目录

[摘要 2](#_Toc187265776)

[程序功能及设计思路 2](#_Toc187265777)

[一、程序功能分析 2](#_Toc187265778)

[二、程序功能设计 3](#_Toc187265779)

[三、系统整体流程 3](#_Toc187265780)

[关键技术的解决方法 4](#_Toc187265781)

[一、Shell 终端的设计 4](#_Toc187265782)

[二、自定义文件操作命令的实现 7](#_Toc187265783)

[工程编译和功能测试 9](#_Toc187265784)

[一、工程文件编译 9](#_Toc187265785)

[二、测试用例设计 10](#_Toc187265786)

[三、测试结果分析 12](#_Toc187265787)

[实习心得或良好建议 12](#_Toc187265788)

# 摘要

本实验旨在深入掌握 Unix 与 Linux 系统的核心知识与关键技术，通过设计并实现一个简易 Shell 终端以及一系列常见 Shell 文件操作命令，达成对系统原理、Shell 命令内部逻辑、文件系统结构及功能的深度理解。运用 C/C++ 语言，遵循模块化编程思想，完成了涵盖命令解析、执行、文件操作等多功能模块的开发，经测试验证了系统的有效性与稳定性，提升了自身在 Unix 与 Linux 系统下的编程实战能力。

# 程序功能及设计思路

## 一、程序功能分析

##### 1.功能需求

**（1）基本Shell终端功能**：循环读取用户输入并执行命令，直到用户输入exit命令退出。提示符显示当前用户名、主机名和工作目录。

**（2）命令解析与执行**：支持命令行解析，包括命令和参数。支持参数输入和错误处理，对无效命令给予提示。

**（3）文件操作命令实现**：实现myls、mycat、mycp、mycd命令，通过系统调用和基础库函数完成文件操作。

##### 2.性能需求

shell应具有良好的响应速度，能够快速解析和执行输入的命令。文件操作命令应能高效地处理文件，避免出现长时间无响应的情况。

##### 3.用户需求

用户应能方便地使用Shell终端，提示符信息清晰，易于理解。命令执行结果应准确，错误提示信息应明确，帮助用户快速定位问题。

## 二、程序功能设计

##### 1. Shell终端设计

**（1）基本框架**：使用循环结构实现命令的持续读取和执行。提示符通过字符串拼接实现，调用getcwd函数获取当前工作目录，解析环境变量获取用户名和主机名。

**（2）自定义命令解析与执行**：使用字符串处理函数，将输入字符串按空格分割，得到命令和参数的数组，并传递给相应的命令处理函数。

**（3）外部命令解析与执行**：使用fork创建子进程，在子进程中调用exec族函数执行命令，父进程使用waitpid等待子进程完成。

##### 2. 文件操作命令设计

**（1）myls命令**：使用opendir和readdir函数读取目录内容，stat函数获取文件详细信息，然后格式化输出文件类型、权限、大小等信息。

**（2）mycat命令**：使用open和read函数逐行读取文件内容，将读取到的内容输出到标准输出.

**（3）mycp命令**：使用rename系统调用实现文件移动或重命名操作，如果目标路径存在，根据需求覆盖或提示用户。

**（4）mycd命令**：使用chdir函数切换路径，并更改提示符输出的路径内容。

## 三、系统整体流程

系统运行流程如下：

图示

描述已自动生成

# 关键技术的解决方法

## 一、Shell 终端的设计

**1. 基本框架**：程序启动后利用循环持续读取用户输入，以[user@hostname current\_dir]$格式显示提示符。getUsername()函数通过获取当前进程用户的ID，再通过getpwuid()获取该ID的用户具体信息。

文本

描述已自动生成

getHostname()获取系统的主机名。

文本

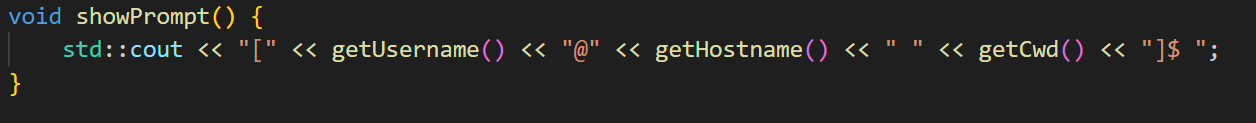
描述已自动生成

getCwd()函数获取到当前工作目录的路径后，存入cwd 的缓存区，用以输出。

文本

描述已自动生成

通过showPrompt()函数，将以上信息拼接成一个shell风格的提示符，输出到终端。

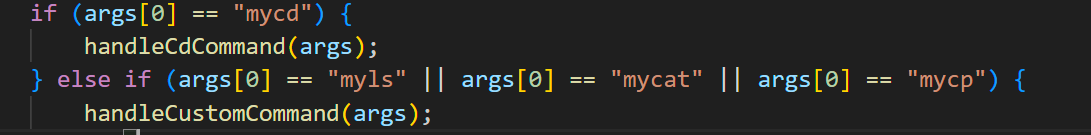


**2.命令解析与执行**：用户输入命令后，主程序判断是否命令为exit，若是，将退出程序，否则进入命令解析阶段。ParseCommand()函数，用于将用户输入的字符串解析成一个字符串向量，将命令和每一个参数分割开。在找到第一个空格之后，就将空格位置储存在found中，如果找不到则跳出循环。input.substr()可提取从pos到found之间的字符串参数，并将其储存在args中。

