

1. 实验内容

分析、设计与实现基于 Linux 内核的命令解释程序，主要包括系统环境变量的设置和初始化、系统命令提示符显示、命令辨别解析，典型内部命令处理

1.1 实验中实现的命令如下：

- (1). cat [选项] 文件名（查看文件内容）
- (2). touch [选项] 文件名（创建新文件）
- (3). cp 源文件 目标文件（文件拷贝）
- (4). rm 文件名（文件删除）
- (5). whoami （查看当前用户）
- (6). ln -s 目标文件夹 连接路径（创建软连接）
- (7). execvp 和 fork 函数执行外部命令

1.2 其他功能：

输出重定向：包括追加 (>>) 和覆盖 (>)

2. 实验环境

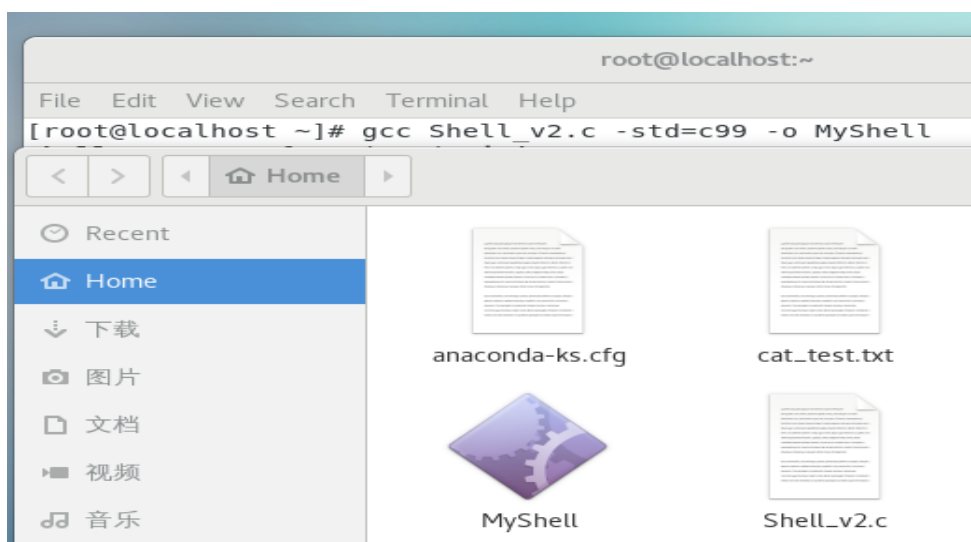
- 1.1 开发环境：vim7.4 编辑器
- 1.2 运行和测试环境：虚拟机（64 位）
 - 操作系统：Linux
 - 版本：CentOS 7

3. 源程序文件和可执行文件清单

实验中的源代码和可执行文件存储在和实验报告同级的 MyShell 目录下，分别命名为 MyShell.c 和 MyShell.out

4. 实验步骤

- (1). 了解 Linux 系统常见的命令，并确定实验中需要实现的指令集
- (2). 设计命令实现逻辑，系统命令提示符界面
- (3). 编制 C 源程序，在 Linux 虚拟机上编译 C 源程序，运行可执行文件（编译结果如下图）



