

**软件学院**

**《Unix/Linux程序设计》**

**实验报告**

**学生姓名：** 刘铭源

**学 号：** 2018214937

**专业班级：** 软件工程18-4班

**2020 年 5 月 14 日**

**实验一vi 编辑器以及 X Window 桌面的使用**

**一、实验目的**

练习vi编辑器以及X Window桌面的使用。

需要的知识：第3章vi编辑器的使用和第4章建立快速链接的内容。

**二、实验内容**

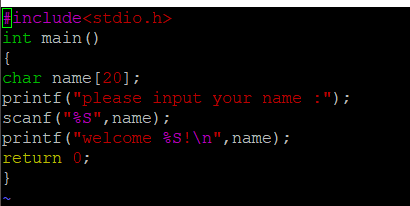
首先使用vi编辑器编辑一个c语言源程序hello.c，程序的功能为：接受用户输入的姓名，然后向屏幕输出一条语句“Welcome 姓名”。使用命令“gcc –o hello hello.c”编译源程序，如果发现程序错误则需要反复使用vi修改源文件并且在修改后重新编译。最后在X Window的桌面上放置源程序hello.c的链接。

**三、实验设备及工具（包括软件调试工具）**

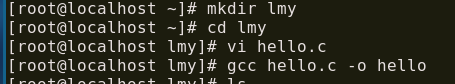
**VMVARE Fedora11系统**

1. **实验步骤**

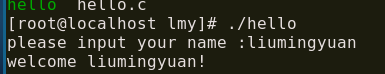
1.使用vi编辑器编辑hello.c



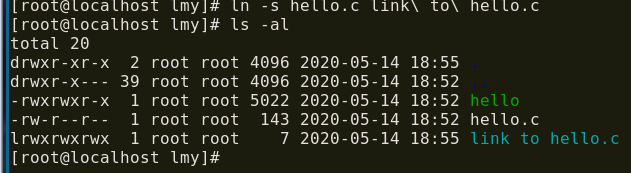
1. 编译文件gcc –o hello hello.c



3.运行可执行文件



1. link文件实验



1. **实验结论与分析**

在Linux系统中有种文件是链接文件，可以为解决文件的共享使用。链接的方式可以分为两种，一种是硬链接（Hard Link），另一种是软链接或者也称为符号链接（Symbolic Link）。

软连接和硬链接的特点：

软链接：

1.软链接是存放另一个文件的路径的形式存在。

2.软链接可以 跨文件系统 ，硬链接不可以。

3.软链接可以对一个不存在的文件名进行链接，硬链接必须要有源文件。

4.软链接可以对目录进行链接。

硬链接：

1. 硬链接，以文件副本的形式存在。但不占用实际空间。

2. 不允许给目录创建硬链接。

3. 硬链接只有在同一个文件系统中才能创建。

4. 删除其中一个硬链接文件并不影响其他有相同 inode 号的文件。

不论是硬链接或软链接都不会将原本的档案复制一份，只会占用非常少量的磁碟空间。

**实验 2 存储设备挂载及管理任务设置**

**一、实验目的**

练习存储设备的挂载以及编写shell程序将日常管理任务简化。

需要的知识：第6章存储设备的挂载和第11章shell编程的内容

**二、实验内容**

假设有一台Windows和Linux双启动的主机，Window有C、D两个分区，编写一个负责挂载的脚本程序mmnt，参数为cdrom时挂载光驱、参数为floppy时挂载软驱、参数为usb时挂载U盘、参数为C（大小写都可以）时挂载Windows的C盘（有中文字符），参数为D（大小写都可以）时挂载Windows的D盘（有中文字符），参数为其它时显示帮助信息：Usage: mmnt cdrom|floppy|usb|C|c|D|d

