库存的生成、应用；

3．初步掌握gdb调度方法；

4．初步掌握makefile工程文件的编写与使用；

5．掌握系统函数的应用。

## 二、实验目标与要求

1.函数库的创建。

2.makefile工程文件的编写。

3.应用gdb调试程序。

4.使用GCC编译时连接库的使用。

## 三、实验工具与准备

计算机PC机，Linux Redhat 9.0或者其他的linux操作系统

## 四、实验任务与操作指导

**任务一：**

调试下列程序。程序通过创建一个小型函数库，它包含两个函数，然后在一个示例程序中调用其中一个函数。这两个函数分别是pro1和pro2。按下面步骤生成函数库及测试函数库。

步骤1 为两个函数分别创建各自的源文件（将它们分别命名为pro1.c和pro2.c）。

[root@localhost root]# cat pro1.c

#include <sdtio.h>

void pro1(int arg)

{

printf(“hello：%d\n”,arg) ;

}

[root@localhost root]# cat pro2.c

#include <sdtio.h>

void pro2(char \*arg)

{

printf(“您好：%s\n”,arg) ;

}

步骤2 分别编译这两个函数，产生要包含在库文件中的目标文件。这通过调用带有-c选项的gcc编译器来实现，-c选项的作用是阻止编译器创建一个完整的程序,gcc将把源程序编译成目标程序，文件名为以.o结尾。如果此时试图创建一个完整的程序将不会成功，因为还未定义main函数。

[root@localhost root]# gcc -c pro1.c pro2.c

[root@localhost root]# ls \*.o

pro1.o pro2.o

步骤3 现在编写一个调用pro2函数的程序。首先，为库文件创建一个头文件lib.h。这个头文件将声明库文件中的函数，它应该被所有希望使用库文件的应用程序所包含。

[root@localhost root]# cat lib.h

/\*lib.h：pro1.c，pro2.c\*/

void pro1(int);

void pro2(char \*);

步骤4 主程序（program.c）非常简单。它包含库的头文件并且调用库中的一个函数。

[root@localhost root]# cat program.c

#include “lib.h”

int main()

{

pro2(“Linux world”);

exit(0);

}

步骤5 现在，来编译并测试这个程序。暂时为编译器显式指定目标文件，然后要求编译器编译的文件并将其与预先编译好的目标模块pro2.o链接。

[root@localhost root]# gcc -c program.c

[root@localhost root]# gcc -o program program.o pro2.o

[root@localhost root]# ./program

您好：Linux world

步骤6 现在，创建并使用一个库文件。用ar程序创建一个归档文件并将目标文件添加进去。这个程序之所以称为ar，是因为它将若干单独的文件归并到一个大的文件中以创建归档文件。注意，也可以用ar程序来创建任何类型文件的归档文件。

[root@localhost root]# ar crv libfoo.a pro1.o pro2.o

函数库现在即可使用了。可以在编译器命令行的文件列表中添加该库文件以创建程序，如下所示：

[root@localhost root]# gcc -o program program.o libfoo.a

[root@localhost root]# ./program

您好：Linux world

也可以用-l选项来访问函数库，但是因为其未保存在标准位置，所以必须用-L选项来指示gcc在何处可以找到它，如下所示：

[root@localhost root]#gcc -o program program.o -L. -lfoo

-L.选项指示编译器在当前目录”.”中查找函数库。-lfoo选项指示编译器使用名为libfoo.a的函数库（或者名为libfoo.so的共享库，如果它存在的话）。

要查看目标文件、函数库或可执行文件里包含的函数， 可使用nm命令。如果查看program和libfoo.a，就会看到函数库libfoo.a中包含pro1和pro2两个函数，而 program里只包含函数pro2。创建程序时，它只包含函数库中它实际需要的函数。虽然程序中的头文件包含函数库中所有函数的声明，但这并不将整个函数库包含在最终的程序中。

（1）按照给出的步骤1-6调试程序；

（2）编写两个函数，其中一个求数组中的最大值与最小值，另一函数求数组中某一数的个数，建立一个库，编写main函数实现对数组中某一数个数的统计，并输出数组的最大值，最小值。对程序进行调试；

（3）试写出问题（2）中的makefile工程文件。

**任务二：**

调试下列程序。

#include<unistd.h>

int main()

{

char passwd[13];

char \*key;

char slat[2];

key= getpass(“Input First Password:”);

slat[0]=key[0];

slat[1]=key[1];

strcpy(passwd,crypt(key slat));

key=getpass(“Input Second Password:”);

slat[0]=passwd[0];

slat[1]=passwd[1];

printf(“After crypt(),1st passwd :%s\n”,passwd);

printf(“After crypt(),2nd passwd:%s \n”,crypt(key slat));