可以看到生成了名为 MyShell 的可执行文件，之后在命令行输入“./MyShell”运行该程序，开始测试。

5. 技术难点及解决方案

(1). 命令解析：在实验中采用以字符串结构存储用户输入的命令，而在实际情况中用户的输入往往千奇百怪，这就给解析输入的命令带来了很大的困难。

解决方案：专门编写一个函数用于解析用户输入的命令，并返回对应的宏定义，根据命令解析函数的返回值来调用对应的命令实现函数。

命令解析函数说明如下：

函数原型：`int AnalysisCmd(char input[MAX_SIZE])`

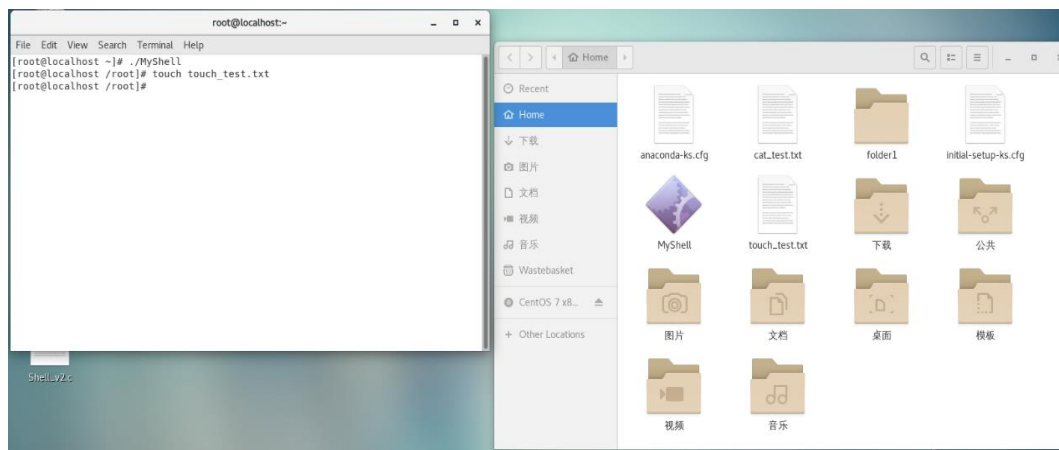
其中 input 为用户输入的命令，函数返回值为一个整数，对应各个命令的宏定义，详细宏定义如下：

```
#define CP 10
#define CAT 11
#define TOUCH 12
#define RM 13
#define WHOAMI 14
#define LN_S 15
#define EXTERNAL 16
```

(2). 系统调用：实验中 cp 命令采用了直接调用相应系统调用的方式实现，但对系统调用函数 syscall 的不熟悉，以及不同系统调用号的不熟悉给实现该功能造成了困难。

解决方案：通过查阅网上资料和教师 ppt，深入了解了系统调用的相关知识后，通过调用文件打开，文件读取，文件写入和文件关闭的系统调用，完成了该功能，部分核心代码如下图所示

执行 touch 命令创建.txt 文件之后



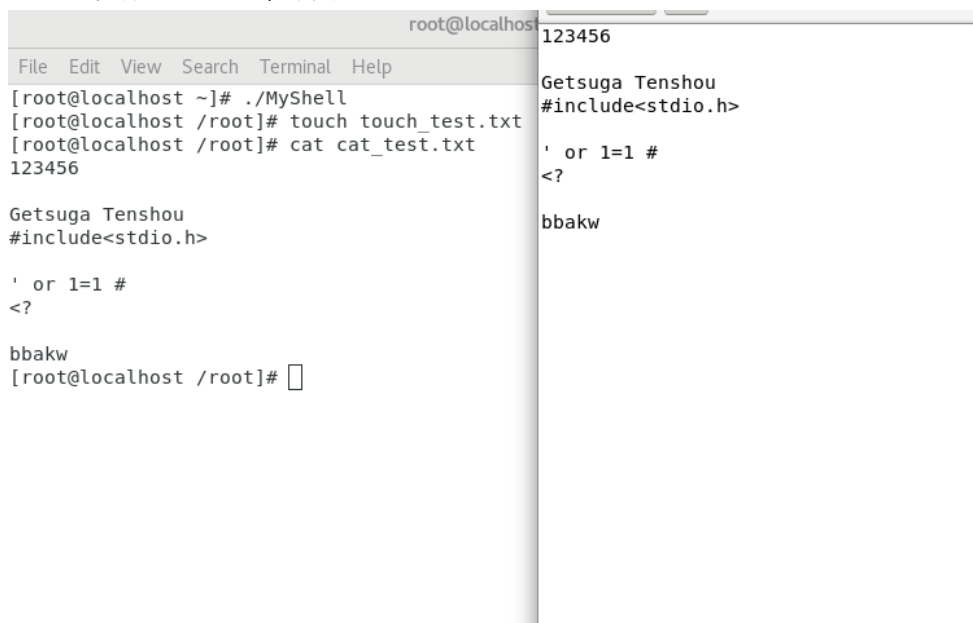
可以看到在虚拟机的主目录下多了一个名为“touch_test.txt”的文本文件，说明命令执行成功