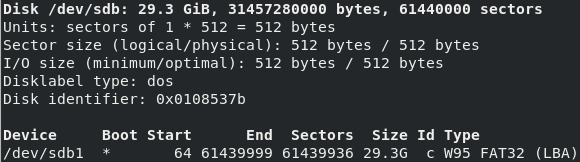
编写一个负责卸载的脚本程序umnt，完成相应的存储设备的卸载，参数的含义同上。

**三、实验设备及工具（包括软件调试工具）**

**VMVARE Fedora11系统**

**四、实验步骤**

步骤1：以root身份登录Linux系统，使用fdisk –l命令查看系统连接的存储设备，主要查看windows的两个分区的名称，比如hda3, hda5等。



步骤2：建立挂载点：一般地，光驱和软驱的挂载点/mnt/cdrom、/mnt/floppy由系统缺省建立，查看一下这两个目录是否存在，如果没有，手工建立；此外使用mkdir命令建立U盘和windows的C盘和D盘的挂载点：

mkdir /mnt/usb

mkdir /mnt/C

mkdir /mnt/D

Screen Shot 2019-06-16 at 8.40.33 AM

步骤3：编写/etc/fstab文件，编辑或者添加如下5行：

（操作时只需要添加usb行，其他四行暂未用到，不影响。）

/dev/hda3 /mnt/C vfat noauto,iocharset=cp936 0 0

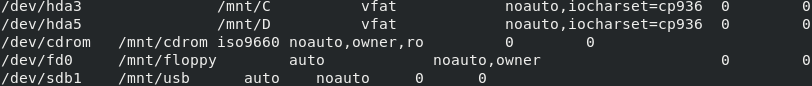
/dev/hda5 /mnt/D vfat noauto,iocharset=cp936 0 0

/dev/cdrom /mnt/cdrom iso9660 noauto,owner,ro 0 0

/dev/fd0 /mnt/floppy auto noauto,owner 0 0

/dev/sdb1 /mnt/usb auto noauto 0 0





步骤4：编写shell脚本mmnt （vi mmnt）

#!/bin/sh

case $1 in

cdrom) mount /mnt/cdrom ;;

floppy) mount /mnt/floppy ;;

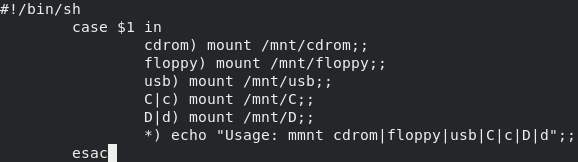
usb) mount /mnt/usb;;

C|c) mount /mnt/C ;;

D|d) mount /mnt/D ;;

\*) echo “Usage: mmnt cdrom|floppy|usb|C|c|D|d”;;

Esac

****

步骤5：编写shell脚本umnt （vi umnt）

#!/bin/sh

case $1 in

cdrom) umount /mnt/cdrom ;;

floppy) umount /mnt/floppy ;;

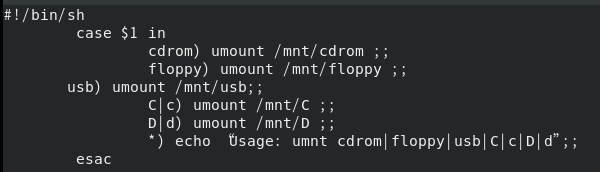
usb) umount /mnt/usb;;

C|c) umount /mnt/C ;;

D|d) umount /mnt/D ;;

\*) echo “Usage: umnt cdrom|floppy|usb|C|c|D|d”;;

Esac



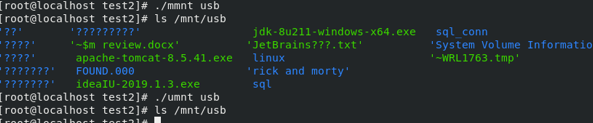
步骤6：使脚本mmnt和umnt可执行

chmod +x mmnt

chmod +x umnt

提示：此实验可以分两步，学完第6章后完成步骤1到步骤3，使用系统命令mount和umount测试自己的设置，学完第11章shell编程之后完成最后几步。



**五、实验结果与分析**

在linux操作系统中，mount命令是用于加载文件系统到指定的加载点。此命令最常用于挂载cdrom，使我们可以访问cdrom中的数据，当你将光盘插入cdrom中时，Linux并不会自动挂载，必须使用Linux mount命令来手动完成挂载。在这里需要注意的是，挂载点必须是一个已经存在的目录，这个目录可以不为空，但挂载后这个目录下以前的内容将不可用，只有在umount卸载以后才会恢复正常，而且只有目录才能被挂载，文件不可以被挂载，而如果挂载在非空目录下可能会导致系统异常，所以建议挂载在空目录下。对于经常使用的设备可写入文件/etc/fastab,以使系统在每次开机时自动加载。mount加载设备的信息记录再/etc/mtab文件中，可使用umount命令卸载清除记录。

