电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 201622020410

姓 名 刘高阳

（实验） 课程名称 UNIX/LINUX操作系统编程

理论教师 肖堃

实验教师 肖堃

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：刘高阳 学号：2016220204010 指导教师：肖堃**

**实验地点：信软楼304 实验时间：2018/10/23**

**一、实验名称：linux开发环境实验**

**二、实验学时：2**

**三、实验目的：**

* + 熟悉基于Linux内核的操作系统
  + 熟悉基于GNU工具链的开发环境
    - ubuntu 11（bourne again shell）
    - gcc 4.5.2
    - gdb 7.2
  + 掌握Linux程序编译调试方法

**四、实验原理：**

**1.GCC编译工具**

GCC是GNU计划的一个项目。是一个自由编译器，如今的GCC已经是一个包含众多语言的编译器了（ C，C++，Ada，Object C，Java及Go等）。所以，GCC也由原来的GNU C Compiler变为GNU Compiler Collection

GCC主要包括:

cpp(预处理器)

gcc(c编译器)、g++(c++编译器)等编译器

binutils等二进制工具.

as(汇编器)

ld(链接器)

…

gcc命令格式：gcc [选项] <文件名>-o filename ：指定输出文件为filename.该选项不在乎gcc产生什么输出，无论是可执行文件，目标文件，汇编文件还是预处理后的C代码

如果没有使用”-o”选项,默认的输出结果是:可执行文件为”a.out”，编译后产生的目标文件是”sourcename.o”，汇编文件是 “sourcename.s”，而预处理后的C源代码送往标准输出

对于源代码main.c，可以通过如下命令编译成最终可执行文件（默认包含了预处理、编译、汇编及链接四个阶段）：

gcc main.c –o main

**调试选项**

-g：以操作系统的本地格式(stabs, COFF, XCOFF,或DWARF).产生调试信息. GDB能够使用这些调试信息.

-ggdb：ggdb选项可以插入的更为丰富的调试信息，但是生成的可执行文件有可能无法用其它调试器调试

调试选项的启动会让二进制文件的大小急剧增长

**优化选项**

-O 或者 –O1 优化选项：编译器会试图减少目标码的大小和执行时间

-O2：除了涉及空间和速度交换的优化选项,执行几乎所有的优化工作，但不进行循环展开和函数内嵌

-O3：乱序执行，循环展开的优化

调试时不能用优化选项，否则变量值和源代码无法对应

**2.GDB调试工具**

GDB是GNU计划开发的程序调试工具

GDB可以完成以下四个方面的功能：

- 启动程序，可以按照自定义的要求随心所欲的运行程序

- 可让被调试的程序在所指定的调置的断点处停住（断点可以是条件表达式）

- 当程序被停住时，可以检查此时程序中所发生的情况

- 动态的改变程序的执行环境

直接在shell中运行gdb命令，进入gdb界面后用 file program 装载程序。

在shell中启动gdb并加载可执行文件

**gdb <program>**

用gdb同时调试一个运行程序和core文件（core是程序非法执行后core dump后产生的文件）

**gdb program core**

调试正在运行的进程

**gdb program <processid>**

进入gdb后用 attach <processid> 调试正在运行的进程

常用命令：

**help [command]**

在gdb中查询command的用法,如果不加command，则列出gdb的命令分类。可以用 help 分类名查看此分类下所有的gdb命令

quit或者ctrl+d 退出gdb

**break 命令**

功能：断点设置命令break(缩写 b)，当gdb执行到该断点时会让程序暂停运行。此时程序员可以查看运行中程序的情况。

格式：break [LOCATION] [thread THREADNUM] [if CONDITION]

[LOCATION]：

linenum（行号）

function name(函数名)

filename:linenum

filename:function

class:function（c++）

b 123 b main b increase:main b increase:123

[thread THREADNUM] 调试多线程程序时，切换到哪个线程或者在那个线程中设置断点。 break frik.c:13 thread 28

[if CONDITION]： 当条件满足时，断点才生效。一般称为条件断点。 CONDITION跟C语言一样

b 123 if index==2

当index为2时，程序在123行停下。

**Watchpoint命令**

watchpoint称为观察点，当观察对象的值有变化时，程序立即停止执行。

watch <expr> ：为表达式（变量）expr设置一个观察点。一旦表达式值有变化时，马上停住程序。

rwatch <expr> ：当表达式（变量）expr被读时，停住程序。

awatch <expr> ：当表达式（变量）的值被读或被写时，停住程序。

info watchpoints ：列出当前所设置了的所有观察点。

**GDB常用命令清除禁止断点或观察点**

clear [linenum] [function name] 清除所有断点,不会清除watchpoints。

delete <num> 清除编号为num的断点或者watchpoint。

disable <num> 禁止某个断点。

enable <num> 开启某个断点。

**GDB调试命令**

* step 单步调试命令，一次执行一行程序。
* next 单步调试命令，但跳过函数调用。
* finish 单步调试时直接从一个函数中返回
* disassemble 显示汇编代码。
* backtrace或者bt 查看目前程序的堆栈情况。
* whre查看目前程序的堆栈情况。
* up/down 向上或者向下移动一个堆栈。
* frame<num>或者f 移动到第num个堆栈。

当移动到某个堆栈时，便可以用gdb命令查看在那个堆栈中的局部变量

* dir 源代码路径：gdb默认有$cdir和$cwd两个源代码搜索路径，如果你要调试程序的源代码不在这两个路径中，可用dir命令增加 。
* r [[参数1]…[参数n]] ：在gdb中运行已经装载的可执行文件，参数为程序所需的参数。
* info [\*\*] 查看设置的断点，函数名称，类名等。

