2019.09.17
QU

Report for lab1: UNIX Shell with History Feature

Introduction:

The project has two parts, the first one is to create a child thread execute command from stdin stream. The second part is to create a program work like a shell with history feature, executing user's command, record commands, and handle some special command like "History", "!!", or "!N"(N is an int variable).

The first part of project is the base of the second part. So I upload part two as one single project.

I'll show my design in following context.

How to create a sub-thread to execute normal command ("ls", "dir"...):

I designed a command class, which could be constructed by receiving a string or char[] (only a construct function receive string is enough, on account of a char[] would be implicitly converted to a string), the command and args will be split, and stored in some high-level class instead of simple char[]. Then, when calling method command::execute(), execute() will allocate some memory space to store temporary variable which will be arguments of execvp().After fork(), the sub-thread would call execvp() to execute the origin command, and the parent thread (main thread) will block itself until sub-thread has died, or go on without caring about sub-thread, depending on the field neadtowatit.In the main thread, just before quitting execute(), execute() will also call doCleanJobs() to make sure the memory space of temporary variable will be freed.

```
class command{
private:
    std::string fullcommand;
    std::string simplecommand;
    std::vector<std::string> args;
    bool checkIfNeedToWait();
    bool needtowait = true;
    char** char_args;
    char* one_char_arg;
    char* char_command;
    command() {}
    void doCleanJobs();
public:
    command(std::string Fullcommand);
    ~command();
    void execute();
    std::string getcommand();
};
```

How to implement a history feature

Firstly, we will need a cycle list to store the lastest 10 command in memory. This could make some feature possible to be implement later. (Eg: print the lastest 10 command, execute the lastest command)

```
cycleList<command> list;
```

Then, we'll nead another class, to hold the list, and handle operating like reading history command from file, write lastest history command to file, deal with different format string, and so on... A huge class named historyList is designed as following.

```
class historyList {
private:
    int commandIndex;
    cycleList<command> list;
    std::string historyFileName = "history.txt";
    std::fstream historyFile;

    void RecordAndExecute(std::string Fullcommand);
    void Record(std::string Fullcommand);
    void Execute(command Command);

    std::string EncrypyCommandWithIndex(std::string CommandWithIndex);
    std::string MakeCommandWithIndex(std::string OriginCommand);

public:
    historyList();
    ~historyList();
    void ReadHistory();

    void Sethistoryfile(std::string NewFileName);
    void Recievecommand(std::string Fullcommand);
    void ExecuteLasteatCommand();
    void ExecuteNthCommand(int N);
    void PrintHistory();
};
```

The most important interface is Recievecommand(string), it receives a command as format of string, record it , writes it to the history file, construct a command class and inserts it into list.

There're two private methods, EncrypyCommandWithIndex(string) and MakeCommandWithIndex(string). They're used to format the string into two style.

```
//the origin linux command look likes this
dir -l /home
//the history command in file look likes this
12: dir -l /home
```

Got a shell with history feature, supporting our customize command

Finally, we need a class to handle every input line, decide to call a special interface of historyList, or transmit the input line to Recievecommand(string), or just quit our program. So, the class historyShell is designed.

```
class historyShell
{
    historyList HL;
public:
    historyShell();
    ~historyShell();
    void handleCommand();
};
```

The method handleCommand() will be called in main(), and take over stdin, waiting for a new input line.Once a input line comes, historyShell will check if the input is q/quit(quit the program), !!(call the ExecuteLasteatCommand()), or!N(call ExecuteNthCommand(int)). If the input is none of the special input, it just call Recievecommand(), make the historylist to handle it.

Exception handle?

The program is divided into three level by the three class. Every class make sure handle exception on its level, and high-level class called an interface from low-level dont care about whether there will be a exception, and low-level class's method being called dont care about anything wrong in arguements too.For example, historyList has a interface for execute our customize command !!.It'll handle the situation if there is no history command, won't transmit a wrong argument to low-level.

```
void historyList::ExecuteLasteatCommand()
{
    if (list.size() > 0)
    {
        auto cmd = list.getLastest();
        cmd.execute();
    }
    else
    {
        perror("No history command");
    }
}
```

How about Compiling and testing?

If you want to compile the project in Linux by yourself, just run "make" in path "qu_lab1". But an executable file is also released in path" qu_lab1\bin\x64\Debug". (The released version is built by Visual Studio automatically, built in very complex rules).A simple history.txt is also provided in path "qu_lab1", contain 30 history command.

In addition, the whole project is released on github, you could get it from https://github.com/devoteasecond/OS_course_lab.git