PROJECT TWO

Shell Simulator

191830187 叶家升

https://github.com/yegcjs/simu-Shell

01

任务需求

标准命令结构
需实现的命令
命令参数分析

02

模块划分

模块设计分析
模块划分图示
模块功能简述

03

数据结构

全局变量
模块变量

04

功能实现

模块核心功能
命令核心功能

任务需求

任务需求：标准命令结构

```
$ cmd ([options]) [arguments] … [options] [arguments]
$ example:
$ wc text1.txt –wc text2.txt text3.txt –l text4.txt
```

命令后跟 选项 操作对象　一个选项后可跟多个操作对象　多个选项可写在一起

任务需求：需实现的命令

复制文件：cp -r

比较文件：cmp

打印文件：cat

统计字数：wc -wcl

用户手册：man

批处理：sh

改变目录：cd

安装命令：install

任务需求：命令分析 cp

```
$ cp file1 file2 –r file3 folder1 folder2 target
```

- 在 –r 后可跟文件或文件夹，否则只能跟文件
- target：目标生成的文件或文件夹
  - target是文件: 前面只能有一个文件
  - target是文件夹：将前面的复制到文件夹下
  - target不存在：
    - 只有一个文件(夹)：生成相同内容且名为target 的文件(夹)
    - Else： 新建文件夹，并将前面复制到文件夹下

任务需求：命令分析 wc

```
$ wc (-wcl) file1 file2 –w file3  (-cl file4 …)
```

- 根据option各个arguments的
  - 字节数(c)
  - 单词数(w)
    - 分割符包括 空格、换行符
  - 行数(line)
    - 若行末没有换行符不算一行
- 输出顺序为行数、单词数、字节数（与option给出顺序无关）
- option不仅对option后的arguments有效，对整个命令行都有效
- 无option默认为三个都统计
- 若文件数大于1，则输出一个total，合计所有文件

任务需求：命令分析 man sh

```
$ man command1 command2 (…)
```

打开文档文件并输出

```
$ sh file1.sh file2.sh (…)
```

逐个读入sh文件，并逐行执行sh文件中的命令

任务需求：命令分析 cd install

```
$ cd new/directory/
```

将工作目录转移到给出的目录

```
$ install command1 command2 (…)
```

安装已经在特定目录下有同名.cpp,.h源文件或.o文件的命令
安装完成后重启myShell

模块划分

模块划分：模块设计分析

- Problem Analysis:
- 各个命令格式相同
- 操作对象一般为文件
- 各个命令都需要检查参数、报错

- Solution:
- 使用继承，设计基类Command
- 基类完成对命令行的分割
- 基类提供检查文件类型的方法
- 基类定义用于检查、初始化、报错的虚函数
- 派生类中完成具体操作

