CS 302 Project 2

> 11812703 CHEN Qinan
>
> 11812704 WANG Shuo
>
> 11812820 CHEN Xinyu

Background and Description

1 SylixOS

SylixOS is an open source operating system designed and developed by Chinese. It is now very complete, and has a wide range of applications in the fields of national defense, aerospace, electric power, rail transportation, industrial automation, etc.

2 Importance of Shell

A shell provides user with an interface to the Unix system. It gathers input from the user and executes programs based on that input. When a program finishes executing, it displays that program's output.Shell is an environment in which users can run their commands, programs, and shell scripts.

3 Shell Types

There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

In UNIX, there are two major types of shells:

- Bourne Shell: the `$` character is the default prompt.
- C Shell: the `%` character is the default prompt.

The Bourne Shell has the following subcategories:

- Bourne Shell (sh)
- Korn Shell (ksh)
- Bourn Again Shell (bash)
- POSIX Shell (sh)

SylixOS already has a command line implementation. It has a rich and useful set of shell commands and simple shell scripting capabilities.

4 Goal

Our goal is to improve the command line mechanism and implement some features that are not yet available in the SylixOS Shell, and to enhance the shell command line programming capabilities.In this project, we will gain a deeper understanding of the OS human-computer interaction, master the internal interaction mechanism of the OS, and become familiar with the c programming language.

Implementation

1 Command line auto-completion

1.1 Significance

The command line Shell encapsulates many system calls and largely reduces the difficulty of direct interaction between programmer and system. It improves the programmer's efficiency in utilizing the system resources and is easy to customize. However, compared to graphical shells, command line shells require the user to be familiar with the commands to be used. As the number of Shell commands increases, it becomes increasingly difficult to familiarize oneself with all the commands, especially when the commands contain a large number of parameters. Therefore, if we can add auto-completion to the command-line Shell, we can greatly improve the usability of the command-line tool while retaining its original efficiency.

1.2 Expected goals

The implementation is divided into two parts.

The first part is the completion of the commands. For example, pressing the **TAB** key after we type **if** can automatically complete to **ifconfig**. If there are multiple completion results, we can print a list of possible commands on the second time pressing **Tab** key, prompting the user for more detailed information.

The second part is completion of the parameters. Completion of parameters is mainly for filenames. If there is a file named **abc.txt**, pressing the **TAB** key after we type **ab** can automatically complete to **abc.txt** only if we need parameters. It can also provide a list of possible input.

1.3 Technical route

For completion of the commands, we can store all the commands supported by the shell in a file and perform a fuzzy search after the user presses the Tab key to retrieve the commands that match our requirements.

For completion of the filenames, we can use **ls** command to get all the file names in the folder, and then fuzzy search after the user presses the Tab key to retrieve the files that match our requirements.

2 Command association help

2.1 Significance

Many commands in the shell have multiple arguments, a fraction of which are often known to people who don't use them often. Therefore, if we add associative help for command, users can easily know all the parameters of the command and their use.

2.2 Expected goals

After the user enters "**command+?**". Then press the Enter key to print the keywords or parameters that can be entered and their use.

For example if we enters "**cat ?**", it will print

```
-A, --show-all           equivalent to -vET
-b, --number-nonblank    number nonempty output lines, overrides -n
-e                       equivalent to -vE
-E, --show-ends          display $ at end of each line
-n, --number             number all output lines
-s, --squeeze-blank      suppress repeated empty output lines
-t                       equivalent to -vT
-T, --show-tabs          display TAB characters as ^I
-u                       (ignored)
-v, --show-nonprinting   use ^ and M- notation, except for LFD and TAB
--help     display this help and exit
--version  output version information and exit
```

2.3 Technical route

Write a parameter specification for each command. After we detect "**command+?**", we can then call a function to print the corresponding description.

3 Pipeline command operator

3.1 Significance

The pipeline command operator takes the standard output that the previous command was supposed to output to the screen as the standard input for the next command. This allows the CLI to do more without having to save the data to a file, greatly improving the CLI's functionality. In addition, using pipeline command operator in conjunction with **more** can make longer outputs more readable.

3.2 Expected goals

We want to be able to use the pipeline command operator multiple times in a single command. Like

```
command A | command B | command C
```

3.3 Technical route

Split the command according to the pipeline command operator we defined, then store these commands in an array. Execute each command in turn and redirect their input and output, printing the output of the last command on the command line.

4 Support for more shell commands, such as find, grep and sed

4.1 Significance

The **find** command is used to find files in a specified directory. The **find** command can be very handy when we have a lot of folders and files. We can also use fuzzy search to quickly find files that match the requirements

One of the most important functions when it comes to file handling in an operating system. The three most important file processing commands are **grep**, **sed** and **awk** which are known as the Three Musketeers.

The **grep** command is a powerful text search tool that searches for text using regular expressions and prints out the matching lines.

The **Sed** command is a stream editor that not only searches for text but also edits it. **Sed** processes one line at a time. When processing, the line currently being processed is stored in a temporary buffer, called the "pattern space" and the contents of the buffer are processed with the sed command. When the processing is complete, the contents of the buffer are sent to the screen. Then read the descent and execute the next loop. This repeats until the end of the document.

The function of **awk** is more powerful and complex. It can do almost everything that **grep** and sed can do, but it can also do style loading, stream control, mathematical operators, process control statements, and even built-in variables and functions. In fact, **awk** does have its own language. Considering that awk may be difficult to implement, we decided to implement only **grep** and **sed**.

4.2 Expected goals

We want to implement most of the functionality of **find**, **grep** and **sed**, and make the SylixOS shell have good text processing power.

4.3 Technical route

We intend to search the Linux source code for **find**, **grep** and **sed** and implement most of their functionality in SylixOS, imitated by the Linux source code implementation.

SylixOS provides an interface for adding custom shell commands. We can complete the addition of functionality through the following steps:

1. Add the `__tshellCostomCmdFunc` function to the `ttinyShellSysCmd.c`
2. Add code to the `__tshellSysCmdInit` function
3. Recompile the base project and bsp project.

Expected Goals

Our goals are twofold. One is to enhance the command line and the other is to enhance the shell.

For CLI, we want to implement keyword automatic completion function, associative help for command's parameters and pipeline command operator. This make we greatly improve the usability of the CLI tools.

For shell, we want to support for more shell commands, such as find, grep and sed. And if we can complete previous plan ahead of time, we also want to implement branching and looping statements, like if, elif, while and etc. Branching and looping statements allow our program to accomplish more. But we haven't figured out how to implement that yet.

Division of labor

CHEN Qinan: Command line auto-completion and find function

WANG Shuo: Command association help and grep function

CHEN Xinyu: Pipeline command operator and sed function