**实验 3 网络服务的配置与管理**

**一、实验目的**

练习网络服务的配置与管理。

需要的知识：第 7 章网络配置、第 8 章网络服务器的配置和使用。

**二、实验内容**

某局域网(yys.com)具有 3 台安装了 Linux 系统的计算机，分别用于提供

WWW、FTP 和 DNS 服务，要求：设计分配 IP 地址和主机名，配置三种服务器，

使得可以通过域名访问网络服务。

**三、实验设备及工具（包括软件调试工具）**

**VMVARE Fedora11系统**

**四、实验步骤**

步骤 1：设计分配各主机的 IP 地址和域名，假设三台主机分别为 A、B、C，可

以如下分配 IP 地址和域名：

主机

IP

主机名

域名

A.

192.168.1.10

www

www.yys.com

B

192.168.1.11

ftp

ftp.yys.com

C

192.168.1.12

dns

dns.yys.com

步骤 2：构建 WWW 服务器

1. 在 A 主机中安装最新版的 Apache 服务器软件

2. 配置 Apache

3. 在其主文档目录中建立 web 主页

4. 启动 Apache 服务器

5. 在本机中测试服务器与网页是否正常运行

步骤 3：构建 FTP 服务器

1. 在 B 主机中安装最新版的 vsFtpd 服务器软件

2. 配置 vsFtpd，按需求建立不同形式的登录方式

3. 在对应的文件目录中复制有关文件资料

4. 启动 vsFtpd 服务器

5. 在本机中测试服务器是否正常运行

步骤 4：构建 DNS 服务器

1. 在 C 主机中安装最新版的 bind 服务器软件

2. 配置系统，建立主机与 IP 地址的映射。

3. 启动 bind 服务器

4. 在本机中测试服务器是否正常运行

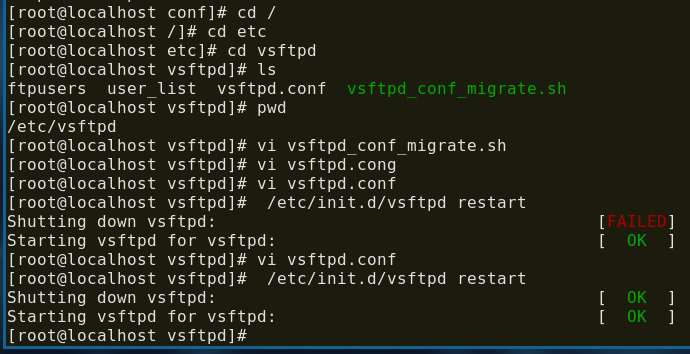
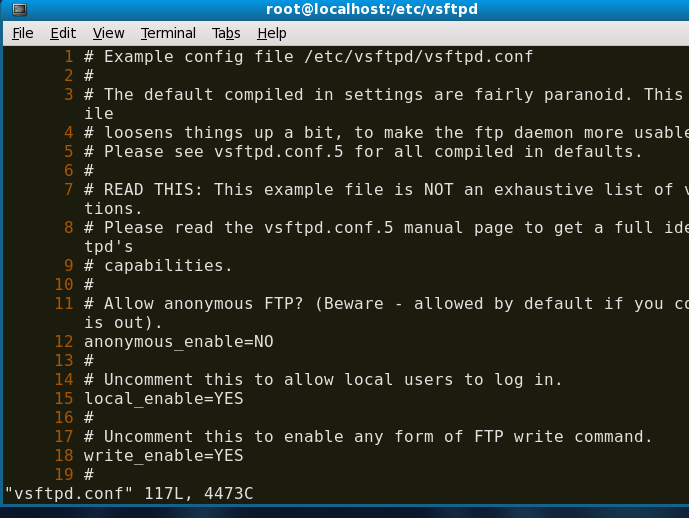
步骤 5：设置客户机

