

《高级操作系统Linux》实验报告



**实验【4】： 编译和调试工具的使用**

**学 生：**  **白乐荣**

**学 号： 201606060101**

**学 院： 电子信息与人工智能学院**

**系 别： 计算机系**

**专 业： 计算机科学与技术**

**《高级操作系统Linux》**实验预习报告

学号 201606060101 姓名 白乐荣 班级 计本163

实验日期： 2019 年 5 月 12 日 成绩：

**一、实验目的：**

1. 练习并掌握Linux提供的vi编辑器来编译C程序
2. 学会利用gcc、gdb编译、调试C程序
3. 学会使用make工具

**二、实验内容**

1. 编写C语言程序，用gcc编译并观察编译后的结果，运行生成的可执行文件。
2. 利用gdb调试程序。
3. 学习编写makefile，并进行编译。

**三、实验操作**

1、GNU C编译器

1) 使用gcc

通常后跟一些选项和文件名来使用gcc编译器。gcc命令的基本用法如下：

gcc [options] [filenames]

命令行选项指定的编译过程中的具体操作

2) gcc常用选项

当不用任何选项编译一个程序时，gcc将建立（假定编译成功）一个名为a.out的可执行文件。

选项含义：

-o FILE 指定输出文件名，在编译为目标代码时，这一选项不是必须的。如果FILE 没

有指定，默认文件名是a.out．

例如，

# gcc test.c

编译成功后，当前目录下就产生了一个a.out文件。

也可用-o选项来为即将产生的可执行文件指定一个文件名来代替a.out。

例如：

#gcc –o count count.c

此时得到的可执行文件就不再是a.out，而是count。

-c GCC 仅把源代码编译为目标代码。默认时GCC 建立的目标代码文件有一个.o 的

扩展名。

-E 对文件进行预处理

-S 对文件进行编译，生成汇编代码。

-O 对源代码进行基本优化。这些优化在大多数情况下都会使程序执行得更快。

-g 在可执行程序中包含标准调试信息。

-Wall 允许发出GCC 能提供的所有有用的警告，也可以用-W(warning)来标识指定的

警告。

-l *name* 链接静态库

-L *dir* 库文件的搜索路径

3) 执行文件

格式： ./可执行文件名

例：

# ./a.out

# ./count

4) 编译程序

写出编译过程，并给出运行结果。

mypow.c：定义mypow()函数

unsigned long long mypow(unsigned int x, unsigned int y)

{

unsigned long long res=1;

if (y==0)

res = 1;

else if (y==1)

res = x;

else

res = x \* mypow(x, y-1);

return res;

}

powtest.c：调用mypow()函数

#include <stdio.h>

#include <stdlib.h>

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

{

unsigned int x, y;

unsigned long long res;

if ((argc<3) || (sscanf(argv[1], "%u", &x)) != 1 || (sscanf(argv[2], "%u", &y)) != 1 )

{

printf("Usage:pow base exponent\n");

exit(1);

}

res = mypow(x, y);

printf("%u ^ %u = %u\n", x, y, res);

return 0;

}

2、gdb调试工具

1) 调试编译代码

为了使gdb正常工作，必须使你的程序在编译时包含调试信息。调试信息里包含你程序里的每个变量的类型和在可执行文件里的地址映射以及源代码的行号。gdb利用这些信息使源代码和机器码相关联。

在编译时用 –g 选项打开调试选项。

2) gdb基本命令

|  |  |
| --- | --- |
| 命 令 | 描 述 |
| file | 装入欲调试的可执行文件 |
| kill | 终止正在调试的程序 |
| list | 列出产生执行文件的源代码部分 |
| next | 执行一行源代码但不进入函数内部 |
| step | 执行一行源代码并进入函数内部 |
| run | 执行当前被调试的程序 |
| quit | 终止gdb |
| watch | 监视一个变量的值而不管它何时被改变 |
| break | 在代码里设置断点，使程序执行到这里时被挂起 |
| make | 不退出gdb就可以重新产生可执行文件 |
| shell | 不离开gdb就执行UNIX shell 命令 |