info b 查看已经设置的断点名称

**五、实验内容：**

在Linux下通过C语言程序设计应用程序，其功能主要包括：通过定义单向链表的数据结构，设计创建链表、插入结点、遍历结点等基本算法

在Linux下通过GNU工具链编译、调试、运行该应用程序

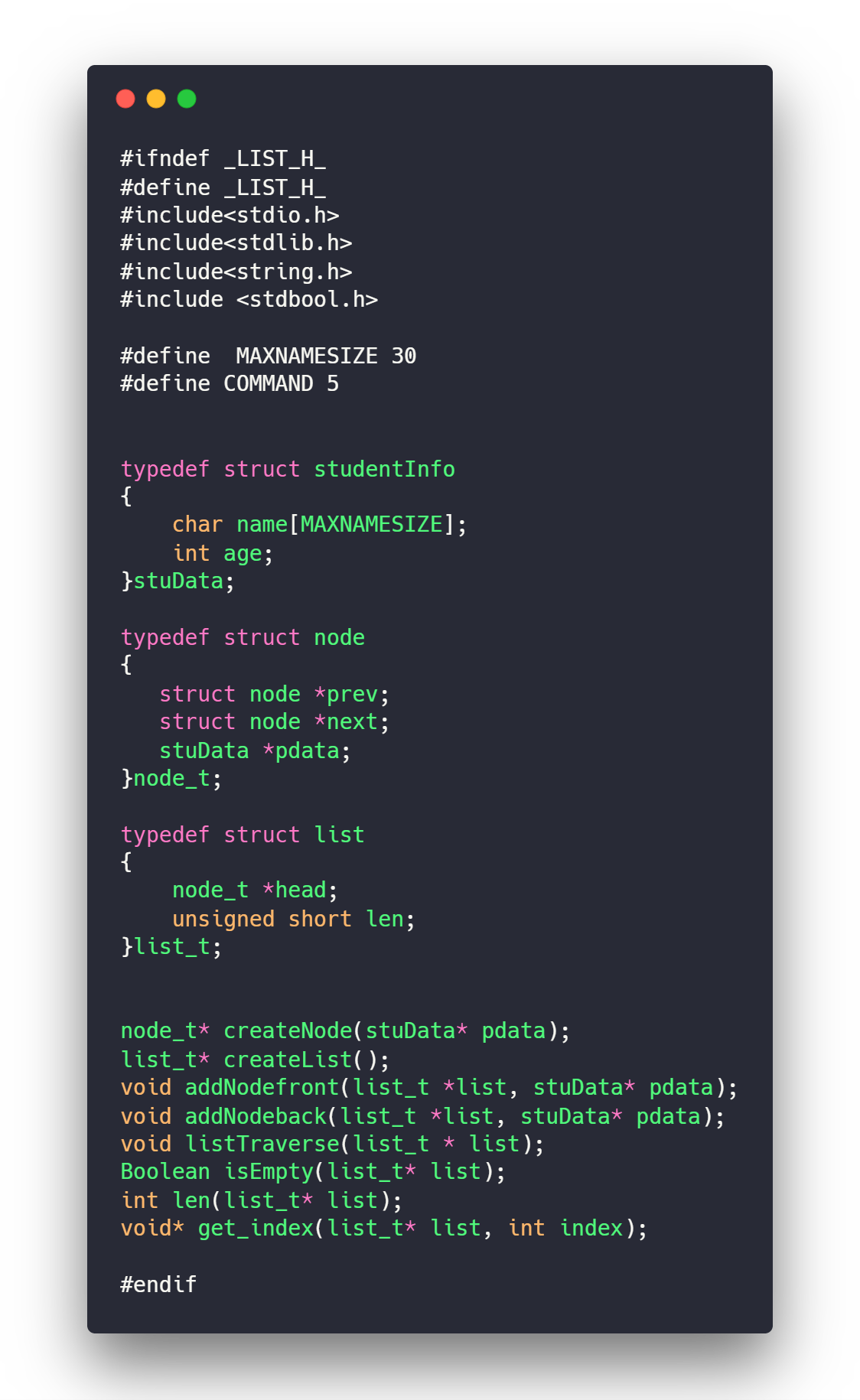
**六、实验器材（设备、元器件）：**

PC、VMware

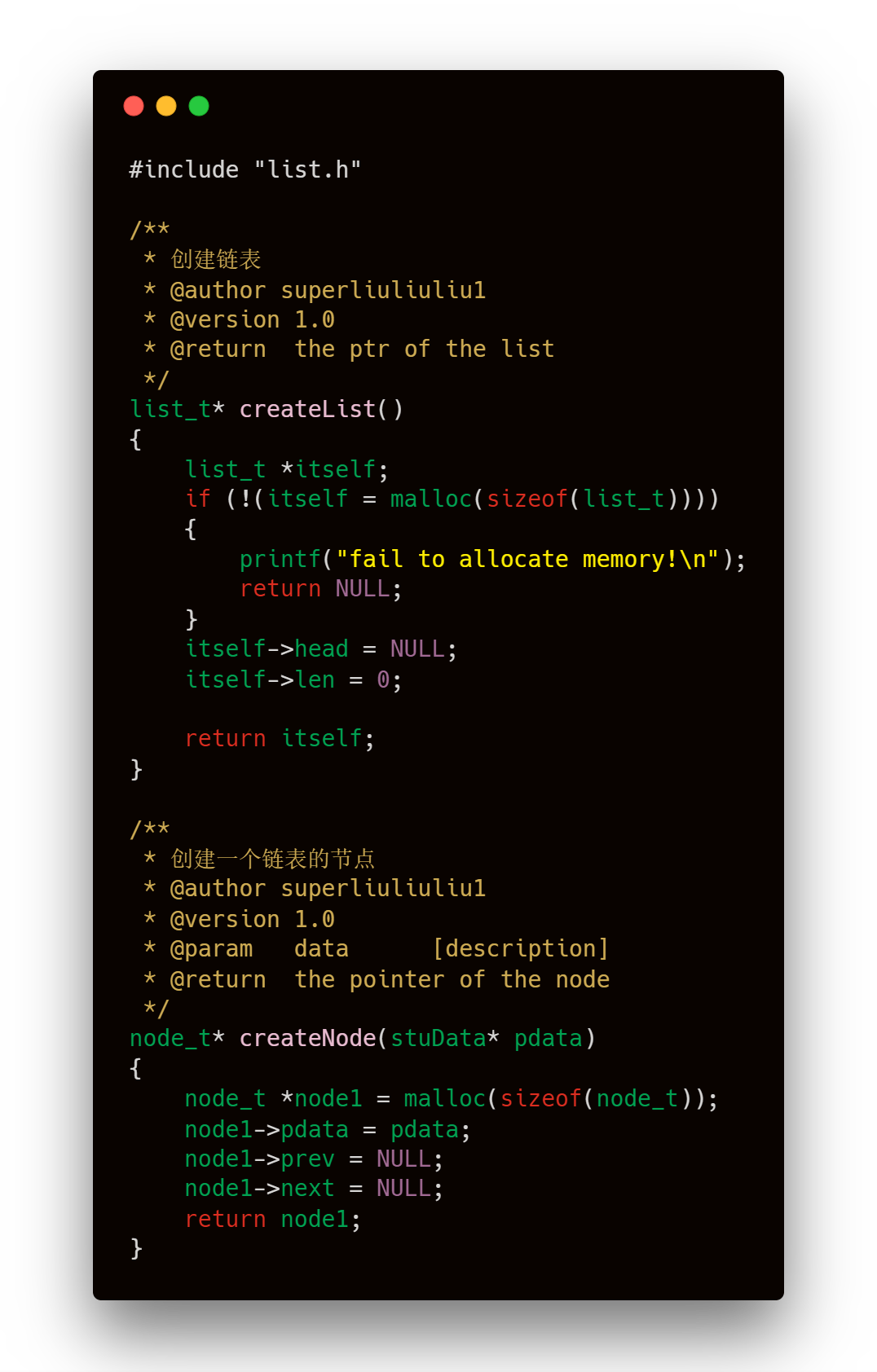
**七、实验步骤：**

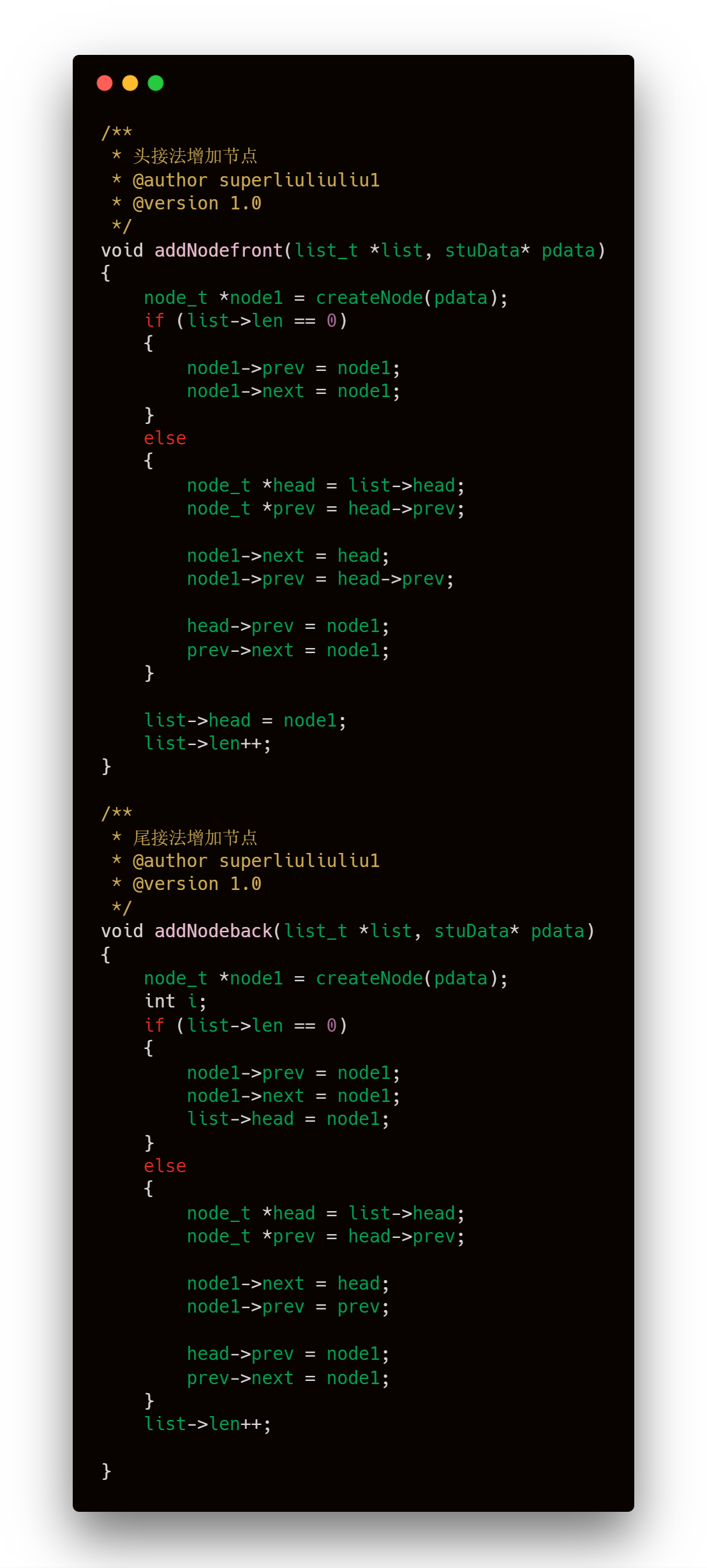
1. 编写代码

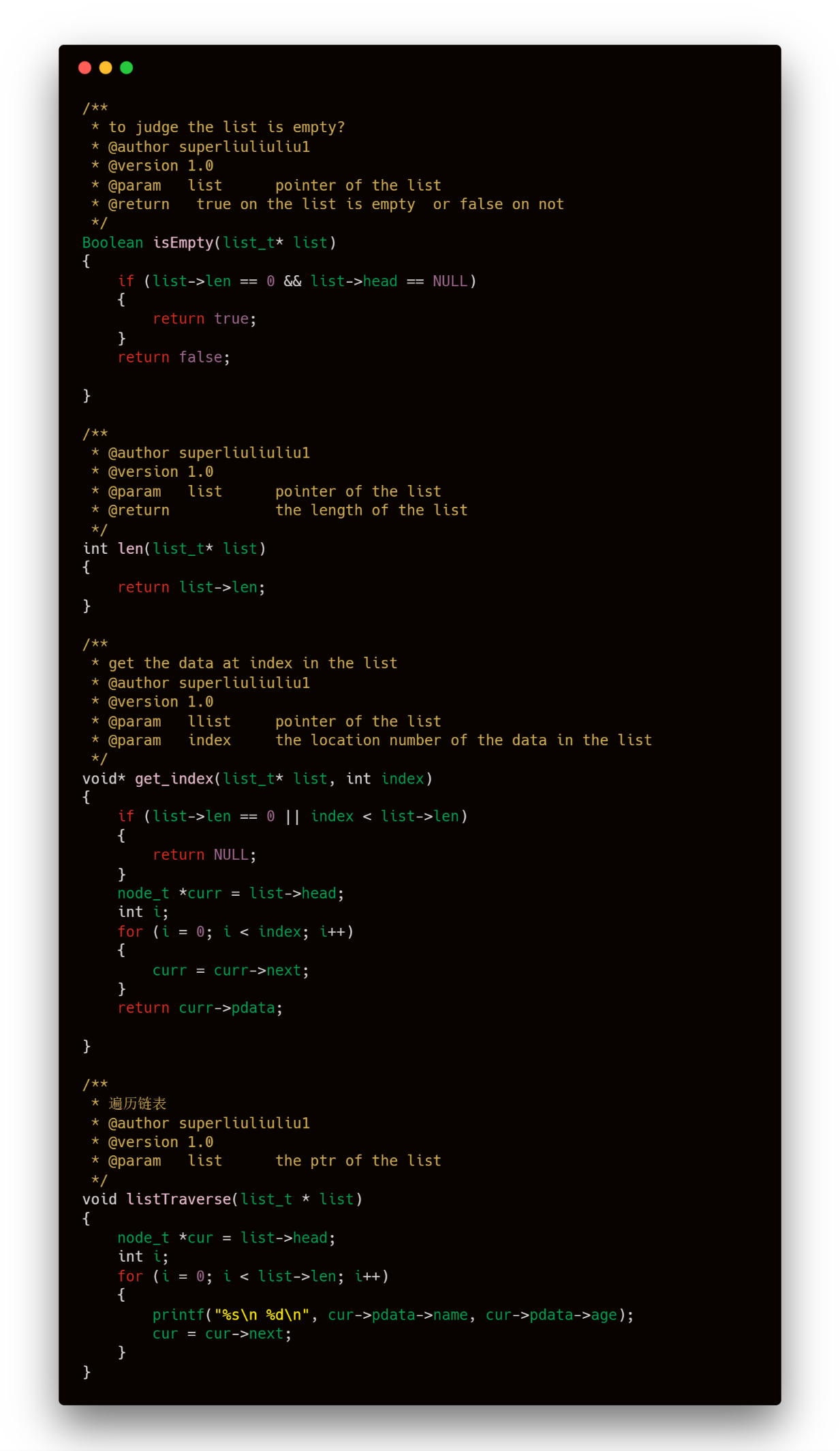
根据实验的要求编写了链表的相关代码，实现的链表为双向链表，链表头文件代码如下：（为了代码看的比较清晰，将其转换成了图片，代码都在附件之中）