(2)测试 cat 命令：

实验中实现的 cat 命令包括以下功能：普通模式；带有参数 -n，即显示行号的模式；以及输出重定向到文件的模式。

在 Linux CentOS 764 位虚拟机的测试结果如下图

无参数普通 cat 命令实现



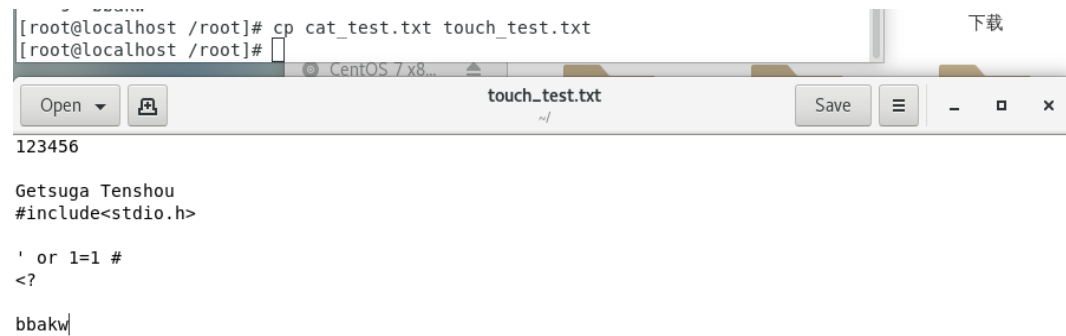
带 -n 参数（显示行号）cat 命令实现

```
[root@localhost /root]# cat -n cat_test.txt
1 123456
2
3 Getsuga Tenshou
4 #include<stdio.h>
5
6 ' or 1=1 #
7 <?
8
9 bbakw
[root@localhost /root]#
```

(3)测试 cp 命令

实验中实现了 cp 命令将源文件内容拷贝到指定文件中去的功能，测试结果如下

```
[root@localhost /root]# cp cat_test.txt touch_test.txt
[root@localhost /root]#
```



touch_test.txt

```
123456

Getsuga Tenshou
#include<stdio.h>

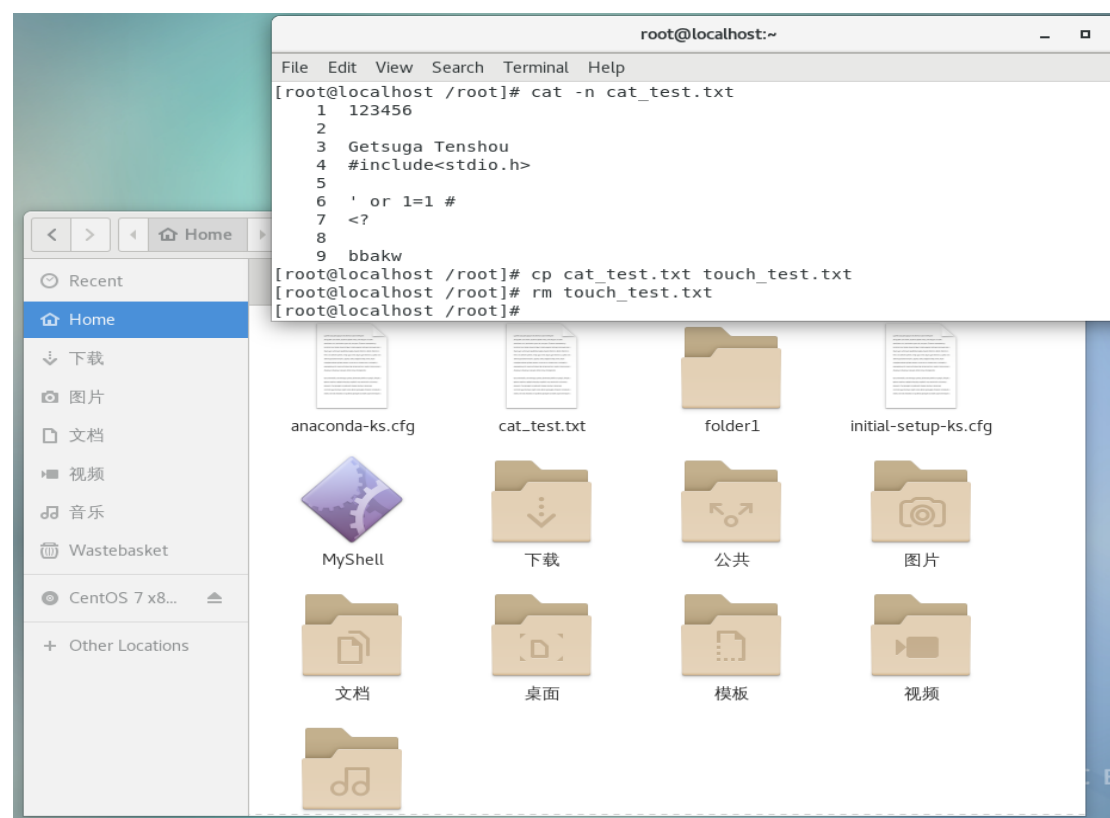
' or 1=1 #
<?

bbakw|
```