3) 调试程序

(1) 源程序gdbtest1.c，分析程序的功能，如果出错，要求用gdb进行调试并给出修改方案。

基本步骤：

* 编译：gcc -g -o gdbtest1 gdbtest1.c
* 运行：./gdbtest1
* 开始调试：gdb gdbtest1

#include <stdio.h>

int add\_range(int low, int high)

{

static int i,sum;

for(i=low; i<=high; i++)

sum = sum +i;

return sum;

}

int main(void)

{

int result[100];

result[0] = add\_range(1,10);

result[1] = add\_range(1,100);

printf("result[0]=%d\nresult[1]=%d\n",result[0],result[1]);

return 0;

}

(2) 源程序greet.c，功能：按照正序和逆序输出给定的字符串。要求用gdb进行调试，，分析出错的原因并给出修改方案。

#include <stdio.h>

#include <string.h>

#include <malloc.h>

void my\_print (char \*string);

void my\_print2 (char \*string);

int main()

{

char my\_string[] = "hello there";

my\_print (my\_string);

my\_print2 (my\_string);

return 0;

}

void my\_print (char \*string)

{

printf ("The string is %s\n", string);

}

void my\_print2 (char \*string)

{

char \*string2;

int size, i;

size = strlen (string);

string2 = (char \*) malloc (size + 1);

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

string2[size - i] = string[i];

string2[size + 2] = '\0';

printf ("The string printed backward is %s\n", string2);

}

3、make的使用

（1）用vi编辑以下程序，程序清单：

main.c

function1.h

function1.c

function2.h

function2.c

//main.c

#include "function1.h"

#include "function2.h"

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

{

function1\_print("hello");

function2\_print("world");

return 0;

}

//function1.h

void function1\_print(char \*str);

//function1.c

#include "function1.h"

void function1\_print(char \*str)

{

printf("This is function1 print %s\n", str);

}

//function2.h

void function2\_print(char \*str);

//function2.c

#include "function2.h"

void function2\_print(char \*str)

{

printf("This is function2 print %s\n", str);

}

实验要求：

（1）画出各个源程序、目标文件以及最终的目标文件之间的依赖关系图。

（2）编辑makefile文件

（3）利用make命令进行上述程序的编译，生成可执行代码并运行。

（4）修改其中一个源文件，重新make，察看编译过程。

（5）通过使用makefile变量和隐含规则，对makefile文件进行简化

四、**预习心得**

通过本次预习，我主要学习了如何利用gcc和gdb对代码进行调试，并且学会了编写简单的makefile对自己的简单代码进行分析，找出错误的地点，进行修改。

**《高级操作系统Linux》**实验报告

学号 201606060101 姓名 白乐荣 班级 计本163

实验日期： 2019 年 5 月 12 日 成绩：

**一、实验目的：**

1. 练习并掌握Linux提供的vi编辑器来编译C程序
2. 学会利用gcc、gdb编译、调试C程序
3. 学会使用make工具

**二、实验内容**

1. 编写C语言程序，用gcc编译并观察编译后的结果，运行生成的可执行文件。
2. 利用gdb调试程序。
3. 学习编写makefile，并进行编译。

**三、实验操作**

1、GNU C编译器

1) 使用gcc

通常后跟一些选项和文件名来使用gcc编译器。gcc命令的基本用法如下：

gcc [options] [filenames]