链表list.c代码如下：





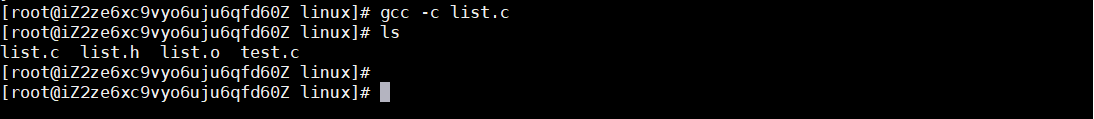


测试代码如下：

2. 编译调试（具体的编译调试步骤在实验结果与分析中展示）

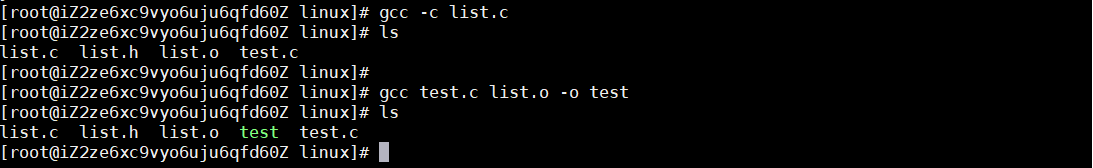
GCC编译部分

首先编译list.c 链表库使用 gcc -c list.c 执行结果如下：



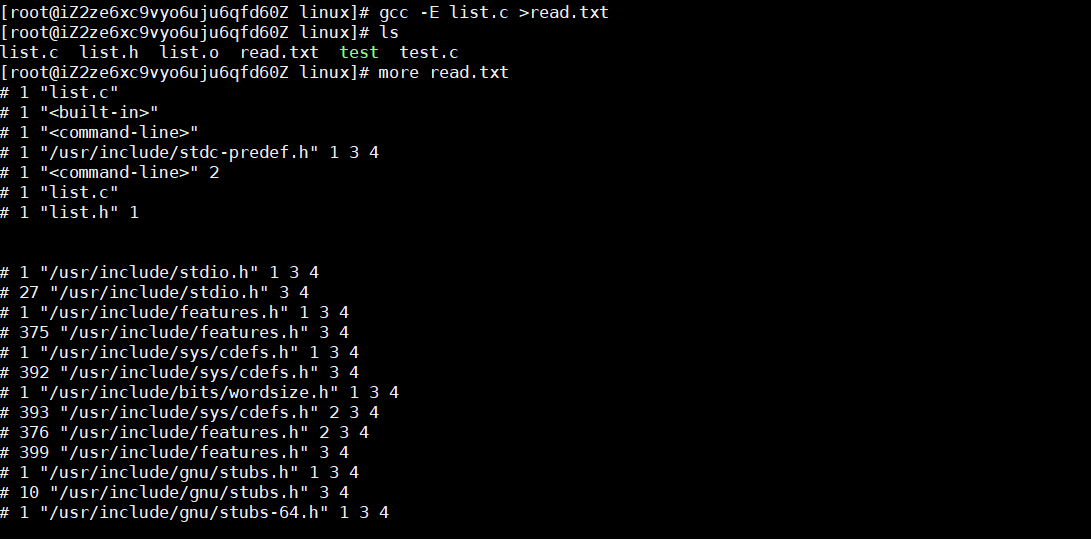
会发现生成了一个目标文件list.o文件

接着链接目标文件生成一个可执行文件如下图所示：

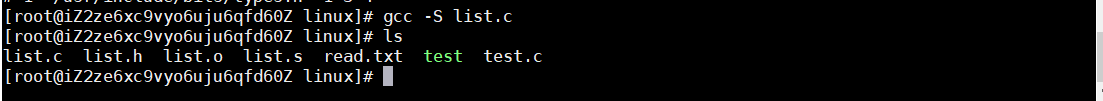


接下来测试gcc 编译的其他功能：

使用-E参数只激活预处理过程并将它重定向到一个read.txt文本之中并查看结果，执行结果如下：



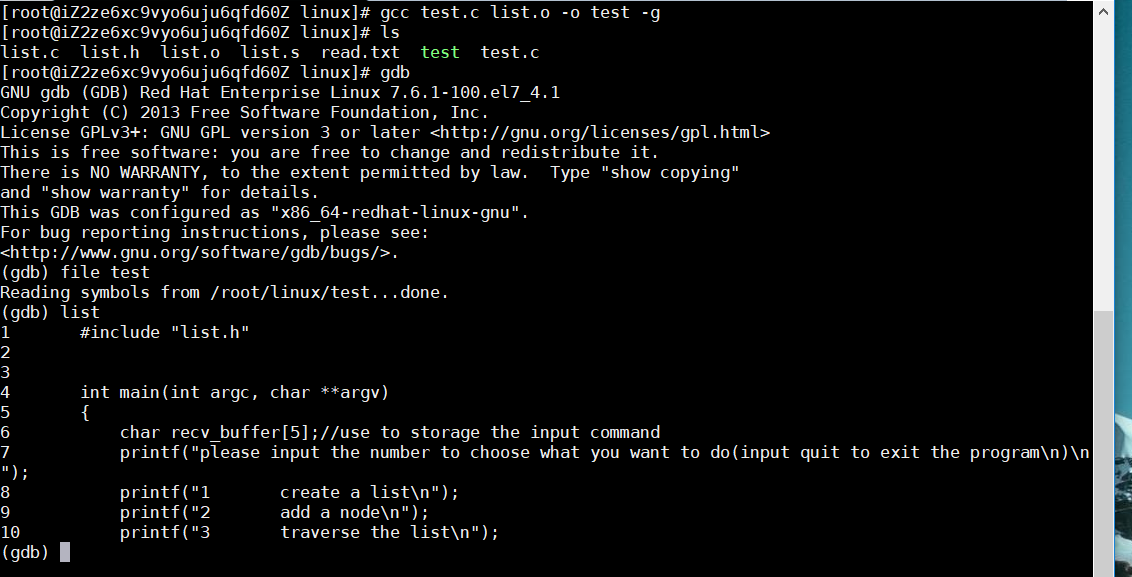
使用-S参数只激活预处理和编译过程，此时会生成一个后缀为.s的文件执行结果如下：



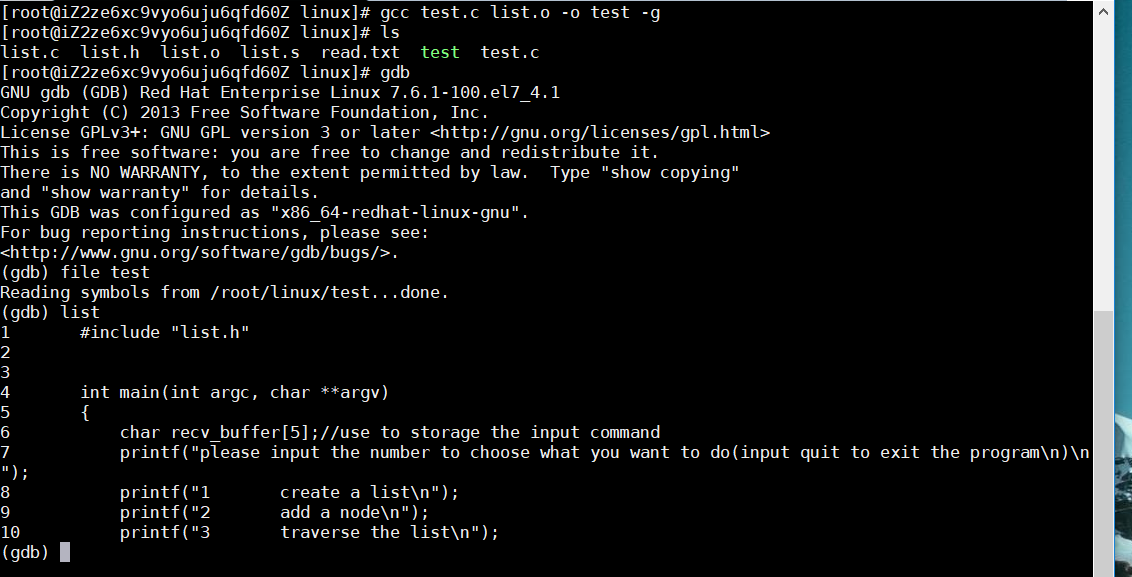
要想能够在gdb中调试程序需要注意在编译程序的时候加上-g参数

GDB调试部分

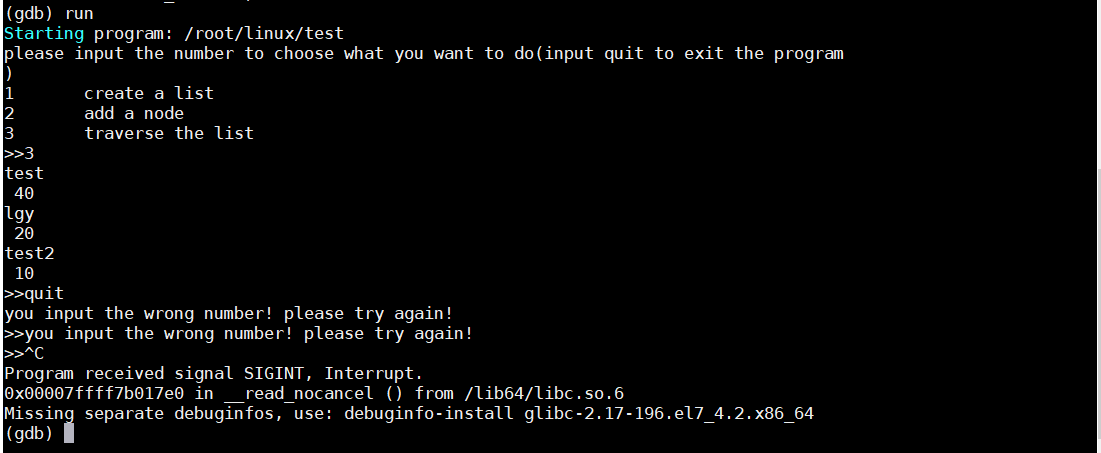
进入gdb并查看程序的源代码：



默认设置的可查看代码行数为10 通过set listsize <count>能够更改默认设置，使用举例如下：（list命令会继承着上一个命令继续查看代码）



在gdb中使用run命令可以执行可执行文件：（为了方便测试，对于链表中的数据没有采取命令行输入的方式，而是直接在main函数中定义了相应的数据结构体）



暂停/恢复程序的运行：

调试程序中，暂停程序运行是必须的，GDB可以方便地暂停程序的运行。你可以设置程序的在哪行停住，在什么条件下停住，在收到什么信号时停往等等。以便于你查看运行时的变量，以及运行时的流程。