可以看到在上一步中创建的空文件 touch_test.txt 中已经被写入了 cat_test.txt 文件的内容，说明 cp 命令成功实现。

(4)测试 rm 命令

实验中测试 rm 命令使用的测试文件是(2)中创建的 touch_test.txt，运行结果如下图



可以看到在执行了 rm touch_test.txt 后，原本存储在主目录下的 touch_test.txt 成功被删除，说明该命令成功实现。

(5)测试 whoami 命令

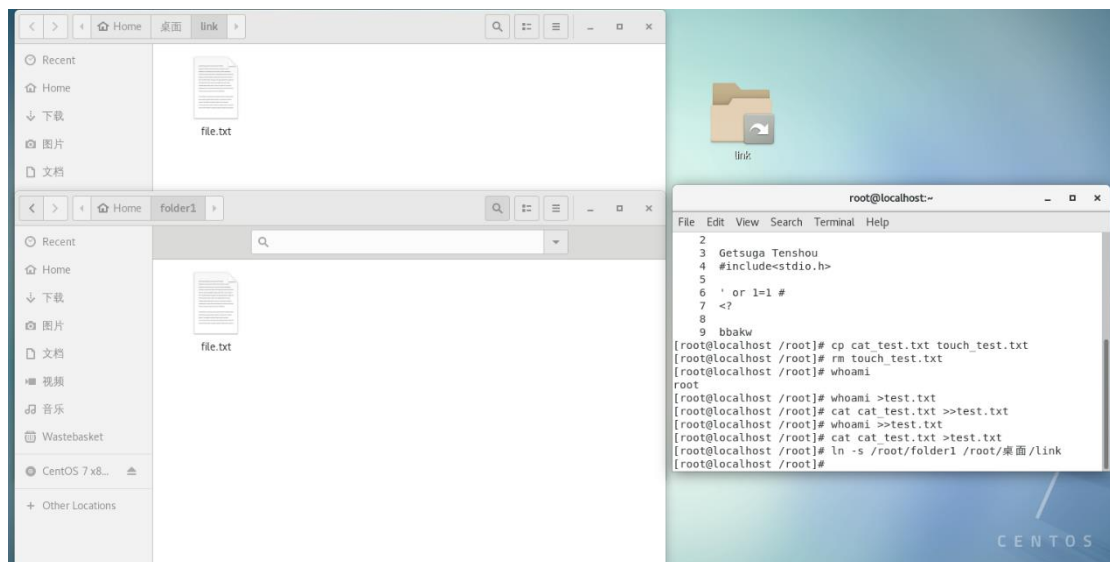
whoami 命令运行结果如下

```
root@localhost:~  
File Edit View Search Terminal Help  
[root@localhost /root]# cat cat_test.txt  
123456  
  
Getsuga Tenshou  
#include<stdio.h>  
  
' or 1=1 #  
<?  
  
bbakw  
[root@localhost /root]# cat -n cat_test.txt  
1 123456  
2  
3 Getsuga Tenshou  
4 #include<stdio.h>  
5  
6 ' or 1=1 #  
7 <?  
8  
9 bbakw  
[root@localhost /root]# cp cat_test.txt touch_test.txt  
[root@localhost /root]# rm touch_test.txt  
[root@localhost /root]# whoami  
root  
[root@localhost /root]#
```