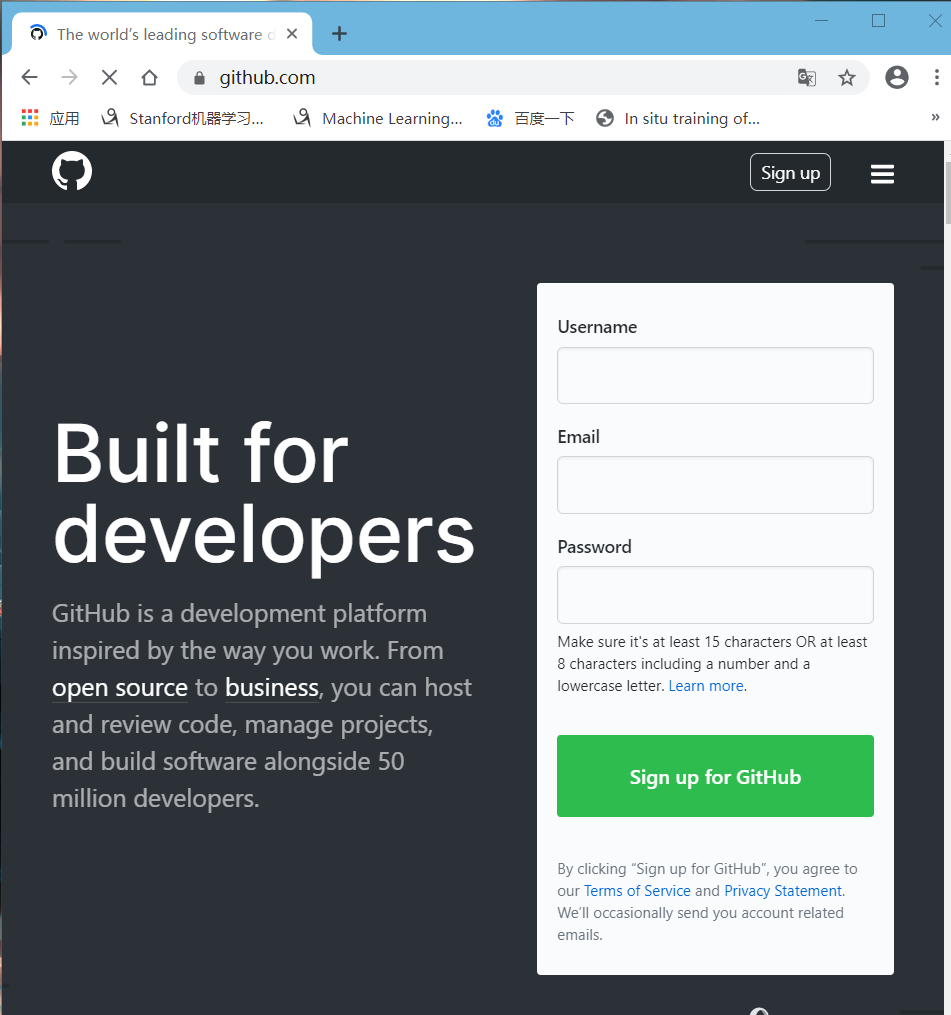
1. 在客户机中修改本地 DNS 服务器的设置。

2. 打开网页浏览器，在地址栏中输入：http://www.yys.com ,检验是否连接 WWW服务器

3. 打开网页浏览器，在地址栏中输入：ftp://ftp.yys.com ,检验是否连接 FTP 服务器

**五、实验结果与分析**

****

****

1. **分析**

这个实验需要仔细的查看项目指导书，完成项目，需要对ftp启动和配置十分了解，完成对ftp的配置，重启ftp

**实验五Linux 环境下 gcc 的使用**

**一、实验目的**

了解 Linux 下的用户管理机制，熟练掌握 Linux 下的 C 语言编程以及 gcc 工

具的使用。

需要的知识：C/C++语言的基本技能、第 9 章用户管理知识、第 12 章 gcc 的使

用。

**二、实验内容**

Linux 系统管理的一项重要工作就是用户管理。用户的口令以加密的形式存

储在口令文件/etc/shadow 中。弱口令就是很容易被猜出来的口令，比如与用户名

相同的口令、像“123456、admin、computer”这类常用的口令等，口令字典是

指将一些常用弱口令写在一个文本文件。管理员应该定期检测系统中是否存在弱

口令。试编写一个 c 语言程序，主动检查自己的 Linux 系统中是否存在弱口令。

1. **实验环境**

**VMWARE** **Fedora11系统**

**四、实验步骤**

步骤 1：分析/etc/shadow 文件的格式。

可以查看自己的系统中该文件的格式，参照 9.1 节的介绍，该文件的每一行

对应一个用户，下面是一个系统中/etc/shadow 文件的实例：

user1:$6$DVLiBPZG$IrR9o0KpjtGQOj7I5WvgxQ.jMQ/Qzl3cJp4w0loUMZs4xQSQ8wk

dIK7Sdkdk2pMXeAfOYq9O07r/QuDdJ8f3c0:14748:0:99999:7:::

user2:$6$RR8pmW2aerqIkySA$PcMV7/Z37QFCe9hJrf1rlUjPTAOdmKsW/mfG40V343k

xG1QNsWWI7mdzl.50SCJpI4TQ/x4z4zsCoiA48EjAn1:14748:0:99999:7:::