当进程被gdb停住时，你可以使用info program 来查看程序的是否在运行，进程号，被暂停的原因。

在gdb中，我们可以有以下几种暂停方式：断点（BreakPoint）、观察点（Watch Point）、捕捉点（Catch Point）、信号（Signals）、线程停止（Thread Stops）。如果要恢复程序运行，可以使用c或是 continue命令。

break <function>

进入指定的函数暂停程序的运行

break <linenum>

进入指定的函数停止程序的运行

break +offset

break -offset

在当前行号的前面或后面的offset行停住。offiset为自然数。

break filename：linenum

在源文件filename的linenum行处停住。

break filename：function

在源文件filename的function函数的入口处停住。

break \*address

在程序运行的内存地址处停住。

break

break命令没有参数时，表示在下一条指令处停住。

break ... if <condition>

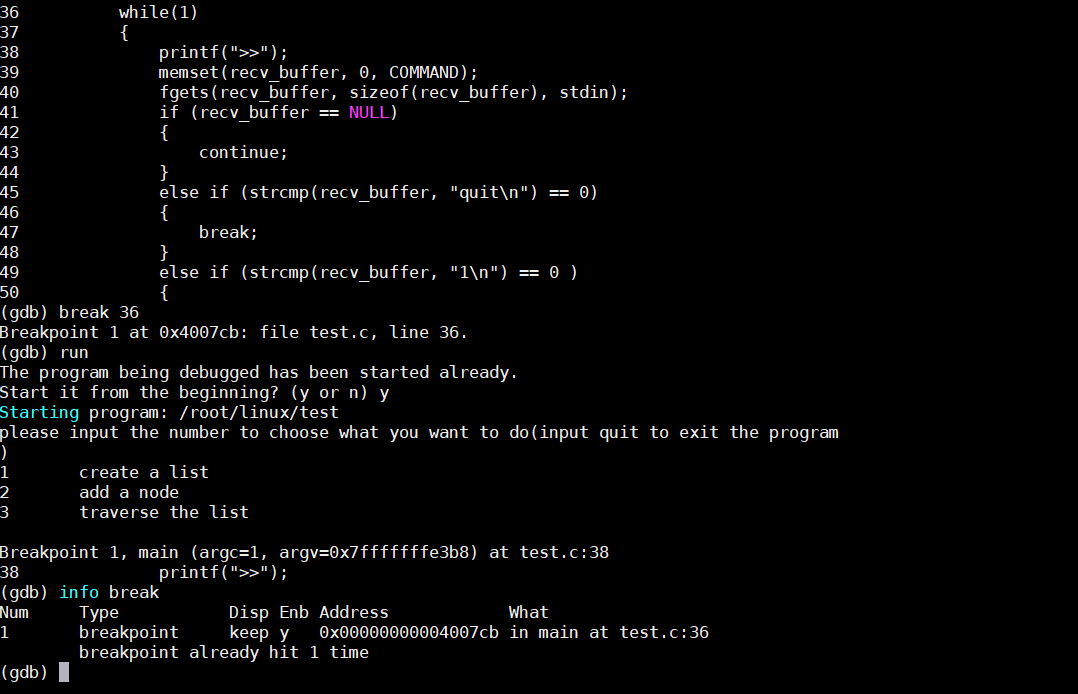
...可以是上述的参数，condition表示条件，在条件成立时停住。比如在循环体中，可以设置break if i==100，表示当i为100时停住程序。

查看断点时，可使用info命令，如下所示：（注：n表示断点号）

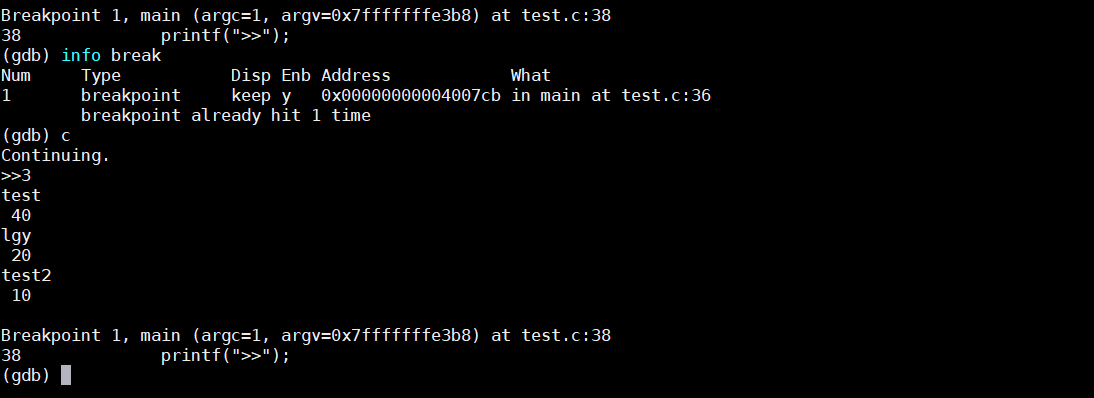
info breakpoints [n]

info break [n]

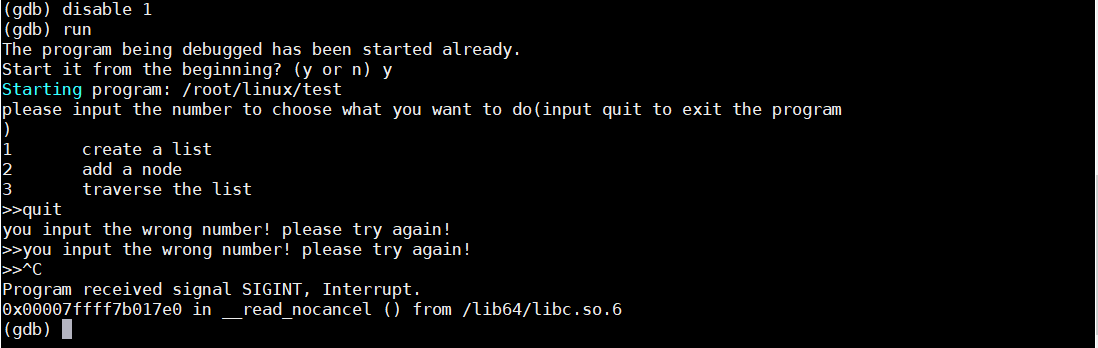
在指定行数设置断点的执行结果：（之后查看断点的相关信息）



使用 continue来恢复程序的运行：

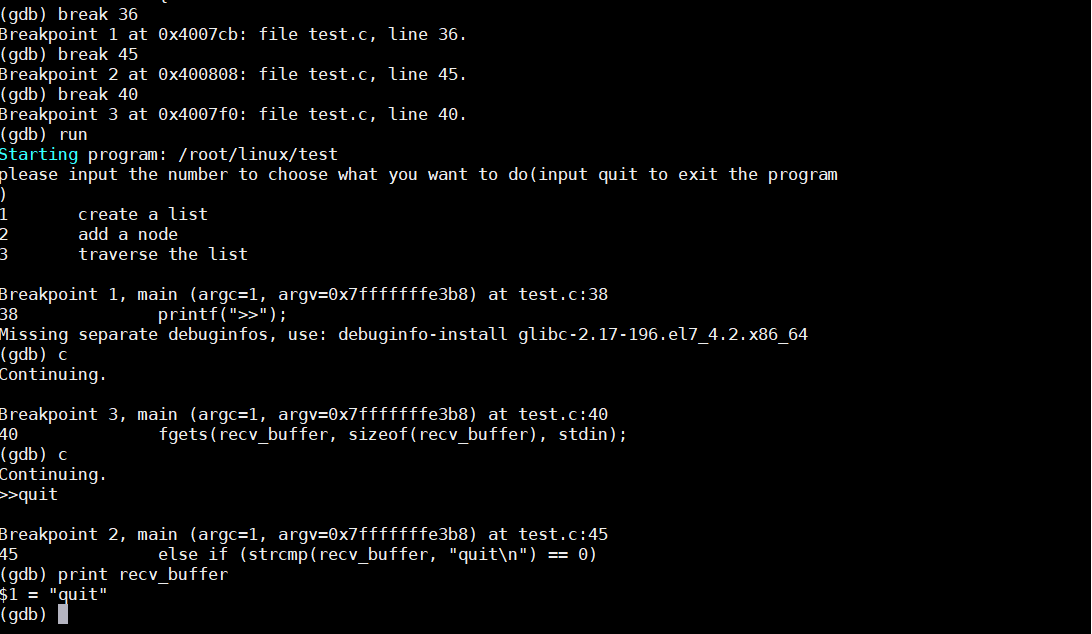


使用clear能够清楚所有的停止点，但是一般使用disable num 来暂停停止点的使用，如果需要恢复使用，使用enable num即可：

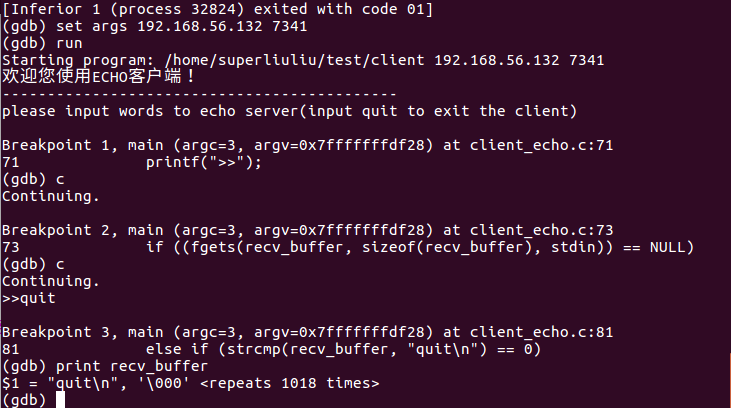


其他设置停止点的手段还有设置观察点（watchpoint）、设置捕捉点（catchpoint）暂时没有使用，不再展示。

经过上述的调试，发现输入quit不能够退出程序的使用，那究竟是什么原因呢？为了找出这个问题，我在test.c中设置了三个断点，并进行相关的调试：

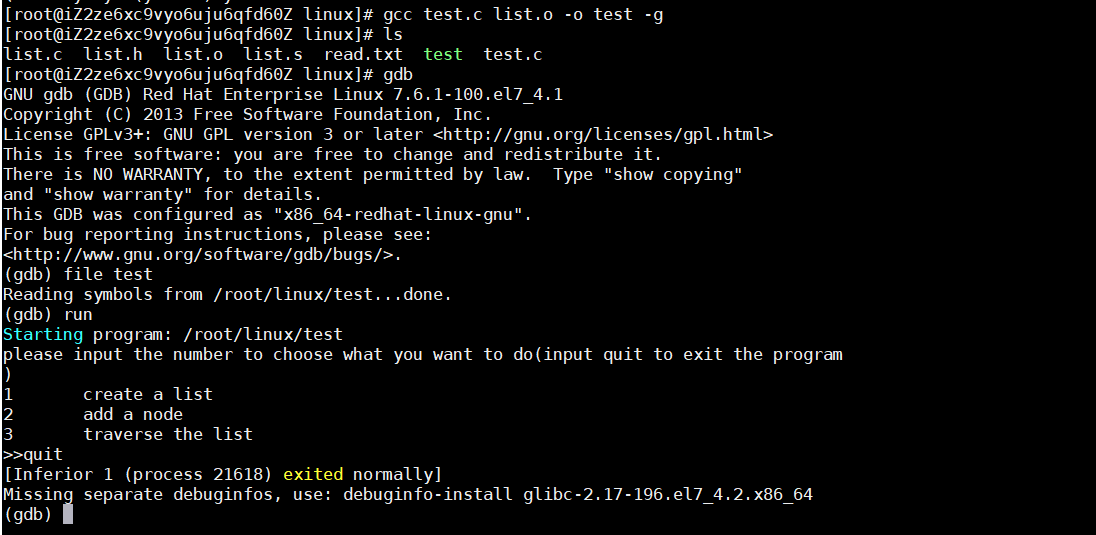


由上图可知，接收到的recv\_buffer = “quit”而不是“quit\n”,因此在用户输入quit时程序并不会退出

出现这个问题的原因是我之前也写过一个类似的问题，输入quit退出程序，但是在代码中是strcmp(recv\_buffer, "quit\n") == 0才可以成功退出，为了搞清楚这个问题我又对当初那个程序进行了gdb调试，调试结果如下