- Problem Analysis:
- 只有部分命令为自己实现
- 部分系统命令应当可以调用
- 因此需要对命令分发

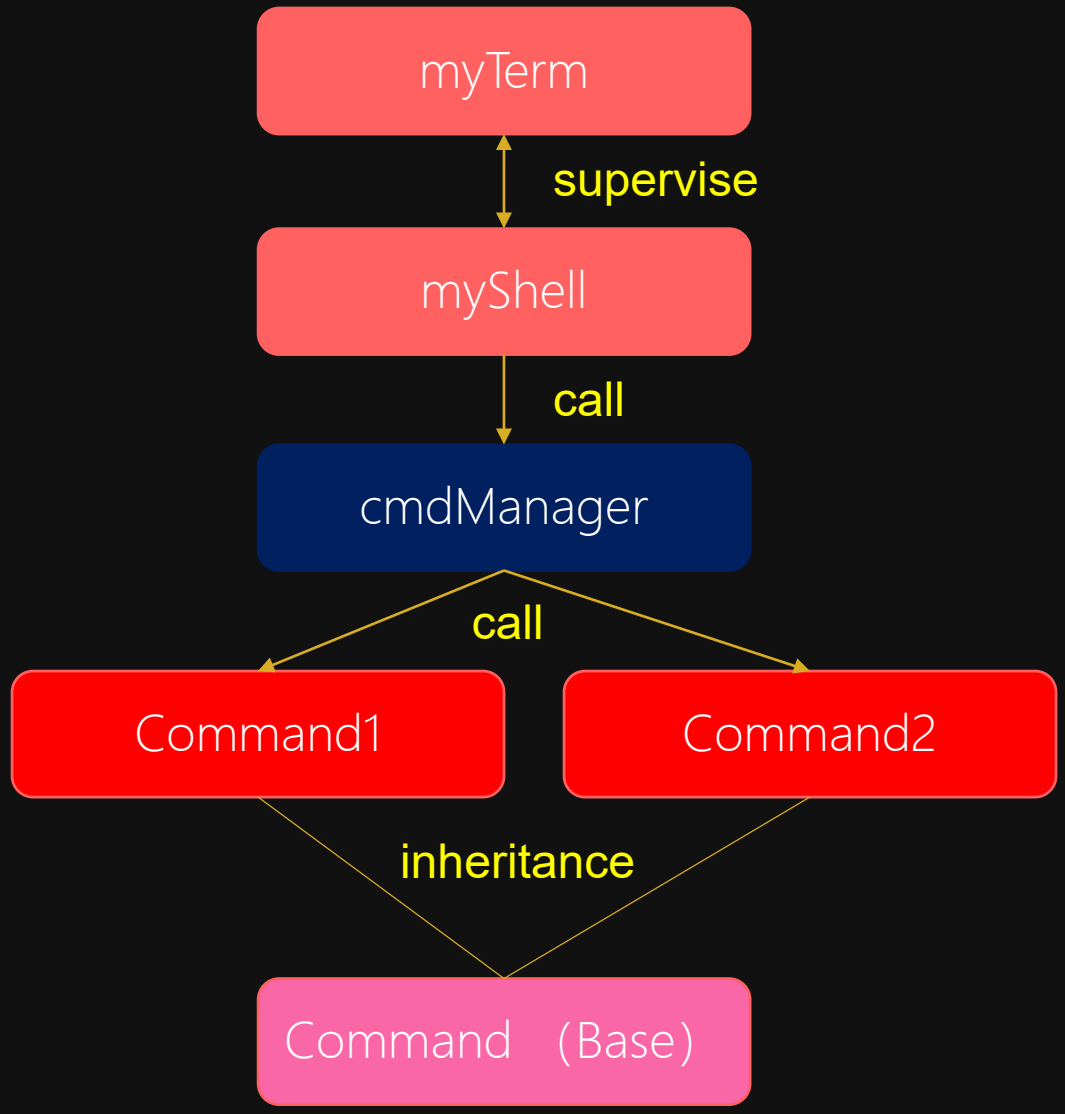
- Solution:
- 系统命令：调用system函数
- 设计一个模块来管理、调用自编命令
- 需要一个配置文件记录属于自编命令的命令

- Problem Analysis:
- 通过install命令加入新命令
- Install完成后内存中的shell仍是旧版
- 需要实现自更新

- Solution:
- 在Shell外面套一个监视器
- 根据Shell推出时返回值判断是否重启
- 需要一个log文件保留Shell退出现场

模块划分：模块划分图示

```
.
├── makefile
├── myShell
├── myTerm
│
├── bin
│   └── *.o
│
├── files
│   ├── config.txt
│   ├── log.txt
│   └── manuals
│       └── *.man
│
└── source
    ├── *.cpp
    └── *.h
```



模块划分：模块功能简述

myTerm

```
#include<cstdlib>
#include<unistd.h>
#include<pwd.h>
using namespace std;

int main(){
    char term_dir[10010];
    getcwd(term_dir,10009);
    while(1){
        int status = system("./myShell");
        if(status/256!=1)
            return 0;
        chdir(term_dir);
    }
    return 0;
}
```

启动myShell，监控myShell返回值

判断是否重启

模块划分：模块功能简述

myShell

```
void init();

void exec();
```

init:

1. 获取Shell根目录
2. 读取配置文件

exec:

1. 读取命令行
2. 提取首字符串为命令
3. 分发命令给System或cmdManager

模块划分：模块功能简述

cmdManager

```
class cmdManager{
public:
    void run(string cmd,string args);
};
```

run:
创建命令对象，并运行命令

模块划分：模块功能简述

Command(Base)

```
class Command{    //Base
protected:
    enum type{_file_,_dir_,_null_};
    type filetype(string file);
public:
    Command(string input);
    virtual ~Command();
    virtual int check_init()=0;
    virtual void print_error
            (char* error_type,string args)=0;
};
```

```
class myCommand:public Command{
    myCommand(string input);
    int check_init();
    void print_error(char *error_type,string args);
    ~cd();
};
```

Command:
拆分options和arguments，
建立对应关系

filetype:
辅助函数，判断文件类型

check_init:
检查option和arguments的合理性
合理参数转化为相应数据结构

print_errro:
报错

myCommand:
调用基类构造函数，调用
check_init()、执行命令

数据结构

数据结构：全局变量

用于解决：命令中需要借助调用其他命令（如更改目录、修改配置文件）来完成
初始化：在myShell中完成

```
set<string> myCommands;

cmdManager manager;
string shell_dir;
```

manager:
命令调用入口

myCommands:
交给manager处理的命令
利用set的去重特性避免重复
利用set的平衡树特性加查找

shell_dir:
myShell所在的根目录

数据结构：模块变量 cmdManager

```
void cmdManager::run(string cmd,string argv)
{
    if(cmd==string("cat"))
        cat tmp(argv);
    if(cmd==string("cd"))
        cd tmp(argv);
    //pass
    return;
}
```

tmp:
命令临时对象
（构造函数中完成命令操作）

数据结构：模块变量 Command

```
class Command{
protected:
    //<[option(s)],[args]>
    vector<string>option;
    vector<vector<string>> args;

    enum type{_file_,_dir_,_null_};
}
```

option:
args:
option[i] 与arguments[i]一一对应

type:
“文件”类型：普通文件、文件夹、不存在

功能实现

功能实现：模块核心功能 myShell

exec()