user3:$6$oGG.739y$9ysybZ.VaTQ7dmR1zyz1vR8OiCUSCnzqnFd1PUYvedJMt.t6ElI

SwUohtOAlqAuT7.sPDjfy.bKCKar82mSp2.:14748:0:99999:7:::

user4:$6$AJerec5o$bwgJnQ0mTPzZMRCZYuivQVsWtD9mlh3.pWK2tR2pZPr4NzSlqk6

hhFq3/zfWJXQCNmXJTlZhubwwW9x6a8mtM0:14748:0:99999:7:::

user5:$6$o1DA6WOiXtme7Zsw$Lj6bXgI5c5Kg/GewGWYv.4pQ0fD/AnSYEjMvmXxuvzE

K3IYhNYjnTIEnTPQr9pZIMzatBOyrC4FgBjYtR5R.n1:14749:0:99999:7:::

user6:$6$UUXC4WKX$mKyl/32n7xrU1ChPIJYs2gwYuEoObdkNsPcVhxR22xFBIspNjVI

fRv4FgoDWsJIxy4TZ.ci70MeDvDMEeExXL0:14749:0:99999:7:::

步骤 2：了解 Linux 下口令加密的原理

口令的加密是使用 Linux 系统的 crypt 函数。使用 info 或者 man 可以了解

该命令的工作原理：Red Hat Linux9 中使用的 MD5 算法，在 Fedora 10 中默认

使用了 SHA-512 算法。它使用一个字符串作为 salt（翻译成盐），长度是 8 到 16

个字节，将用户输入的口令 key 进行加密。函数 crypt(key,salt)的结果存入

/etc/shadow 文件的第 2 个域。这个域前面$6$salt$就是 crypt 函数的 salt 参

数。当给用户设置口令时，salt 是由系统随机选取的，因此即使是相同的 key，

因 salt 不同而第二个域的值也不同。

当用户在登录中输入自己的口令时，系统使用 crypt 进行同样的计算，如果

所得的结果与/etc/shadow 中存储的一致，则接受用户的登录。

步骤 3：什么是弱口令：弱口令就是很容易被猜出来的口令，比如 admin，guest，flower,123456，beauty 等等。在口令猜测或者破解时，一般不会穷尽所有字符