发现在里recv\_buffer = “quit\n”,这个我就有点搞不明白了？先来标注一下。

现在只有先修改此代码并重新编译运行得到下图：



证明我的上述猜测是正确的，通过这种方式很快找到了错误所在之处。

GDB的功能十分强大，也十分便捷好用，我所接触使用的只是凤毛麟角，以后要多利用GDB来调试程序。

**八、总结及心得体会：**

经过这次实验，我深刻体会到了gdb的强大，以前虽然听过老师谈过gdb，但是一方面一直找不到入门的方式，另一方面在linux上编程的次数还是比较的少的，对于命令行的使用还是有一点恐惧的，但是在使用之后，感觉就好像给我打开了一个新世界的大门，未来还有很多的东西等待着我去探索。

gdb调试还有许多功能，我还没有使用过，甚至是不曾了解，在以后的学习中，我会多用gdb调试，多在linux上编程，这不仅能够提高我编程的效率，也有利于我再一次深入理解程序运行的逻辑。

**九、对本实验过程及方法、手段的改进建议：**

由于链表之前自己写过一个小的库，所以本次实验编译调试遇到的错误较少，可能对于gdb的功能的使用并不全面，在以后的实验中，我会多用GDB调试，全面的分析自己的程序，从而找到错误所在。

**报告评分：**

**指导教师签字：**

电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 201622020410

姓 名 刘高阳

（实验） 课程名称 UNIX/LINUX操作系统编程

理论教师 肖堃

实验教师 肖堃

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：刘高阳 学号：2016220204010 指导教师：肖堃**

**实验地点：304 实验时间：2018/10/30**

**一、实验名称：文件I/O实验**

**二、实验学时：2**

**三、实验目的：**

掌握Linux应用程序命令行参数传递方法

掌握POSIX API中文件I/O操作方法，包括：打开文件、关闭文件、创建文件、读写文件、确定和改变文件当前位置

**四、实验原理：**

1.应用程序从命令行获取参数

2.open函数

头文件：fcntl.h

int open( const char \*pathname, int oflag, …);

该函数打开或创建一个文件。其中第二个参数oflag说明打开文件的选项（第三个参数是变参，仅当创建新文件时才使用）

O\_RDONLY:：只读打开；

O\_WRONLY：只写打开；

O\_RDWR：读、写打开；

O\_APPEND：每次写都加到文件尾；

O\_CREAT：若此文件不存在则创建它，此时需要第三个参数mode，该参数约定了所创建文件的权限，计算方法为mode&~umask

O\_EXCL：如同时指定了O\_CREAT，此指令会检查文件是否存在，若不存在则建立此文件；若文件存在，此时将出错。

O\_TRUNC：如果此文件存在，并以读写或只写打开，则文件长度0

返回值是打开文件的文件描述符

3. read函数

头文件unistd.h

ssize\_t read( int filedes, void \*buf, size\_t nbytes);

read函数从打开的文件中读数据

如读取成功，返回实际读到的字节数。一般情况下实际读出的字节数等于要求读取的字节数，但是也有例外：读普通文件时，在读到要求字节数之前就到达文件尾

如已到达文件的末尾或无数据可读，返回0（可以作为文件读取是否完成的判断条件！）

如果出错，返回-1

读操作完成后，文件的当前位置将从读之前的位置加上实际读的字节数。

4.write函数

头文件unistd.h

ssize\_t write( int filedes, const void \*buf, size\_t nbytes);

write函数向打开的文件中写数据

写入成功返回实际写入的字节数，通常与要求写入字节数相同

写入出错返回－1，出错的原因可能是磁盘满、没有访问权限、或写超过文件长度限制等等

写操作完成后，文件的当前位置将从写之前的位置加上实际写的字节数。

5.close函数

头文件unistd.h

int close( int filedes );

该函数关闭打开的一个文件

关闭文件后，就不能通过该文件描述符操作该文件了

6.lseek函数

头文件unistd.h

off\_t lseek( int filedes, off\_t offset, int whence);

进程中每打开一个文件都有一个与其相关联的“文件当前位置”

打开文件时，文件当前位置默认为文件头（0），如果指定了O\_APPEND选项则文件当前位置变为文件尾（文件长度），

lseek函数用于设置或查询文件当前位置。

对参数的解释与参数whence的值有关：

若whence是SEEK\_SET,则将该文件当前位置设置为文件头+offset（以字节为单位）

若whence是SEEK\_CUR,则将该文件当前位置设置为文件当前位置+offset （以字节为单位）

若whence是SEEK\_END,则将该文件当前位置设置为文件尾+offset个字节（以字节为单位）

**五、实验内容：**

利用POSIX API在Linux系统上编写应用程序，仿写cp命令的部分功能，将源文件复制到另外一个文件或复制到另外一个目录。源文件路径和目标文件路径通过命令行参数来指定

1、将test1.text复制成test2.txt：

[test@linux test]$ ./mycp /home/test1.txt /usr/test2.txt

2、将test1.txt复制到/tmp目录中：

[test@linux test]$ ./mycp /home/test1.txt /tmp(目录)

（扩展）

在实现基本功能实现的基础上，针对各种特殊情况和边界条件等进行流程的完善与优化，包括：

a) 目标文件存在时给出“是否覆盖”，“是否合并”等提示信息，用户可以选择覆盖目标文件或者将已经存在的目标文件和源文件实现合并（在目标文件的尾部实现追加写入）

b)源文件不存在时给出错误提示信息

d)源文件是目录时给出错误提示信息

**六、实验器材（设备、元器件）：**

PC、远程服务器

**七、实验步骤：**

1.程序的运行流程图如下：



2.根据流程图编写源代码

3.编译代码并测试运行情况

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

1.实验代码如下：

#include<stdio.h>

#include<string.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<stdlib.h>

#define BUFFSIZE 1024

void error(char\* message);

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

{

int sourcefd, destfd;

struct stat s\_buff;//stat结构体用来存储文件的相关属性信息

char command[5];

ssize\_t numread;

char buffer[BUFFSIZE];

if (argc != 3 || strcmp(argv[1], "--help") == 0)

{

printf("使用指南\n");

printf("Usage: %s [source file path] [dest file path]\n", argv[0]);

exit(0);

}

//判断输入的源文件是否是目录

stat(argv[1], &s\_buff);

if (S\_ISDIR(s\_buff.st\_mode))

{

error("The source file is a directionry");

}

//对应源文件不存在的情况

sourcefd = open(argv[1], O\_RDONLY);

if (sourcefd == -1)

{

error("Error on find the source file");

}

//判断目标文件是否存在，然后做出选择

if (access(argv[2], F\_OK) == 0)

{

printf("目标文件已经存在，输入y去拼接文件（默认直接覆盖文件）\n");

if ((fgets(command, sizeof(command), stdin)) == NULL)

{

error("Error on input");

}

if (strcmp(command, "y\n") == 0)

{

//以合并的方式打开文件

destfd = open(argv[2], O\_WRONLY | O\_APPEND);

printf("将以合并的形式对文件操作\n");

if (destfd == -1)

{

error("Error on output");

}

lseek(destfd, 0, SEEK\_END);//改变文件的偏移量至文件尾部，已达到拼接的目的

}

else

{

//以覆盖的方式打开文件

destfd = open(argv[2], O\_WRONLY | O\_TRUNC);

if (destfd == -1)

{

error("Error on output");

}

}

}

else

{

destfd = open(argv[2], O\_WRONLY | O\_CREAT | O\_EXCL | O\_TRUNC);

if (destfd == -1)

{

error("Error on output");

}

}

//初始化buffer

memset(buffer, 0, BUFFSIZE);

while ((numread = read(sourcefd, buffer, BUFFSIZE)) > 0 )

{

if (write(destfd, buffer, numread) != numread)

{

error("Error on write");

}

}

close(sourcefd);

close(destfd);

}

/\*\*

\* error

\* @author superliuliuliu1

\* @version 1.0

\* @param message the pointer

\*/

void error(char \*message)

{

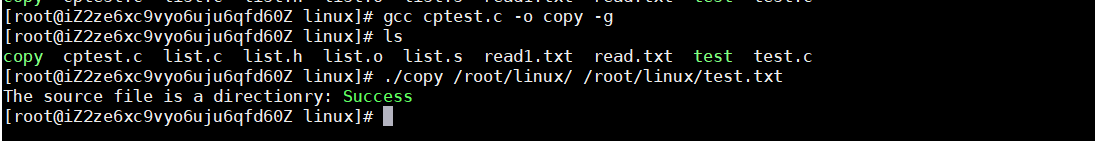
perror(message);

exit(1);