命令行选项指定的编译过程中的具体操作

2) gcc常用选项

当不用任何选项编译一个程序时，gcc将建立（假定编译成功）一个名为a.out的可执行文件。

选项含义：

-o FILE 指定输出文件名，在编译为目标代码时，这一选项不是必须的。如果FILE 没

有指定，默认文件名是a.out．

例如，

# gcc test.c

编译成功后，当前目录下就产生了一个a.out文件。

也可用-o选项来为即将产生的可执行文件指定一个文件名来代替a.out。

例如：

#gcc –o count count.c

此时得到的可执行文件就不再是a.out，而是count。

-c GCC 仅把源代码编译为目标代码。默认时GCC 建立的目标代码文件有一个.o 的

扩展名。

-E 对文件进行预处理

-S 对文件进行编译，生成汇编代码。

-O 对源代码进行基本优化。这些优化在大多数情况下都会使程序执行得更快。

-g 在可执行程序中包含标准调试信息。

-Wall 允许发出GCC 能提供的所有有用的警告，也可以用-W(warning)来标识指定的

警告。

-l *name* 链接静态库

-L *dir* 库文件的搜索路径

3) 执行文件

格式： ./可执行文件名

例：

# ./a.out

# ./count

4) 编译程序

写出编译过程，并给出运行结果。

mypow.c：定义mypow()函数

unsigned long long mypow(unsigned int x, unsigned int y)

{

unsigned long long res=1;

if (y==0)

res = 1;

else if (y==1)

res = x;

else

res = x \* mypow(x, y-1);

return res;

}

powtest.c：调用mypow()函数

#include <stdio.h>

#include <stdlib.h>

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

{

unsigned int x, y;

unsigned long long res;

if ((argc<3) || (sscanf(argv[1], "%u", &x)) != 1 || (sscanf(argv[2], "%u", &y)) != 1 )

{

printf("Usage:pow base exponent\n");

exit(1);

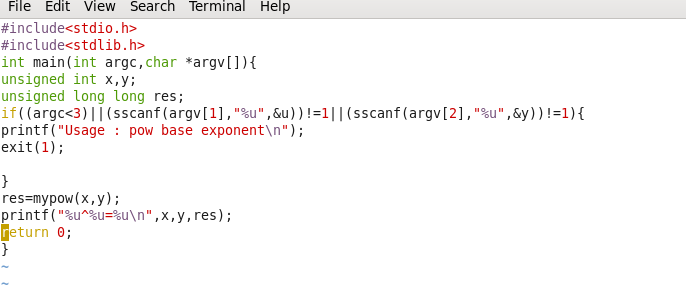
}

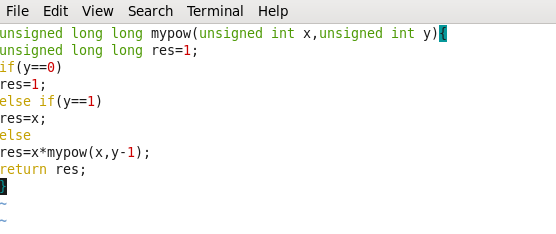
res = mypow(x, y);

printf("%u ^ %u = %u\n", x, y, res);

return 0;

}



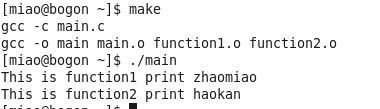


2、gdb调试工具

1) 调试编译代码

为了使gdb正常工作，必须使你的程序在编译时包含调试信息。调试信息里包含你程序里的每个变量的类型和在可执行文件里的地址映射以及源代码的行号。gdb利用这些信息使源代码和机器码相关联。

在编译时用 –g 选项打开调试选项。

2) gdb基本命令

|  |  |
| --- | --- |
| 命 令 | 描 述 |
| file | 装入欲调试的可执行文件 |
| kill | 终止正在调试的程序 |
| list | 列出产生执行文件的源代码部分 |
| next | 执行一行源代码但不进入函数内部 |
| step | 执行一行源代码并进入函数内部 |
| run | 执行当前被调试的程序 |
| quit | 终止gdb |
| watch | 监视一个变量的值而不管它何时被改变 |
| break | 在代码里设置断点，使程序执行到这里时被挂起 |
| make | 不退出gdb就可以重新产生可执行文件 |
| shell | 不离开gdb就执行UNIX shell 命令 |