1. 输出 [username] @ [host]: current dir $
2. 读入一行命令
3. 切割出第一个字符串作为命令
4. 分发命令
    - exist in myCommand: manager.run()
    - not exist in myCommand: system()

功能实现：模块核心功能 cmdManager

run()

parameters: command, arguments

创建与command同名的命令类的对象

传入arguments为构造函数参数

功能实现：模块核心功能 Command

file_type()

headers: dirent.h cstdio

parameters: file (string)

按照以下顺序尝试

1. 尝试以directory打开
2. 尝试以file打开

（不能反过来）

```
Command::type Command::filetype(string file){
    FILE *f;
    DIR *dir;

    if(dir=opendir(file.c_str())){
        closedir(dir);
        return _dir_;
    }
    else if(f=fopen(file.c_str(),"r")){
        fclose(f);
        return _file_;
    }
    return _null_;
}
```

功能实现：命令核心功能 cp

Data Stucture:

```
private:
    string target;
    vector<string>  rel_file_list,file_list;
    vector<string> dir_list
```

target: 复制目标

file_list: 用于获取文件路径

rel_file_list:文件相对根目录的路径

dir_list:需创建的文件夹

Functions:

```
int check_init();
void open_dir(string path,string rel_path);
```

check_init: 参数中的文件(夹)加入数据结构

open_dir: 打开文件夹

功能实现：命令核心功能 cp

check_init:

遍历所有arguments：

- 文件：add to file_list, rel_file_list
- 文件夹：open_dir

open_dir: dfs

add current dir to dir_list

遍历目录下所有文件：

- 文件：add to file_list, rel_file_list
- 文件夹：open_dir

打开目录：

调用 DIR *opendir(char * directory)

获取目录下文件：

反复调用 dirent *readdir(DIR *)

```
DIR *cur_dir =  opendir(path.c_str());
while(item=readdir(cur_dir))
   string cur_file = string(item->d_name);
```

功能实现：命令核心功能 cd

```
if(check_init())
    chdir(new_dir.c_str());
```

headers: unistd.h pid.h

new_dir:

- 从参数中直接获得
- 如果是~开头
  1. 通过getpwid(getuid())->pw_name获得username
  2. 将~的开头换成/home/username/

功能实现：命令核心功能 install

```
class install:public Command{
private:
    set<string> newCommands;
    FILE *config,*makefile,*managercpp;
    //successfully 1 else 0
    int update_config();
    int update_makefile();
    int update_manager();
public:
    install(string input);
    ~install();
    //number of OK
    int check_init();
    void print_error
            (char *error_type,string argv);
};

extern set<string> myCommands;
extern string shell_dir;
```

headers: unistd.h pid.h

new_dir:

- 从参数中直接获得
- 如果是~开头
    1. 通过getpwid(getuid())->pw_
    2. 将~的开头换成/home/userna

功能实现：命令核心功能 install

Data Stucture:

```
private:
    set<string> newCommands;
    FILE *config,*makefile,*managercpp;
extern set<string> myCommands;
extern string shell_dir;
```

newCommands:

新增加的命令，利用set去重

config,makefile,managercpp:

分别对应三个更新时需要修改的文件

Functions:

```
private:
    //success 1 else 0
    int update_config();
    int update_makefile();
    int update_manager();
```

update_*:

修改

config.txt,Makfile,cmdManager.cpp

功能实现：命令核心功能 install

install:

1. check_init()
2. 生成log.txt保存当前运行目录
3. update_config
4. update_makefile
5. update_manager
6. 在shell_dir目录下system(“make”)
7. exit(1) 表示发生更新

myTerm接受返回值1后重启myShell

update_config:

将newCommands中的命令加入到config.txt中

update_managercpp:

1. #include所有命令头文件
2. 在run函数中，对每个命令CMD按如下格式

```
if(cmd==string(“CMD"))
    CMD tmp(argv);
```

update_makefile:

Command,cmdManager为不变，myShell增加新命令

对每个cmd:

```
bin/cmd.o:source/cmd.cpp source/cmd.h
    g++ -g -c source/cmd.cpp -o bin/cmd.o
```