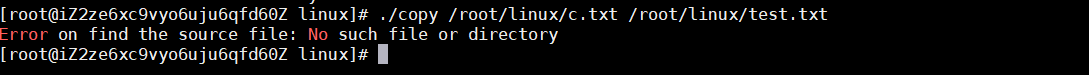
}

2.编译代码并进行功能测试

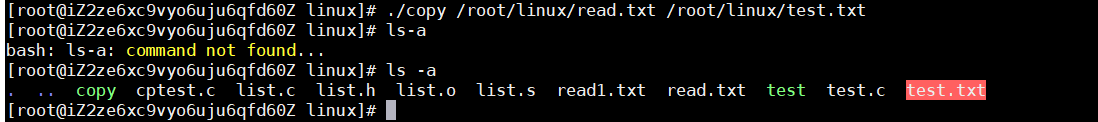
测试用例1：源文件为目录



测试用例2：源文件不存在

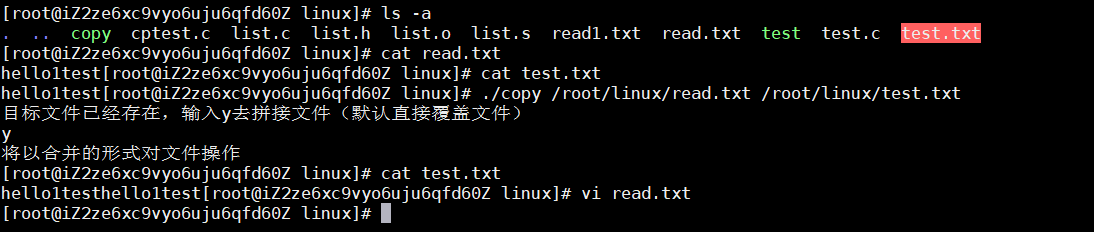


测试用例3：复制文件 目的文件不存在直接创建一个新文件

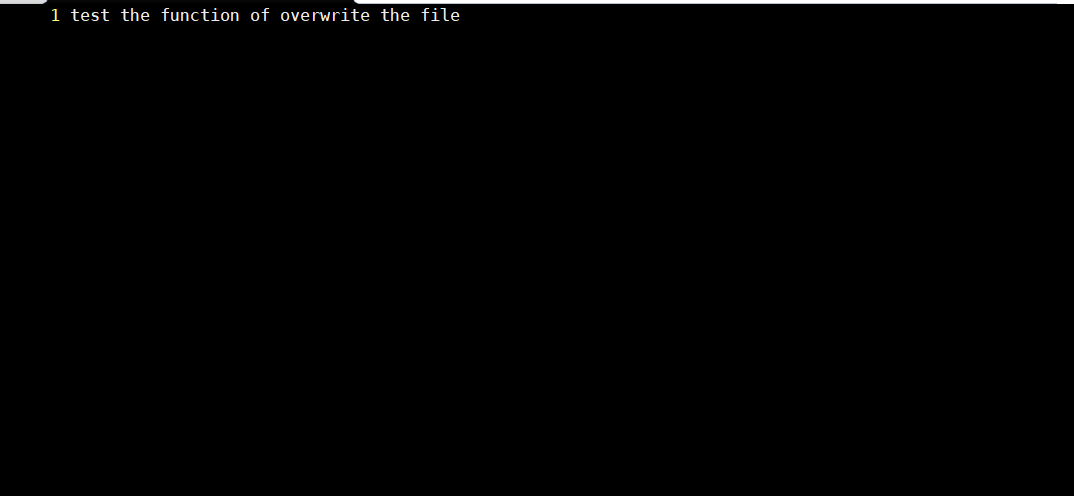


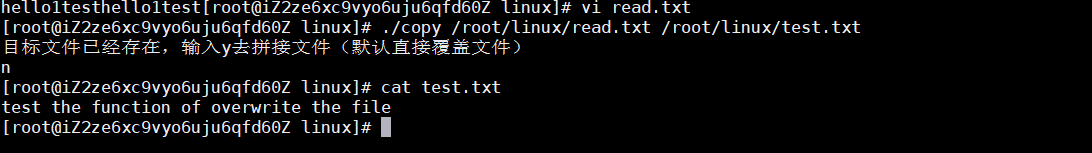
测试用例4：复制文件目的文件已存在，选择拼接形式复制文件





测试用例5：选择覆盖的形式复制文件





**九、总结及心得体会：**

通过这次实现copy功能的实验，我对文件I/O的了解更加深入，对open函数的相关参数的使用和作用有了更加深刻的理解与体会，同时也了解了文件偏移量这个概念，对lseek这个函数有了初步的了解。

**十、对本实验过程及方法、手段的改进建议：**

在实现这个函数时，并没有考虑和测试文件为二进制的形式，只是简单的实现了文本文件的复制功能，对与过大的文件的复制也没有尝试过，在接下来会对程序代码进行优化，并且做足够的测试用例。

**报告评分：**

**指导教师签字：**

电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 201622020410

姓 名 刘高阳

（实验） 课程名称 UNIX/LINUX操作系统编程

理论教师 肖堃

实验教师 肖堃

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：刘高阳 学号：2016220204010 指导教师：肖堃**

**实验地点：304 实验时间：2018/11/06**

**一、实验名称：文件与目录操作实验**

**二、实验学时：2**

**三、实验目的：**

掌握Linux目录操作方法

* 打开目录、关闭目录
* 读取目录文件

掌握Linux文件属性获取方法

* 三个获取Linux文件属性的函数

掌握文件属性解析相关的重要数据结构、函数和宏定义

**四、实验原理：**

1.获取当前文件路径：

常用函数：getcwd，get\_current\_dir\_name

头文件：unistd.h

函数定义：

char \*getcwd(char \*buf, size\_t size)

将当前的工作目录绝对路径字符串复制到参数buf 所指的缓冲区，参数size 为缓冲区大小

若参数buf 为NULL，参数size 为0，则函数根据路径字符串的长度自动分配缓冲区，并将分配的路径字符串缓冲区指针作为函数返回值（该内存区需要手动释放）

失败返回NULL

char \*get\_current\_dir\_name(void)

成功返回路径字符串缓冲区指针（该内存区需要手动释放），失败返回NULL

2.打开关闭目录：

常用函数：opendir，closedir

头文件：dirent.h

函数定义：

DIR \* opendir(const char \* name);

打开参数name指定的目录，并使一个目录流与它关联

目录流类似于C库函数中的文件流

失败返回NULL

int closedir(DIR \*dir);

关闭指定目录流，释放相关数据结构

成功返回0；失败返回-1

3.读取目录文件

常用函数：readdir

头文件：sys/types.h；dirent.h

函数定义：

struct dirent \* readdir(DIR \* dir);

读取目录流标识的目录文件

目录文件是一系列目录项的列表，每执行一次readdir，该函数返回指向当前读取目录项结构的指针

如果到达目录结尾或者有错误发生则返回NULL

重要数据结构：

**struct dirent  
{  
 ino\_t d\_ino; i节点号  
 off\_t d\_off; 在目录文件中的偏移  
 usigned short d\_reclen; 文件名长度  
 unsigned char d\_type; 文件类型  
 char d\_name[256];文件名  
};**

4.获取文件属性

常用函数：stat，lstat

头文件： sys/stat.h

函数定义：

int stat(const char \*path, struct stat \*buf);

int lstat(const char \*path, struct stat \*buf);

两个函数参数相同，功能类似

读取path参数所指定文件的文件属性并将其填充到buf参数所指向的结构体中

对于符号链接文件，lstat返回符号链接文件本身的属性，stat返回符号链接引用文件的文件属性。

重要数据结构：

**struct stat {**

**mode\_t st\_mode; 文件类型与访问权限**

**ino\_t st\_ino; i节点号**

**dev\_t st\_dev; 文件使用的设备号**

**dev\_t st\_rdev; 设备文件的设备号**

**nlink\_t st\_nlink; 文件的硬链接数**

**uid\_t st\_uid; 文件所有者用户ID**

**gid\_t st\_gid; 文件所有者组ID**

**off\_t st\_size; 文件大小（以字节为单位）**

**time\_t st\_atime; 最后一次访问该文件的时间**

**time\_t st\_mtime; 最后一次修改该文件的时间**

**time\_t st\_ctime; 最后一次改变该文件状态的时间**

**blksize\_t st\_blksize; 包含该文件的磁盘块的大小**

**blkcnt\_t st\_blocks; 该文件所占的磁盘块 数**

**};**

5.判断文件类型

是否为普通文件: S\_ISREG(st\_mode)

#define S\_IFMT 0170000

#define S\_IFREG 0100000

#define S\_ISREG(m) (((m) & S\_IFMT) == S\_IFREG)

是否为目录文件 S\_ISDIR(st\_mode)

是否为字符设备 S\_ISCHR(st\_mode)

是否为块设备 S\_ISBLK(st\_mode)

是否为FIFO S\_ISFIFO(st\_mode)

是否为套接字 S\_ISSOCK(st\_mode)

是否为符号连接 S\_ISLINK(st\_mode)

6.根据文件的uid和gid获取用户与用户组信息

常用函数：getpwuid

头文件： sys/types.h，pwd.h

函数定义：

struct passwd \*getpwuid(uid\_t uid);

输入用户ID，返回用户属性信息（passwd结构）

struct passwd{

char \*pw\_name; /\* 用户名\*/

char \*pw\_passwd; /\* 密码.\*/

\_\_uid\_t pw\_uid; /\* 用户ID.\*/

\_\_gid\_t pw\_gid; /\*组ID.\*/

char \*pw\_gecos; /\*真实名\*/

char \*pw\_dir; /\* 主目录.\*/

char \*pw\_shell; /\*使用的shell\*/};

常用函数：getgrgid

头文件： sys/types.h，grp.h

函数定义：

struct passwd \*getgrgid(gid\_t gid);

输入用户组ID，返回用户组属性信息（group结构）

struct group{

char \*gr\_name; /\*组名称\*/

char \*gr\_passwd; /\* 组密码\*/

gid\_t gr\_gid; /\*组ID\*/

char \*\*gr\_mem; /\*组成员账号\*/ }

**五、实验内容：**

利用POSIX API和文件属性解析的相关宏在Linux系统上编写应用程序，实现ls命令的部分功能（执行“ls -l”的功能），参数通过命令行参数传入，包括：

a)获取当前工作目录路径并对该目录实现遍历

b)以列表形式出当前工作目录下的所有文件（包括子目录），并显示每个文件的属性信息（文件类型、文件权限、文件硬链接数、文件所有者用户名、文件所有者所在组用户名、文件大小、文件最后修改时间）

在实现基本功能实现的基础上，针对各种特殊情况和边界条件等进行流程的完善与优化，包括：

a)根据命令行参数决定是否显示当前目录本身“.”和上级目录“..”

b)根据命令行参数决定是否显示隐藏文件（文件名以“.”作为开始的文件）

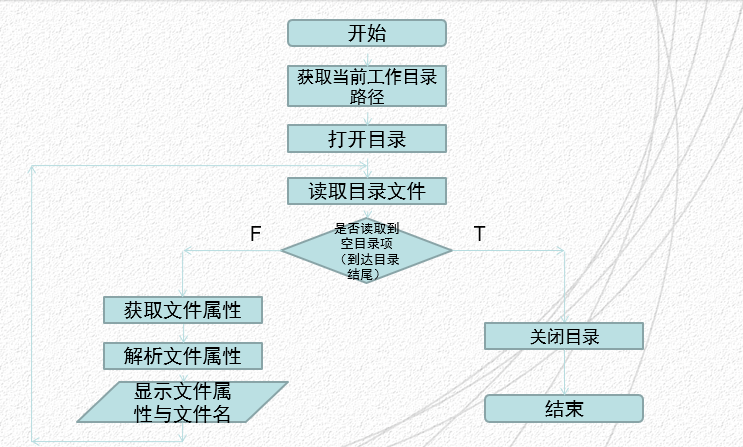
c)根据命令行参数决定是显示符号链接文件本身的属性还是符号链接指向文件的文件属性

**六、实验器材（设备、元器件）：**

PC

**七、实验步骤：**

1.根据实验要求绘制流程图如下：



2.根据流程图编写代码

3.编译测试功能

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

1.头文件说明

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<dirent.h>

#include<sys/types.h>

#include<time.h>

#include<pwd.h>

#include<grp.h>

#include<stdbool.h>

//改变字体颜色 从而实现与ls 类似的功能

#define PRINT\_GREEN(s) printf("\033[0;32m%s\033[0;32m",s); //打印目录的颜色\033[0;32;32m

#define PRINT\_BLUE(s) printf("\033[0;32m%s\033[0;34m",s);//打印链接的颜色 033[0;32;34m

//定义一个结构体 用来存储文件的相关信息

typedef struct file\_info

{

char authority[11];//文件的权限 包括文件类型 用户权限 用户组权限 其他用户权限

char \*u\_name;//文件所属用户名

char \*g\_name;//文件所属用户所在用户组名

char \*f\_name;//文件名

off\_t f\_size;//文件大小

time\_t \*latest\_time;//最近修改时间

nlink\_t link\_num;//文件的硬连接数

}file\_detail;

bool listall = false;

bool listlong = false;

bool listlink = false;

void traversal\_dir(char \*dir);

void error(char\* message);

void print\_info(file\_detail info);

char\* uid\_2uname(uid\_t u1);

char\* gid\_2gname(gid\_t g1);

为了方便打印文件的相关信息，我重新定义了一个结构体struct file\_info;用来存储文件的相关信息，之前为了实现与ls类似的功能将不同类型的文件名打印成不同颜色，但是好像出了点问题，所以后面就没有使用，对于附加参数实现扩展功能，我通过定义布尔变量将其作为扩展功能的开关来实现在有参数的要求才采用该功能。

主要的函数为

void traversal\_dir(char \*dir);该函数用来遍历目录下的所有文件，并调用下面的print\_info(),每遍历一个函数，打印出一个文件的相关信息。

void error(char\* message);该函数用来处理错误的情形，在控制台打印出相关的错误信息，便于调试。

void print\_info(file\_detail info);

char\* uid\_2uname(uid\_t u1);将uid转换为用户名

char\* gid\_2gname(gid\_t g1);将gid转换为用户组的名称

2.核心代码分析

在此部门主要介绍main函数与void traversal\_dir(char \*dir)函数。

Main函数代码如下：

/\*\*

\* 实现自己的ls命令 当前只支持查看当前目录下的文件的相关信息

\* @author superliuliuliu1

\* @version 1.0

\* @param argc [description]

\* @param argv [description]

\* @return [description]

\*/

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

{

struct stat stat\_file;

int opt\_arg;

char \*current\_dir = ".";

//获取输入的 -x参数确定命令

//-a 代表显示所有的文件

//-d 表示显示 . 和 ..