可以看到在命令行窗口输出了当前用户名（即 root 用户），说明 whoami 命令成功实现。

(6)测试 ln -s 命令

ln -s 命令的功能是创建一个符号链接（类似于 windows 下的快捷方式），实验选取的连接目标文件夹是位于主目录下的 folder1，运行结果如下图




可以看到在执行了该命令后桌面上成功创建了一个名为 link 的软连接，打开后显示的正是和 folder1 一样的内容，说明该命令成功实现。

(7)测试外部命令执行

实验中执行外部命令使用的是 fork 函数创建子进程，并调用 execvp 函数从 \$PATH 环境变量中寻找可执行文件的方式，运行结果如下图

未将可执行文件放入 PATH 变量包含的文件夹之前：

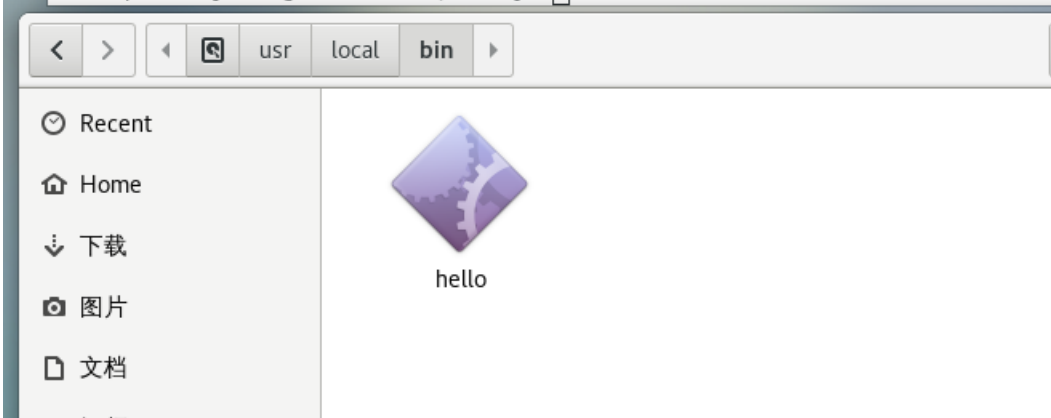
```
[root@localhost /root]# echo $PATH
/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/root/bin
[root@localhost /root]# hello
21251141辛少寒的bash:hello:command not found
[root@localhost /root]#
```



可以看到用于测试的可执行文件现在位于桌面（不包含在\$PATH 变量的范围内），因此程序提示未找到该命令，并重新进入命令输入模式，此出也有效验证了程序对于错误输入的处理能力。

