本科生实验报告

实验课程：操作系统原理实验

实验名称：Shell 的实现

专业名称：计算机科学与技术

学生姓名：数据删除

学生学号：数据删除

实验成绩：

报告时间：2025 年 6 月 12 日

1. 实验要求

虽然我们已经实现了操作系统的许多基本概念，但是我们的操作系统并不能接受和处理我们的命令。在 Linux 系统中，我们可以很方便地在 Terminal 下输入并执行我们的命令，如`cd`、`ls`等，这个用于解释和处理用户命令的 Terminal 被称为 Shell。

现在，同学们需要在本学期自己的实验基础上实现 Shell，同学们的 Shell 应该具有以下基本功能。

- 实现键盘输入并能够解析和处理用户输入的命令，如 cd，pwd，ls，cat，echo，rm 等。
- 在实现 Project 3 的基础上，或者是基于 ucore 或 xv6 的文件系统，实现从文件系统中加载程序到内存、然后解析 ELF 文件、最后创建进程运行这个程序。

在实现 Shell 后，同学们需要自行提供测例来测试你的 Shell。根据测试方法和输出结果来解释你的 Shell 的正确性。最后将结果截图并说说你是怎么做的。

2. 实验过程

文件系统是不可能写的，但是在 Windows 下模拟 shell 就有 85 分，那为什么不模拟呢。

模拟思路是每次从控制台输入一行字符串并分割成命令和一系列参数，根据命令调用不同的函数，工作流程如下：

- 初始化：设置当前路径为程序启动时的工作目录
- 主循环：
    - 显示提示符（当前目录名 + ">"）
    - 读取用户输入
    - 解析命令和参数
    - 分发到对应的命令处理函数
    - 执行命令并输出结果
- 退出：用户输入"exit"时结束程序

除此之外还要处理一系列意外情况，如非法命令、缺少参数、命令执行失败等。

3. 关键代码

解析命令：

```
vector<string> parseCommand(const string& input) {
    vector<string> tokens;
    istringstream iss(input);
    string token;
    while (iss >> token) {
        tokens.push_back(token);
    }
    return tokens;
}
```

几个实现的命令：

```
void CD(const vector<string>& args) {
    if (args.size() < 2) {
        cout << "Usage: cd <directory>" << endl;
        return;
    }

    fs::path newPath = currentPath / args[1];
    error_code ec;

    if (fs::exists(newPath, ec) && fs::is_directory(newPath, ec)) {
        currentPath = fs::canonical(newPath, ec);
        if (ec) {
            cout << "Error: " << ec.message() << endl;
        }
    }
    else {
        cout << "Error: Directory not found" << endl;
    }
}

void PWD() {
    cout << currentPath.string() << endl;
}

void LS(const vector<string>& args) {
    fs::path targetPath = currentPath;

    if (args.size() > 1) {
        targetPath = currentPath / args[1];
    }

    error_code ec;
```

```
    if (!fs::exists(targetPath, ec)) {
        cout << "Error: Path not found" << endl;
        return;
    }

    if (fs::is_directory(targetPath, ec)) {
        for (const auto& entry : fs::directory_iterator(targetPath, ec)) {
            if (entry.is_directory()) {
                cout << "[DIR]  ";
            }
            else {
                cout << "[FILE] ";
            }
            cout << entry.path().filename().string() << endl;
        }
    }
    else {
        cout << targetPath.filename().string() << endl;
    }
    if (ec) {
        cout << "Error: " << ec.message() << endl;
    }
}

void CAT(const vector<string>& args) {
    if (args.size() < 2) {
        cout << "Usage: cat <filename>" << endl;
        return;
    }

    fs::path filePath = currentPath / args[1];
    ifstream file(filePath);

    if (!file.is_open()) {
        cout << "Error: Cannot open file" << endl;
        return;
    }

    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }
    file.close();
}

void ECHO(const vector<string>& args) {
    for (size_t i = 1; i < args.size(); ++i) {
        if (i > 1) cout << " ";
```

```
        cout << args[i];
    }
    cout << endl;
}

void RM(const vector<string>& args) {
    if (args.size() < 2) {
        cout << "Usage: rm <filename>" << endl;
        return;
    }

    fs::path filePath = currentPath / args[1];

    if (!fs::exists(filePath)) {
        cout << "Error: File not found" << endl;
        return;
    }
    error_code ec;

    if (!fs::remove(filePath, ec)) {
        cout << "Error: " << ec.message() << endl;
    }
}
```

主循环：

```
void run() {
    string input;

    while (true) {
        cout << currentPath.filename().string() << "> ";
        getline(cin, input);

        if (input == "exit") {
            break;
        }

        if (input.empty()) {
            continue;
        }

        executeCommand(input);
    }
}

void executeCommand(const string& input) {
    auto args = parseCommand(input);
    if (args.empty()) return;
```

```
    const string& cmd = args[0];

    if (cmd == "cd") {
        CD(args);
    }
    else if (cmd == "pwd") {
        PWD();
    }
    else if (cmd == "ls") {
        LS(args);
    }
    else if (cmd == "cat") {
        CAT(args);
    }
    else if (cmd == "echo") {
        ECHO(args);
    }
    else if (cmd == "rm") {
        RM(args);
    }
    else {
        cout << "Unknown command: " << cmd << endl;
    }
}
```

4. 实验结果

ls 命令：可以正确显示当前目录下的子目录/文件

```
→ & "e:\Working\proj\mirai\build\mirai-vscode\shell.exe"
mirai> ls
[DIR]  .cache
[FILE] .clang-format
[FILE] .clangd
[FILE] .editorconfig
[DIR]  .git
[FILE] .gitignore
[DIR]  .vscode
[DIR]  .xmake
[DIR]  build
[DIR]  include
[FILE] mirai.code-workspace
[FILE] mirai_config.json
[FILE] README.md
[DIR]  src
[FILE] xmake.lua
mirai>
```

pwd、cd 命令：

```
mirai> pwd
E:\Working\proj\mirai
mirai> cd src
src> pwd
E:\Working\proj\mirai\src
src> cd C:/
> pwd
C:\
>
```

没有传入参数时能正常处理：

```
> cd
Usage: cd <directory>
>
```

cat 命令：

```
Temp> cat test.txt
this is a sample file
Temp>
```

遇到不存在的文件或无权限打开的文件能正常报错：

```
> cat 123
Error: Cannot open file
>
```

```
> cat swapfile.sys
Error: Cannot open file
>
```

rm 命令：

```
test> ls
[FILE] 1.txt
[FILE] 2.txt
[FILE] 3.txt
test> rm 2.txt
test> ls
[FILE] 1.txt
[FILE] 3.txt
test>
```

5. 总结