//-f 表示显示符号链接指向文件的参数

//无参数的话只显示ls命令下的文件信息

while ((opt\_arg = getopt(argc, argv, "adf")) != -1)

{

switch(opt\_arg)

{

case 'a':

{

listall = true;

break;

}

case 'd':

{

listlong = true;

break;

}

case 'f':

{

listlink = true;

break;

}

case '?':

{

printf("Unkown option!\n");

printf("-a 列出目录下所有文件的相关信息\n");

printf("-d 列出当前目录.和上级目录..\n");

printf("-f 列出符号链接指向文件的文件属性\n");

break;

}

}

}

traversal\_dir(current\_dir);

return 1;

}

Main函数主要用getopt来获取相关的扩展参数，如果获得则将相应的开关打开，对应下方实现相应的功能，之前没有了解过getopt函数，这次使用之后感觉还蛮好玩的，关键是 int opt\_arg = optget(),到后面的switch就变成了字符了，这点有点不明白。

void traversal\_dir(char \*dir)函数是实现ls功能的最主要代码：

void traversal\_dir(char \*dir)

{

DIR \*dp;

int i = 1;

struct stat stat\_file;

struct dirent \*entry;//目录的入口

//打开目录

if ((dp = opendir(dir)) == NULL)

{

error("Error on open the dir");

}

//遍历目录中的文件，并打印相关信息直至到达目录结尾

while((entry = readdir(dp)) != NULL)

{

//扩展功能 -d 即如果不输入-d 则会跳过. 和 .. 文件

if (!listlong)

{

if (strcmp(".", entry->d\_name) == 0 || strcmp("..", entry->d\_name) == 0)

{

continue;

}

}

//扩展功能 -f

//将文件的数据填充到stat\_file中 当listlink为true时使用lstat(返回符号链接文件本身的属性)，否则使用stat函数(返回符号引用文件的属性)

if (listlink)

{

lstat(entry->d\_name, &stat\_file);

}

else

{

stat(entry->d\_name, &stat\_file);

}

file\_detail info;

memset(&info, 0, sizeof(info));

//memset(info.authority, 0, 11);

//将文件的类型以及权限信息转换为字符串

info.authority[0] = '-';

if (S\_ISDIR(stat\_file.st\_mode))

{

info.authority[0] = 'd';

}

if(S\_ISCHR(stat\_file.st\_mode))

{

info.authority[0] = 'r';

}

if(S\_ISBLK(stat\_file.st\_mode))

{

info.authority[0] = 'b';

}

if(S\_ISFIFO(stat\_file.st\_mode))

{

info.authority[0] = 'p';

}

if(S\_ISLNK(stat\_file.st\_mode))

{

info.authority[0] = 'l';

}

if(S\_ISSOCK(stat\_file.st\_mode))

{

info.authority[0] = 's';

}

//进行位运算从而确定权限

info.authority[1] = (stat\_file.st\_mode & S\_IRUSR) == S\_IRUSR ? 'r': '-';

info.authority[2] = (stat\_file.st\_mode & S\_IWUSR) == S\_IWUSR ? 'w': '-';

info.authority[3] = (stat\_file.st\_mode & S\_IXUSR) == S\_IXUSR ? 'x': '-';

info.authority[4] = (stat\_file.st\_mode & S\_IRGRP) == S\_IRGRP ? 'r': '-';

info.authority[5] = (stat\_file.st\_mode & S\_IWGRP) == S\_IWGRP ? 'w': '-';

info.authority[6] = (stat\_file.st\_mode & S\_IXGRP) == S\_IXGRP ? 'x': '-';

info.authority[7] = (stat\_file.st\_mode & S\_IROTH) == S\_IROTH ? 'r': '-';

info.authority[8] = (stat\_file.st\_mode & S\_IWOTH) == S\_IWOTH ? 'w': '-';

info.authority[9] = (stat\_file.st\_mode & S\_IXOTH) == S\_IXOTH ? 'x': '-';

info.authority[10] = '\0';

info.f\_name = entry->d\_name;

info.f\_size = stat\_file.st\_size;

info.latest\_time = &(stat\_file.st\_mtime);

info.link\_num = stat\_file.st\_nlink;

info.u\_name = uid\_2uname(stat\_file.st\_uid);

info.g\_name = gid\_2gname(stat\_file.st\_gid);

print\_info(info);

}

closedir(dp);

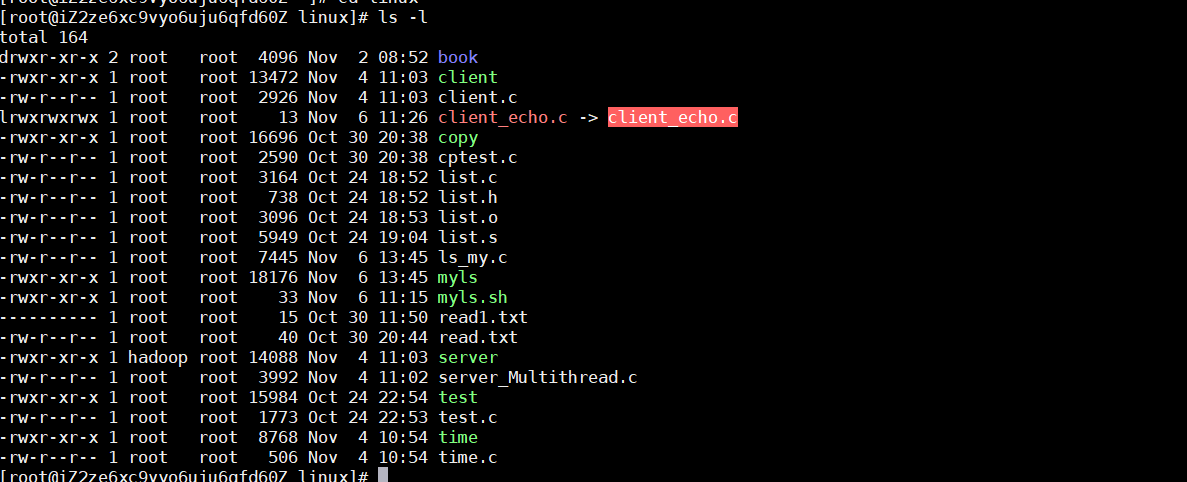
}

该函数首先对当前目录进行遍历，然后对于目录中的每一个文件进行相关的操作，将其信息存储到对应的file\_info之中。

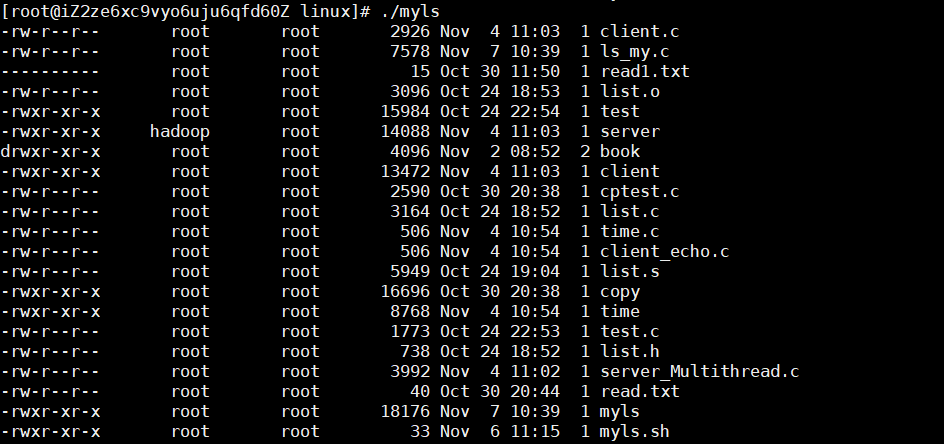
对于文件的类型利用宏来判定，并赋予相对应的值，对于文件的权限利用三位运算符来确定，嗯这里还使用了位运算。