3) 调试程序

(1) 源程序gdbtest1.c，分析程序的功能，如果出错，要求用gdb进行调试并给出修改方案。

基本步骤：

* 编译：gcc -g -o gdbtest1 gdbtest1.c
* 运行：./gdbtest1
* 开始调试：gdb gdbtest1

#include <stdio.h>

int add\_range(int low, int high)

{

static int i,sum;

for(i=low; i<=high; i++)

sum = sum +i;

return sum;

}

int main(void)

{

int result[100];

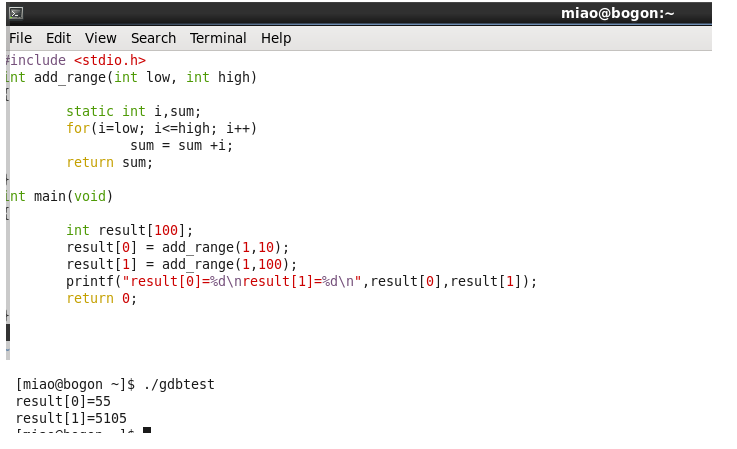
result[0] = add\_range(1,10);

result[1] = add\_range(1,100);

printf("result[0]=%d\nresult[1]=%d\n",result[0],result[1]);

return 0;

}



(2) 源程序greet.c，功能：按照正序和逆序输出给定的字符串。要求用gdb进行调试，，分析出错的原因并给出修改方案。

#include <stdio.h>

#include <string.h>

#include <malloc.h>

void my\_print (char \*string);

void my\_print2 (char \*string);

int main()

{

char my\_string[] = "hello there";

my\_print (my\_string);

my\_print2 (my\_string);

return 0;

}

void my\_print (char \*string)

{

printf ("The string is %s\n", string);

}

void my\_print2 (char \*string)

{

char \*string2;

int size, i;

size = strlen (string);

string2 = (char \*) malloc (size + 1);

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

string2[size - i] = string[i];

string2[size + 2] = '\0';

printf ("The string printed backward is %s\n", string2);

}



错误原因：将string[0]赋给string2[11]时出现错误，原因是给string2的第一位赋的是“\0”。

调试过程：在函数的第10行设置断点，并使用display实时查看string2、string、i的值，运行到第一次循环结束后发现string[0]赋给string[11]出现了错误

修改措施：使string最后一位的”\0“不赋给string2的第一位。

3、make的使用

（1）用vi编辑以下程序，程序清单：

main.c

function1.h

function1.c

function2.h

function2.c

//main.c

#include "function1.h"

#include "function2.h"

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

{

function1\_print("hello");

function2\_print("world");

return 0;

}

//function1.h

void function1\_print(char \*str);

//function1.c

#include "function1.h"

void function1\_print(char \*str)

{

printf("This is function1 print %s\n", str);

}

//function2.h

void function2\_print(char \*str);

//function2.c

#include "function2.h"