}

请写出调试命令。

**任务三：**

调试下列程序。调试目的是实现多文件编译。文件结构是：

（1）bubble.h 存放函数bubble的声明。

（2）bubble.c 存放函数bubble的实现。

（3）main.c 存放main函数实现，其中调用了bubble函数。

（4）补充完整makefile文件

其中各文件中的代码如下：

bubble.h

#ifndef \_\_BUBBLE\_H\_\_

#define \_\_BUBBLE\_H\_\_

void bubble(int\* a, int n);

#endif // \_\_BUBBLE\_H\_\_

bubble.c

#include "bubble.h"

void bubble(int\* a, int n)

{

int i, j, t;

for(i = 1; i < n; i++)

{

for(j = 0; j < n - i; j++)

{

if(a[j] > a[j + 1])

{

t = a[j];

a[j] = a[j + 1];

a[j + 1] = t;

}

}

}

}

**main.c**

#include <stdio.h>

#include "bubble.h"

#define COUNT 10

int main(void)

{

int i;

int a[COUNT] = {3, 5, 4, 8, 9, 6, 2, 1, 7, 0};

bubble(a, COUNT);

for(i = 0; i < COUNT; i++)

{

printf("%d ", a[i]);

}

printf("\n");

return 0;

}

**Makefile**

CC = gcc

FLAGS = -Wall -O2

main: main.o bubble.o

（请补充完整makefile文件）

**任务五：**

程序调试。程序的功能是输入两个整数，输出两个整数间的所有自然数。

步骤1 设计编辑源程序代码。

[root@localhost root]#**vim 4-3.c**

程序代码如下：

#include <stdio.h>

int num(int x, int y);

int main()

{

int a1,a2,min\_int;

printf("请输入第一个整数：");

scanf("%d",&a1);

printf("请输入第二个整数：");

scanf("%d",&a2);

num(a1,a2);

}

int num(int x, int y)

{

int temp,i;

if (x>y)

{temp=x;x=y;y=temp;}

for(i=x;i<=y;i++)

printf(“%5d”,i);

printf(“\n”);

return 0;

}

步骤2 用gcc编译程序。

在编译的时候要加上选项“-g”。这样编译出的可执行代码中才包含调试信息，否则之后gdb无法载入该可执行文件，请写出编译命令。

[root@localhost root]#

步骤3 进入gdb调试环境。

gdb进行调试的是可执行文件，因此要调试的是**4-3**而不是**4-3**.c，输入如下，请输入gdb的调试命令：

[root@localhost root]#

回车后就进入了gdb调试模式。

GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)

Copyright 2003 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are

welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "i386-redhat-linux-gnu"...

(gdb)

图4.1 进入gdb调试环境

可以看到，在gdb的启动画面中有gdb的版本号、使用的库文件等信息，在gdb的调试环境中，提示符是“(gdb)”。

步骤4 用gdb调试程序。

（1）查看源文件

在gdb中输入“l”(list)就可以查看程序源代码，一次显示10行。

可以看出，gdb列出的源代码中明确地给出了对应的行号，这样可以大大方便代码的定位。

（2）设置断点

设置断点在调试程序时是一个非常重要的手段，它可以使程序到一定位置暂停运行。软件工程师可以在断点处查看变量的值、堆栈情况等，从而找出代码的问题所在。

在gdb中设置断点命令是“b”(break)，后面跟行号或者函数名。

不指定具体行号的断点设置在“b”(break)后面跟函数名。本例可以输入“break num”。

（3）查看断点信息

设置完断点后，可以用命令“info b”(info break)查看断点信息。

（4）运行程序

接下来可以运行程序，可以输入“r”(run)开始运行程序。

（5）查看与设置变量值

调试程序重要手段就是查看断点处的变量值。程序运行到断点处会自动暂停，此时输入“p变量名”可查看指定变量的值。

调试程序时，若需要修改变量值，可程序运行到断点处时，输入“set 变量=设定值”，例如，给变量“a2” 赋值100，输入“set a2=100”。

gdb在显示变量值时都会在对应值前加“$n”标记，它是当前变量值的引用标记，以后想再引用此变量，可以直接使用“$n”，提高了调试效率。

（6）单步运行

很多情况下，调试的时候要单步运行程序。在断点处输入 “n”(next)或者“s”(step)可单步运行。它们之间的区别在于：若有函数，调用时，“s”会进入该函数，而“n”不会进入该函数。

（7）继续运行程序

在查看完变量或堆栈情况后可以输入“c”(continue)命令恢复程序的正常运行，把剩余的程序执行完，并显示执行结果。

（8）退出gdb环境

退出gdb环境只要输入“q”(quit)命令，回车后退出gdb环境。