的所有排列组合，一般都使用一个弱口令字典，字典中包含了常被用来作为口令

的字符串。可以到网上查找口令字典。

步骤 4：弱口令检查的原理

对每个用户而言，逐个将那些经常被用来作为口令的词，使用 crypt 做运算，

如果运算结果与/etc/shadow 所存储的加密后的口令相同，则口令被猜出来。

步骤 5：用 c 语言编写实现口令检查的程序

关键部分的代码如下：

通过读取/etc/shadow 的每一行，获得一个关于用户的数据结构，存放在 pwd

中。

char saltstr[21]; //存放种子，最多是 20 个字节

bzero(saltstr,sizeof(saltstr));//将变了全部清 0

strncpy(saltstr,pwd->pw\_passwd,20);

//将 salt 取出,最多 20 个字节，以其中$作为边界

cp = crypt (guess, saltstr); //调用 crypt 加密函数

if (strcmp (cp, pwd -> pw\_passwd)) //与口令文件中存储的信息比较

return (0); //若不一致则返回

printf ("Warning! Password Problem: Guessed:\t%s\tpasswd: %s\n",

pwd -> pw\_name, guess); //找到口令，显示口令

步骤 6：编译程序

gcc –o passchk pass.c –lcrypt

其中-o 选项指定编译后的可执行文件名，-l 选项表示使用 crypt 函数库

步骤 7：执行口令检查程序

./passchk –P myshadow –w words –p

其中 passchk 是可执行文件名，myshadow 是口令文件，如果没有-P 选项，

则缺省使用 Linux 系统的/etc/shadow 文件，words 是存放弱口令的口令字典文

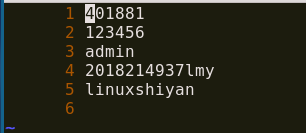
件，选项-p 表示将破解后的口令输出。

**五、实验结果**

1、将/etc/shadow拷贝到myshadow

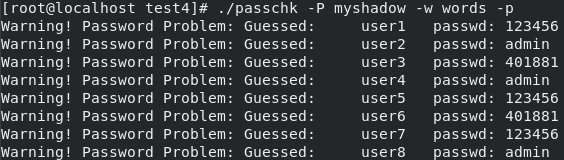


1. 编辑口令字典words



3、编译程序



1. 运行程序

**六、分析**

弱口令指的是仅包含简单数字和字母的口令，很容易被别人破解。所以我们实验的过程就是如果检测到输入的密码过于简单，在弱口令字典里面就会输出warning提示用户。

实验六**Gtk+编程和 gcc 工具的使用**

**一、实验目的**

练习 Gtk+编程和 gcc 工具的使用。

需要的知识：C/C++语言的基本技能、第 12 章 gcc 的使用与开发、第 13 章 Gtk+

图形程序设计。

**二、实验内容**

日志的查询统计是系统管理的一项重要任务。Linux 系统的日志以文本文件

的形式存在，使用 Gtk+编写一个图形界面的系统日志管理器（针对日志文件

/var/log/messages\* ）， 它 可 以 列 出 系 统 当 前 的 日 志 个 数 （ /var/log/message,

/var/log/message.1, /var/log/message.2,……共多少个），对指定的日志文件可以查

询哪些条目含有（include）指定信息（比如含有“succeed”的条目），哪些不含

有(exclude)指定信息。

图 5-1 是一个示例程序的效果图：

 数字 1 为一个标签，显示日志文件的数量；

 显示 succeed 的单行文本框用来输入包含的指定信息；

 显示 kernel 的单行文本框用来输入不包含的指定信息；

 点击标签为“/var/log/messages”的按钮后，可以选择日志文件；

 点击标签为“View”的按钮后，可以根据输入的条件查看日志内容；

下面的多行文本框用来显示日志内容。

图 5-1 程序效果图

程序目录结构如图 5－ 2 所示：图 5－ 1 程序目录结构图

其中：

 文件 main.cpp：为程序的主函数所在文件，通过对其它文件的调用启动和结

束程序；

 目录 core：其中的文件分为三组，每一组的一个.h 文件和一个.cpp 文件共同

定义了一个类，其中的类名和对应的文件名相同。目录 core 下定义的类实现

了与界面无关的大部分功能。包括：检索日志文件个数、检索和根据条件过

滤日志内容等作用。

 目录 widgets：包括两个文件，实现程序界面，并和目录 core 下的类交互，

利用图形界面接收用户的输入和显示程序的运行结果。

**三、实验环境**

**VMWARE** **Fedora11系统**

**四、实现步骤**

1) 根据目录结构创建目录和文件；

为了方便组织程序代码，可以将本实验的代码放到单独的目录下。因此，先

建立一个目录作为项目的根目录，本例中根目录命名为：ViewLogGtk。

• 在当前工作目录下，建立项目根目录：***mkdir ViewLogGtk***

• 进入根目录：***cd ViewLogGtk***

• 创建空白主程序文件：***touch main.cpp***

• 创建目录 core：***mkdir core***

• 创建目录 widgets：***mkdir widgets***

• 进入目录 core：***cd core***

• 创建目录 core 下的空白程序文件：