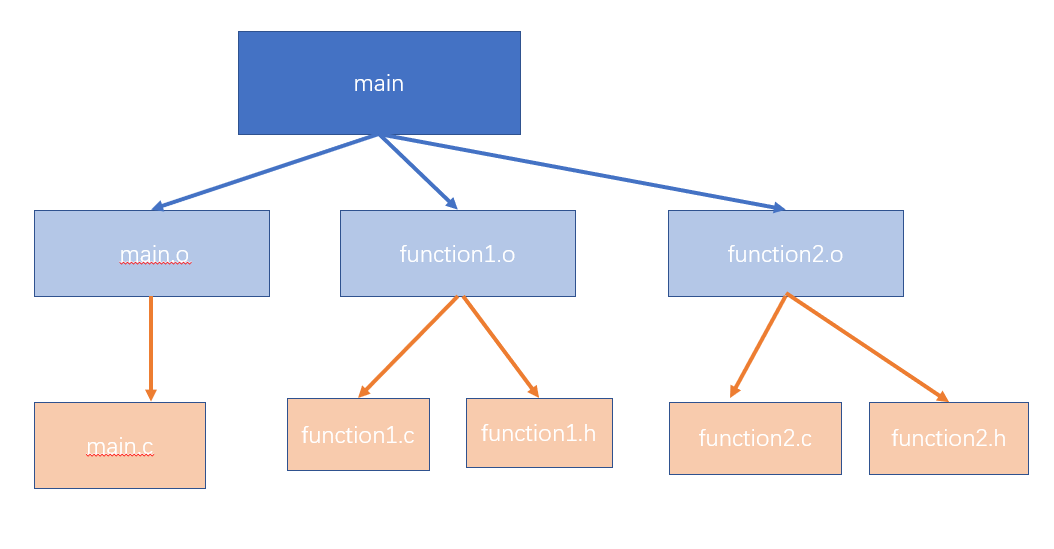
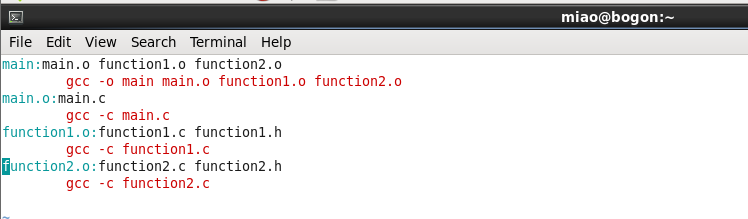
void function2\_print(char \*str)

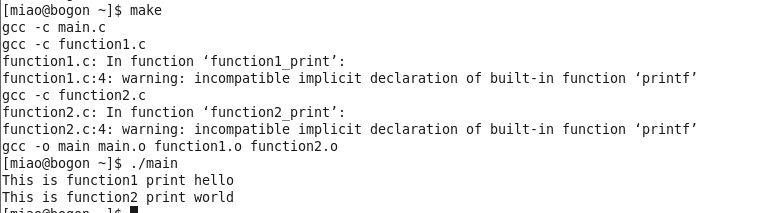
{

printf("This is function2 print %s\n", str);

}

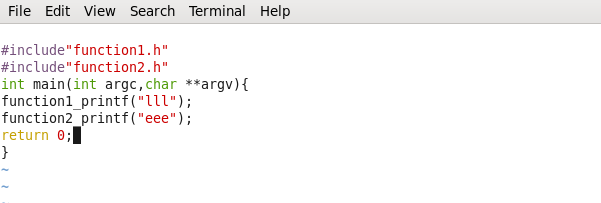
实验要求：

1. 画出各个源程序、目标文件以及最终的目标文件之间的依赖关系图。
2. 编辑makefile文件
3. 利用make命令进行上述程序的编译，生成可执行代码并运行。

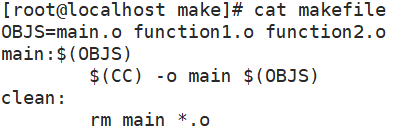


1. 修改其中一个源文件，重新make，察看编译过程。

修改main.c



1. 通过使用makefile变量和隐含规则，对makefile文件进行简化



**四、实验心得**

通过本次实验我学到了如何使用gdb调试代码，找出代码中的错误，学会了简单的makefile处理以及一些编程操作。