将可执行文件 hello 拖入目录/usr/local/bin 目录下之后，再次尝试运行

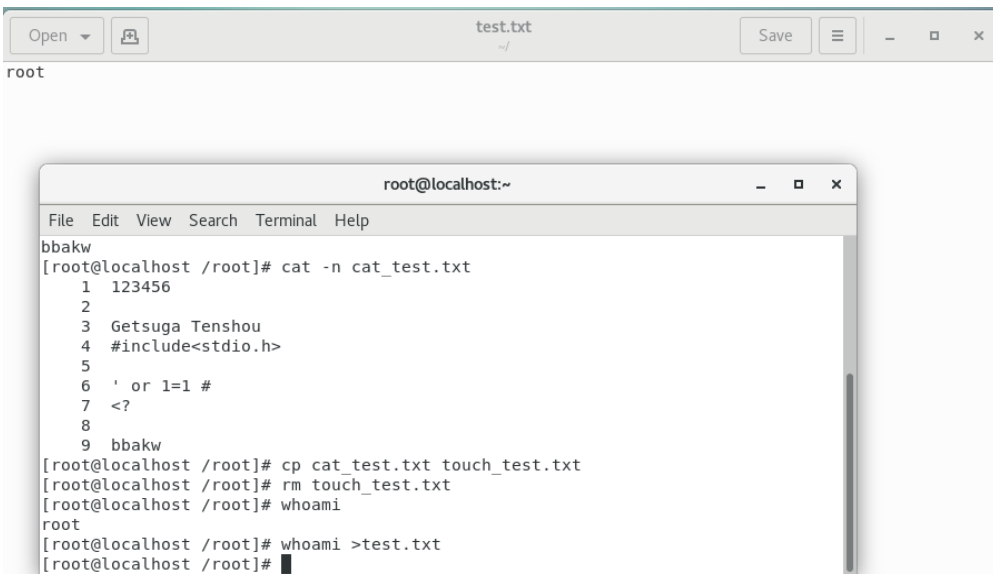
```
[root@localhost /root]# hello
hello,world[root@localhost /root]#
```



可以看到成功输出了"hello,world"字符串，说明执行外部命令的功能成功实现。

(8)输出重定向功能测试

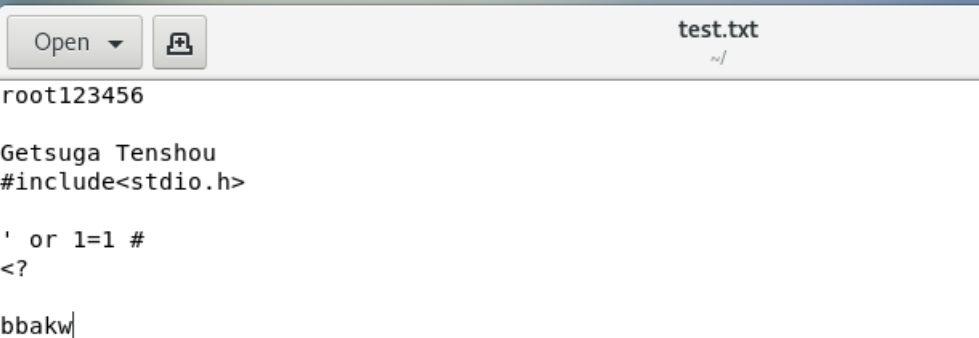
whoami 输出重定向（覆盖模式）



```
Open test.txt Save
root
root@localhost:~
File Edit View Search Terminal Help
bbakw
[root@localhost /root]# cat -n cat_test.txt
1 123456
2
3 Getsuga Tenshou
4 #include<stdio.h>
5
6 ' or 1=1 #
7 <?
8
9 bbakw
[root@localhost /root]# cp cat_test.txt touch_test.txt
[root@localhost /root]# rm touch_test.txt
[root@localhost /root]# whoami
root
[root@localhost /root]# whoami >test.txt
[root@localhost /root]#
```

为了对比明显，之后进行的是 cat 输出重定向（追加模式）的测试，如下图所示

```
[root@localhost /root]# whoami >test.txt
[root@localhost /root]# cat cat_test.txt >>test.txt
[root@localhost /root]#
```



```
root123456

Getsuga Tenshou
#include<stdio.h>

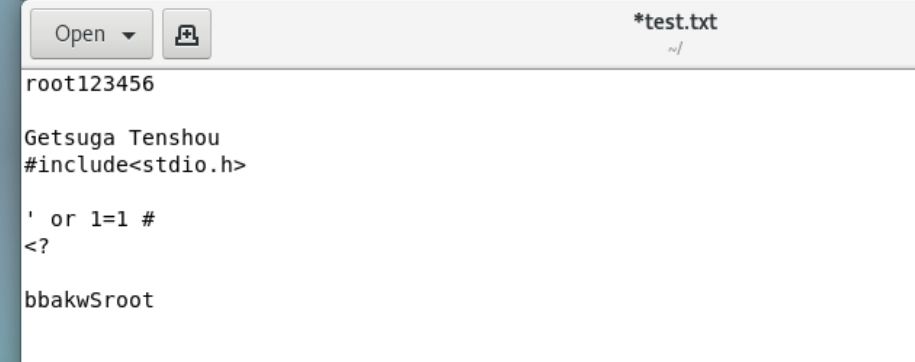
' or 1=1 #
<?

bbakw|
```