文本

描述已自动生成

在主函数中，判断args的内容，分别对输入命令进行处理，进入不同的处理函数。



HandleCustomCommand()函数具体判断自定义命令的种类，并为函数提供需要的参数。

文本

描述已自动生成

若非不是自定义命令，则进入系统命令处理部分：调用fork函数创建子进程，子进程中通过exec族函数执行命令，父进程则使用waitpid函数等待子进程任务完成。

文本

描述已自动生成

## 二、自定义文件操作命令的实现

1. **myls 命令**：列出指定目录下的文件和子目录。

文本

描述已自动生成

1. **mycat 命令**：依托open和read函数，设计逐行读取逻辑，将指定文件内容输出至终端，确保文件内容展示的准确性与完整性。

文本

描述已自动生成

1. **mycp 命令**：以只读模式 (O\_RDONLY) 打开源文件，转换为 C 字符串。循环读取源文件，将内容写入目标文件。

文本

描述已自动生成

1. **mycd命令**：调用chdir()函数，更改当前的工作目录。若参数为home且未设定其环境变量时，输出未设定环境变量的提醒。

文本

描述已自动生成

# 工程编译和功能测试

## 一、工程文件编译

##### 1.进入编译目录

图形用户界面

描述已自动生成

##### 2.显示工程文件结构

文本

描述已自动生成

##### 3.Makefile文件内容

文本

描述已自动生成

##### 4.进行编译

屏幕上有字

描述已自动生成

##### 5.编译后的文件目录

文本

描述已自动生成

## 二、测试用例设计

##### 1.运行二进制文件

终端成功显示由用户名，主机名，运行路径组成的提示符 [freak@racknerd-db840a /home/freak/shell]$。

蓝色的标志

描述已自动生成

##### 2.输入合法命令

（1）输入合法无参数命令myls进行测试，成功显示该路径下的工程文件

图形用户界面, 文本

描述已自动生成

（2）输入合法有参数命令mycp hello /home/freak/hello



在该目录下找到被移动的目标文件

文本

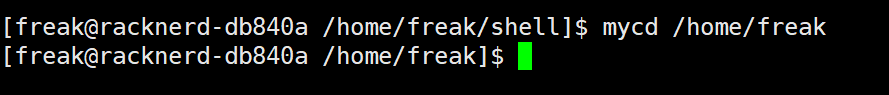
低可信度描述已自动生成

（3）输入合法有参数命令 mycat testfile，在终端成功输出测试文件的内容

文本

描述已自动生成

（4）输入合法有参数命令 mycd /home/freak，成功切换运行路径



（5）输入合法外部命令，成功执行



##### 3.输入非法命令 （1）输入非法命令sl ，提示未找到该命令。

图形用户界面

中度可信度描述已自动生成

（2）输入带错误参数的命令myls aaa，myls提示未找到该文件或目录。

手机屏幕截图

中度可信度描述已自动生成

（3）输入带错误参数的命令mycp /home/freak/shell/notexist /home/freak/shell，移动不存在的文件。提示原文件不存在。



## 三、测试结果分析

经测试用例验证，简易 Shell 终端在命令解析、执行及交互功能上表现稳定，错误提示及时准确；各项文件操作命令均能精准达成预期功能，对边界情况处理得当，如mycp在目标路径冲突时的智能决策，整体系统达到实验设计要求，为 Unix 与 Linux 系统实践应用奠定坚实基础。

# 实习心得或良好建议

通过本次实验，我认识到如何设计类shell程序的方法，加深了我在Linux系统的命令行操作，以及输入输出流，系统调用和键盘处理方式的理解。以及在实际编程中，如何利用操作系统提供的系统调用来实现这些命令功能。

通过自定义命令，我理解了如opendir()，read()，write()等低级Linux系统函数的使用，如myls命令中，opendir()和readdir()被用来遍历指定目录下的所有文件和子目录，先打开指定的目录并获取目录的曲柄，再从目录中逐个读取条目，直到目录末尾。

这些调用提供了直接的底层控制，接近操作系统的核心机制。让我能功能加精确地控制文件的读取和输出，提高了我对操作系统底层机制的理解。

在实现外部命令执行时，我使用 fork() 和 execvp() 来创建子进程并执行命令。这使得我理解了进程的创建与执行过程，以及如何通过系统调用管理父子进程间的关系。

虽然项目没有完全实现历史命令功能，但我意识到在实现这一功能时，可能需要维护一个命令历史的存储结构（如 std::vector<std::string> 或文件），并结合键盘输入事件处理。处理键盘事件时涉及到的控制字符（如 ^[[A 和 ^[[B）也让我对终端输入输出的工作方式有了更多的了解。

这个项目不仅加深了我对操作系统底层机制的理解，还让我更加熟悉了 C++ 在与操作系统直接交互时的强大功能。通过实现一个简单的命令行解释器，我学会了如何利用系统调用处理文件和目录的操作，如何创建和管理子进程，以及如何处理用户输入并执行命令。这个项目让我理解了编写可靠、灵活命令行工具的思路，也为我在更复杂的系统编程任务中打下了基础。