2.功能测试

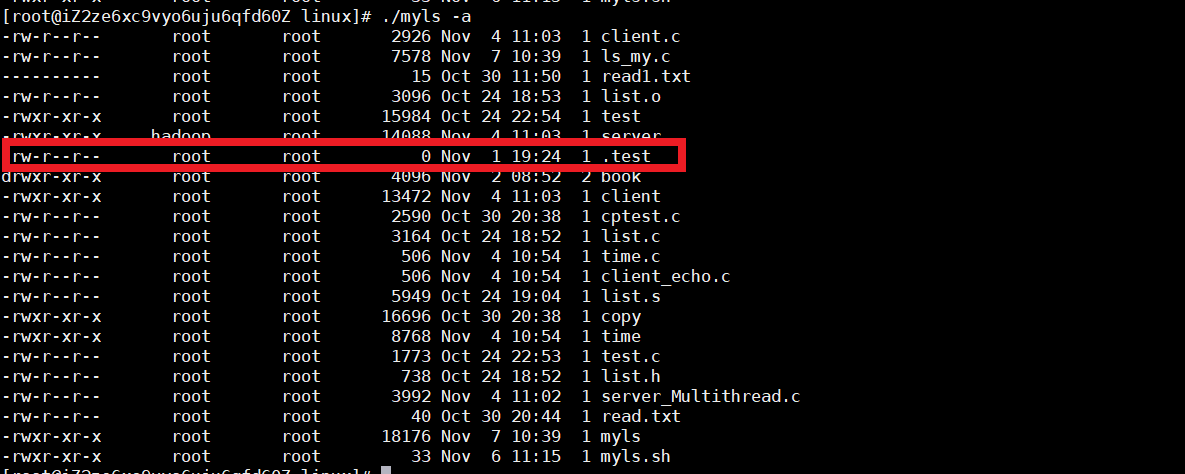
系统原ls展示



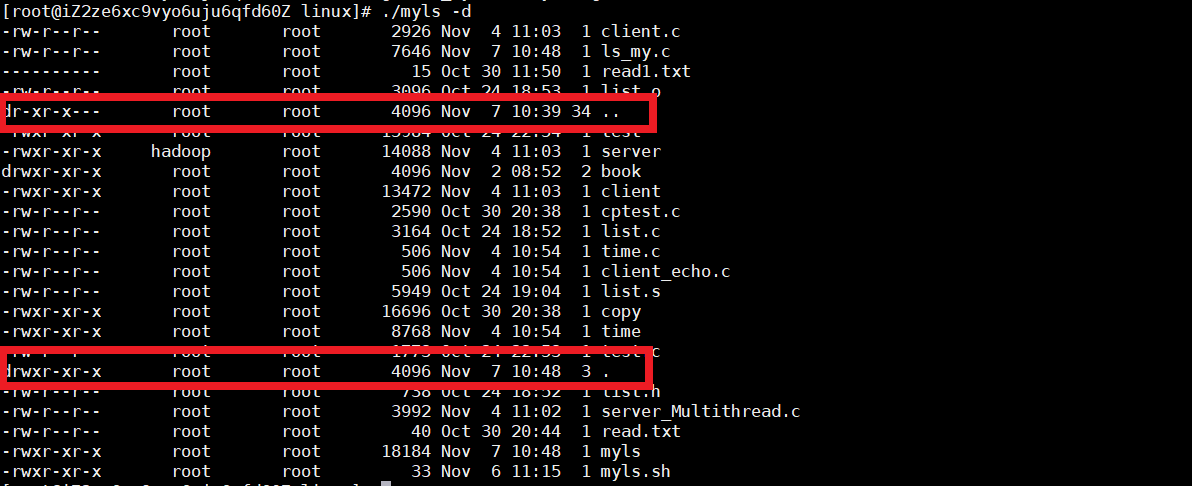
不加参数的使用：



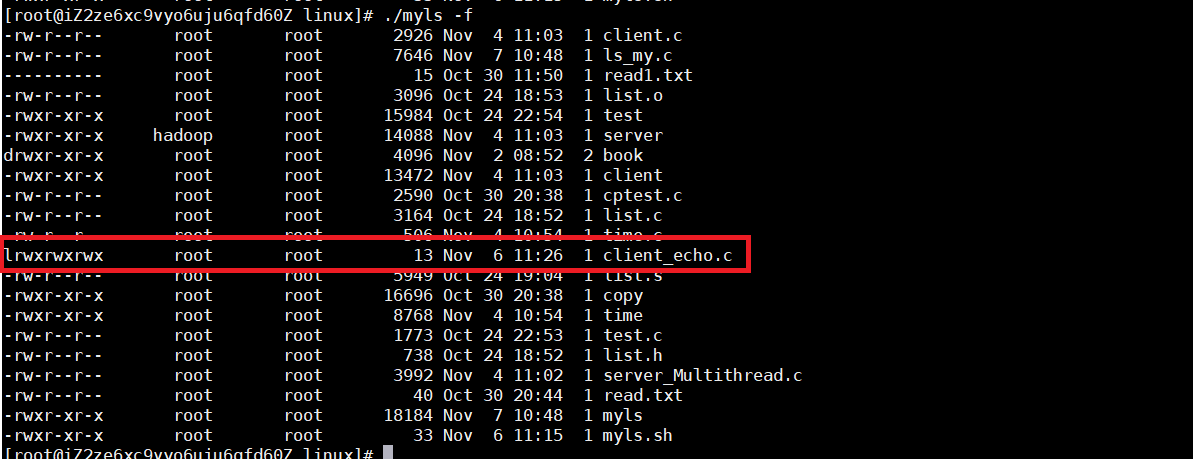
使用参数-a表示展示隐藏文件



展示.和..文件：



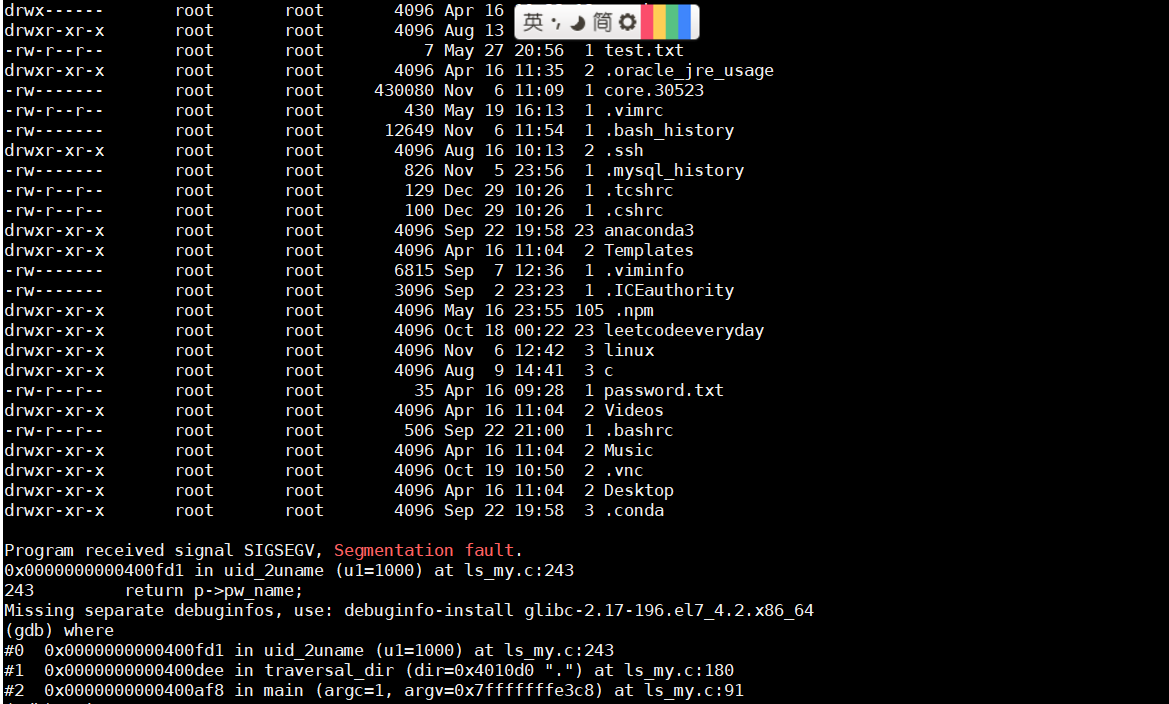
展示链接文件文件属性（默认展示的是链接文件的源文件）



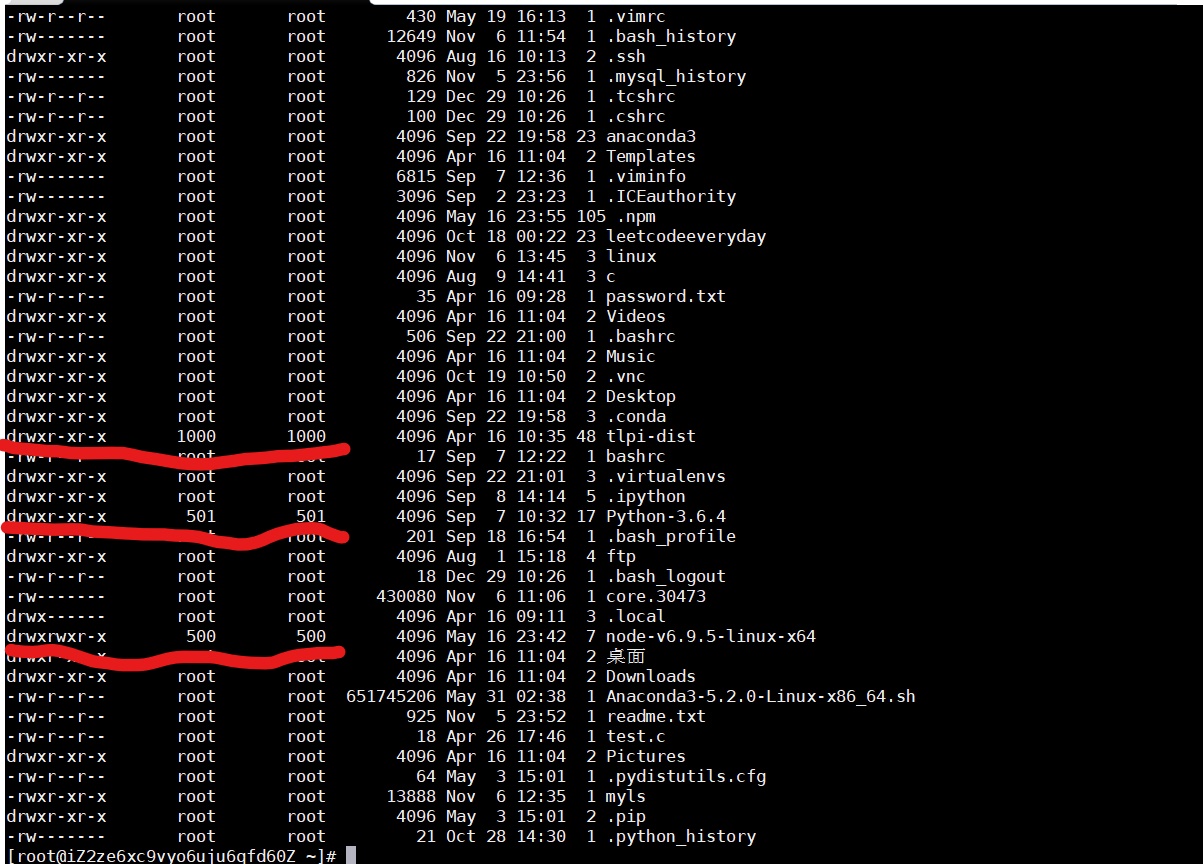
**九、总结及心得体会：**

1.遇到的问题以及解决方法

在第一次编译调试的时候，发现在一些文件夹执行myls命令会出现段错误，这时运用了GDB调试，发现是在将uid和gid转换为用户名和用户组名的时候出现了问题。



经过搜索资料，发现在linux 的文件系统，存在用户为NULL的情况，这时使用getpwuid和getgrgid函数时，返回的是一个NULL值，这时如果再执行return p->pw\_name;就会出现段错误，为了解决之个问题，在转换函数中新增了一个逻辑，即如果文件的所有者，或者文件组用户为空，即返回打印其uid和gid而不是打印用户名和用户组名，这样这个问题就迎刃而解。执行结果如下：



2.收获体会

GDB的调试功能还是很强大的，用好它是一门不小的学问。

经过此次试验，也深入了解了结构体struct direct和struct stat以及相关的操作，对linux下的文件系统也更加了解，收获颇丰。

**十、对本实验过程及方法、手段的改进建议：**

根据文件的不同类型打印出不同颜色的字体功能还没有试验，但是也找不出bug在哪里，但是就会出现颜色不对的情况，这点我很苦恼。

**报告评分：**

**指导教师签字：**

电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 201622020410

姓 名 刘高阳

（实验） 课程名称 UNIX/LINUX操作系统编程

理论教师 肖堃

实验教师 肖堃

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：刘高阳 学号：2016220204010 指导教师： 肖堃**

**实验地点：304 实验时间：2018/11/13**

**一、实验名称：**

**二、实验学时：2**

**三、实验目的：**

**四、实验原理：**

**五、实验内容：**

**六、实验器材（设备、元器件）：**

**七、实验步骤：**

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

**九、总结及心得体会：**

**十、对本实验过程及方法、手段的改进建议：**

**报告评分：**

**指导教师签字：**