***touch Util.h***

***touch Util.cpp***

***touch LogRecord.h***

***touch LogRecord.cpp***

***touch LogContent.h***

***touch LogContent.cpp***

• 进入目录 widgets：***cd ../widgets***

• 创建目录 widgets 下的空白程序文件：

***touch widget.h***

***touch widget.cpp***

当前目录./

widgets/

main.cpp

Util.h，Util.cpp

core/

LogRecord.h，LogRecord.cpp

widget.h，widget.cpp

LogContent.h，LogContent.cpp• 目录文件创建完毕。

2) 编辑并实现代码文件：这里仅给出关键部分的代码。

Util.h 定义了类 Util：

类 Util 中只定义了一个静态的方法：queryFileCount（）用来查询指定目录下指

定类型的文件个数。其中方法的第一个参数为指定查询文件的目录，第二个参数

为指定文件名中包含的字符串。对于本例，第一参数固定的传入字符串

“/var/log/”，第二个参数固定的传入字符串“message”，即查询目录/var/log/下

文件名中包含 message 的文件。

LogRecord.h

1 #ifndef LOGRECORD\_H

2 #define LOGRECORD\_H

3 #include <string>

4 using namespace std;

5 class LogRecord

6 {

7 private:

8

string contents;

9 public:

10

LogRecord(char\* contents);

11

~LogRecord(void);

12

string getContents(void);

13

LogRecord(string contents);

14 };

15 #endif

类 LogRecord 代表日志中每一行。构造函数 LogRecord（char\*）和 LogRecord

（string）分别提供了利用 char\*类型和 string 类型形成日志行记录的两种方法。

其中私有字符串成员 contents 储存日志记录的一行字符串，公有方法 getContents

（）对外提供了一个获得字符串记录内容的接口。

LogContent.h

1 #ifndef LOGCONTENT\_H

2 #define LOGCONTENT\_H

3 #include<string>

4 #include<list>

5 #include"LogRecord.h"

Util.h

1 #ifndef UTIL\_H

2 #define UTIL\_H

3 class Util{

4 public:

5 static int queryFileCount(const char\* dir,const char\* keyword);

6 };

7 #endif6 using namespace std;

7 class LogContent

8 {

9 private:

10

string contains; //记录中需要出现的关键字

11

string notcontains; //记录中不能出现的关键字

12

string file\_name; //记录文件名

13

list<LogRecord> records; //符合过滤条件的记录集合

14 public:

15

LogContent(void);

16

~LogContent(void);

17

void setFilter(string contains, string not\_contains); //设置过

滤条件

18

void set\_file\_name(string file\_name); //设置检索的日志记

录文件名

19

int fill\_records(void); //根据文件、过滤条件填充记录集合

20 private:

21

int filter\_record(LogRecord record); //判断记录行是否满足

过滤条件

22 public:

23

list<LogRecord> get\_records(void); //得到记录集合

24 };

25 #endif

类 LogContent 是检索日志文件、过滤日志行记录的主要场所。它利用

setFilter(string,string)，设置日志行记录中必须出现的关键字和必须不出现的关键

字，利用 set\_file\_name（string）设置日志文件的文件名称。检索关键字和日志

文件设置完毕后，就可以利用 fill\_records（）检索日志文件并填充日志行记录集

合。填充完毕，可以使用方法 get\_records（）得到检索的结果。

widget.h

1

#ifndef WIDGET\_H

2

#define WIDGET\_H

3

#include <string>

4

#include <iostream>

5

#include <gtk/gtk.h>

6

#include "LogContent.h"

