


Linux系统编程-作业2

- 学号：1120192092
- 班级：07111905
- 姓名：曾群鸿

功能介绍

本次作业 `shell` 起名为 `zsh`，来自我的名字的第一个字母（此 `zsh` 非彼 `zsh`）

✓ `ls` 命令

- 支持 `-a` 选项，打印隐藏文件（默认不打印文件）
- 仿bash，支持目录打印颜色为绿色，普通文件使用默认终端颜色

✓ `mv` 命令

- 支持移动一个文件
- 支持递归移动目录

- 仿bash，不添加 `-r` 选项

✓ `rm` 命令

- 支持删除普通文件
- 支持 `-r` 选项，递归删除目录文件

✓ `cp` 命令

- 支持复制普通文件
- 支持 `-r` 选项，递归复制目录文件

✓ `mkdir` 命令

✓ `pwd` 命令

✓ `cd` 命令

✓ `history` 命令

✓ `exit` 命令

使用方法

下载源码后，进入文件目录，文件结构如下：

```
├─ Makefile
├─ src
│   ├── my_cp.c
│   ├── my_cp.h
│   ├── my_rm.c
│   ├── my_rm.h
│   ├── utils.c
│   ├── utils.h
│   └─ zsh.c
```

运行 `makefile` 命令：

```
make
```

此时文件目录结构如下

```
.
├─ bin
│   └─ zsh
├─ Makefile
├─ obj
│   ├── my_cp.o
│   ├── my_rm.o
│   ├── utils.o
│   └─ zsh.o
└─ src
    ├── my_cp.c
    ├── my_cp.h
    ├── my_rm.c
    ├── my_rm.h
    ├── utils.c
    ├── utils.h
    └─ zsh.c
```

其中，`bin/zsh` 为可执行程序，使用如下命令执行：

```
./bin/zsh
```

设计方案

主体结构

主体的控制结构主要包括三个部分，首先读取用户输入，然后将用户输入通过空格切分成若干个 `token`，最后将用户输入 `token` 与系统命令匹配，选择对应的命令执行。

考虑到本次实现的命令较多，且为了后续扩展的方便性，我设计了函数指针数组的形式来维护 `shell` 中的命令，采用这一设计就可以简化程序逻辑，不需要使用大量的 `if-else` 语句来判断用户输入的命令是什么。

具体来说，通过如下的方式存储函数：

```
// 当前shell支持的命令名
char *shell_funcname[] = {
    (char*)"ls",
    (char*)"pwd",
    (char*)"cd",
    (char*)"mkdir",
    (char*)"rm",
    (char*)"cp",
    (char*)"mv",
```

```

        (char*)"history",
    };
    // 当前shell支持的命令对应的函数指针
    int (*shell_function[]) (char **) = {
        &my_ls,
        &my_pwd,
        &my_cd,
        &my_mkdir,
        &my_rm,
        &my_cp,
        &my_mv,
        &my_history,
    };

```

然后就可以通过如下的方式进行命令匹配和调用：

```

int zsh_func_num = sizeof(shell_funcname) /
sizeof(char*);
for (int i = 0; i < zsh_func_num; ++i)
{
    if (!strcmp(args[0], shell_funcname[i]))
    {
        // 查找到用户命令，通过函数指针调用对应命令执行
        int ret = (*shell_function[i])(args);
        break;
    }
}

```

通过这样的设计大量简化了我的代码，并且高度可扩展，想象一下目前 **Linux-shell** 支持的命令个数，如果使用 **if-else** 会让代码变得极为臃肿。

CP命令

为了实现仿 `linux` 的 `cp` 命令，需要根据带复制文件的类型（普通文件还是目录文件）做不同的处理，实现复制的基本原理为创建目标目录下的文件，然后打开待复制的文件，读取待复制文件的内容，写入目标文件。

这样的做法会导致复制过去的文件与源文件不统一，如创建时间，修改时间等，我的解决方法为通过系统调用 `stat` 拿到待复制文件的基本信息，然后通过 `utime` 改变目标文件的时间属性，设置成与待复制文件的时间属性一致。

因为需要支持单个文件的复制，也需要支持目录文件的复制，因此我写了两个函数（见 `src/my_cp.c`），一个用于复制单个文件，一个用于递归复制目录文件。

这里需要注意目录文件时间属性的设置节点，不能在递归之前设置其时间属性，因为之后需要递归到其子目录中修改文件，会导致时间属性被我们覆盖。需要等到回溯时，即其子文件都完成复制和时间属性设置后，才能对其进行时间属性的设置。

RM与MV命令

`rm` 命令同样需要支持删除普通文件和目录文件，对于普通文件，使用系统调用 `remove` 进行删除；对于目录文件，则需要递归删除其子文件后，才能删除目录文件。递归遍历的做法与 `cp` 命令类似，使用 `opendir` 和 `readdir` 获取当前目录下所有文件，然后逐一判断进行处理，这里不再赘述。（代码见 `src/my_rm.c`）

`mv` 命令比较特殊，`mv source target` 的操作可以看成是两个操作：`cp source target`，然后 `rm source`。所以没有必要再实现一次 `mv` 命令，直接调用我们已经实现好的 `cp` 和 `rm` 命令就可以实现。

其他命令

`ls` 命令使用 `opendir` 和 `readdir` 获取目录下文件，然后进行判断和输出。考虑到用户体验，模仿 `bash` 对目录文件输出绿色，而普通文件则使用终端默认颜色输出。

`pwd`，`cd`，`mkdir` 则分别使用 `getcwd`，`chdir`，`mkdir` 系统调用实现，相对来说比较简单，这里不再赘述。（代码请见 `src/zsh.c`）

参考

我在之前没有使用过 `Makefile`，通过本次作业，我学习和实践了 `Makefile` 的应用，收益匪浅，下面是 `makefile` 的资料：

- 李老师的课件
- Chase Lambert 等人编写的 [makefile 教程](#)
- 一个开源的 [makefile 工程实践](#)