可以看到在前述图片的基础上，追加了一段 cat_test.txt 文件的内容。

之后进行的是 whoami 输出重定向（追加模式）的测试，如下图

```
[root@localhost /root]# whoami >test.txt
[root@localhost /root]# cat cat_test.txt >>test.txt
[root@localhost /root]# whoami >>test.txt
[root@localhost /root]#
```



```
root123456

Getsuga Tenshou
#include<stdio.h>

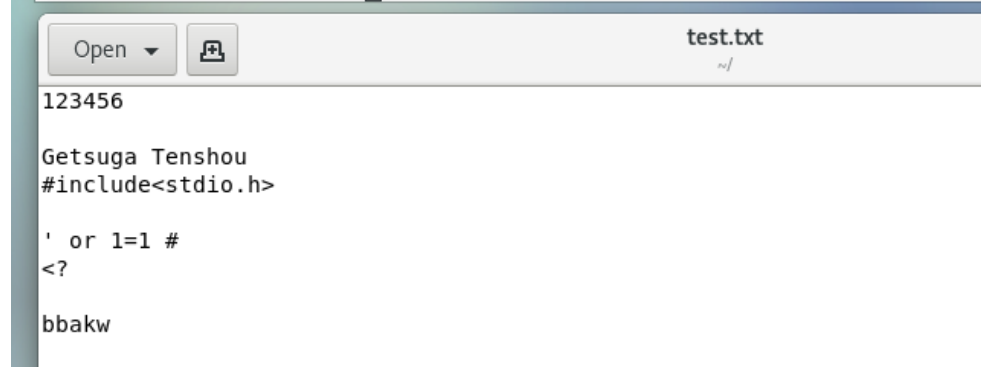
' or 1=1 #
<?

bbakwSroot
```

可以看到在上一张图片基础上，又在末尾多了一个"root"，说明追加写成功

之后进行的是 cat 输出重定向（覆盖模式）的测试，如下图


```
[root@localhost /root]# whoami >test.txt
[root@localhost /root]# cat cat_test.txt >>test.txt
[root@localhost /root]# whoami >>test.txt
[root@localhost /root]# cat cat_test.txt >test.txt
[root@localhost /root]#
```



可以看到执行了该命令后，文档内容变得和 cat_test.txt 一样，说明覆写成功。

8. 疑难解惑与经验教训

(1)疑难问题：

在最初测试时，想法是用自己写的命令解释器替换掉原有的 bash 程序，但替换完后虚拟机出现莫名卡死的问题，限于技术水平，一直无法找到解决办法，后来只能放弃替换原有 bash 程序的方案，改为在命令行直接执行自己的 Shell 程序进行验证。

(2)经验教训：

在最初设计 Shell 命令解释器时，没有全面考虑各种输入情况，导致程序在遇到一些刁难型的输入时出现异常，从中得到教训，在设计程序之初应该多注意命令解析的逻辑必须严密，考虑到多种情况。

9. 结论与体会

Linux 的命令解释器是一个复杂的大型程序，实现了对多种命令的解析和运行，是 Linux 系统的一种典型人机交互接口，本次实验中仅实现了 Linux 指令集中比较常见的 6 条指令，但从中也体会到了实现一个 Linux 命令解释器不但需要对操作系统的提供的系统调用有所了解，还需要考虑到命令的语法分析、词法分析、语义分析、命令历史管理、错误处理等多个方面，以及安全性和可扩展性等问题。总之，Linux 命令解释器设计实验是一个具有挑战性和实用性的实验，通过本次实验加深了我对操作系统和编程的理解，提高自己的系统设计能力。