7

using namespace std;

8

//fields

9

string file\_name; //日志文件名

10

string kw\_contains; //日志过滤条件中，需要包含的关键字

11

string kw\_not\_contains; //日志过滤条件中，不需要包含的关键字

12

int file\_num; //指定目录下，特定文件的个数

13

GtkWidget\* window; //主界面窗口

14

GtkWidget\* lbl\_file\_num; //文件个数标签

15

GtkWidget\* btn\_choose\_file; //选择文件按钮16

GtkWidget\* btn\_view; //查看日志文件按钮

17

GtkWidget\* ety\_contains; //包含关键字输入文本框

18

GtkWidget\* ety\_not\_contains; //不包含关键字输入文本框

19

GtkWidget\* scroll\_win; //用于显示日志的滚动窗口

20

GtkWidget\* text; //显示日志多行文本框

21

GtkWidget\* table; //用于摆放界面控件的表格布局

22

GtkWidget\* dialog; //用于选择文件的对话框

23

GtkTextBuffer\* buf; //显示日志的文字缓冲区

24

LogContent logContent; //完成文件检索、过滤填充的对象

25

//functions

26

void layout\_ctrls(); //放置界面控件

27

void onChooseFile(GtkWidget\* sender,gpointer data); //点击选择文件

时的动作

28

void onView(GtkWidget\* sender,gpointer data); //点击查看日志文件

按钮的动作

29

void onFileSelect(GtkWidget\* widget,GtkFileSelection\* fs);

//在对话框中选择文件的动作

30

void dispLogNum(int num); //显示日志文件个数

31

void connectHandlers(); //关联界面控件和相应的事件处理程序

32

#endif

33

#ifndef WIDGET\_H

widget.h 声明了与程序界面直接相关的变量和方法，其中的方法的实现部分即

函数体在 widget.cpp 中。两个文件相结合，完成了程序界面的生成、界面控件元

素和事件处理函数的关联、接收用户输入并反馈处理结果的功能，是用户之间交

互的部分。

main.cpp

1

#include "widget.h"

2

#include "widget.cpp"

3

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

4

gtk\_set\_locale();

5

gtk\_init(&argc,&argv);

6

layout\_ctrls(); //调用布局界面的方法，形成并显示图形用户界

面

7

gtk\_main();

8

return 0;

9

}

main.cpp 功能代码都很简单，除了常规的初始化程序环境的功能外，唯一特殊

的就是调用 widget.h 中定义的 layout\_ctrls（）方法，形成并显示界面。

3) 编译程序：

程序包含文件较多，可以采取分步编译的方式。由于代码中用到了类、string

等概念，是 C++风格的代码，因此，编译时采用 g++命令，而不是 gcc 命令。g++

命令和 gcc 命令使用方法参数信息都相同，g++用于 C++风格代码而 gcc 常用于

c 风格的代码。

假设当前工作目录为项目的根目录即 ViewLogGtk 目录下，首先编译 core 目录下文件：

• 编译类 Util：***g++ -c core/Util.cpp***

• 编译类 LogRecord：***g++ -c core/LogRecord.cpp***

• 编译类 LogContent：***g++ -c core/LogContent.cpp***

然后，编译 main.cpp 和 widgets 目录下的文件:

• ***g++ -c main.cpp -Icore -Iwidgets `pkg-config gtk+-2.0***

***--cflags --libs`***

由于 widges 目录下的文件涉及到了 Gtk+的界面部分，因此，在编译时需要加入

`pkg-config ....`参数。

最后，将得到的目标文件编译为统一的可执行文件，并将可执行文件命名为

ViewLog，命令：

• ***g++ LogRecord.o LogContent.o Util.o main.o -o ViewLog***

***-Icore -Iwidgets `pkg-config gtk+-2.0 --cflags --libs`***

4) 运行程序：

• • ***./ViewLog***

**五、实验结果与分析**

