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 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.

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.

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

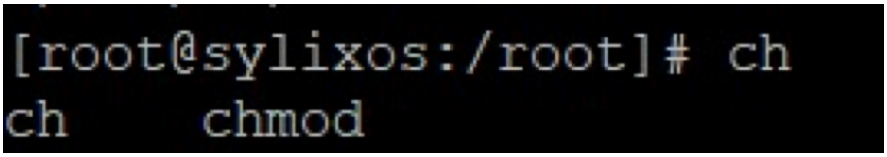
1 Command line auto-completion

1.1 Significance

Although CLI (command-line) type tools are efficient and easy to customize, CLI tools are visually harder to use than GUI tools, especially when we see the large number of arguments in the tool. Therefore, if we can add auto-completion to the CLI tools, we can greatly improve the usability of the tool while retaining the original efficiency of the command line tool.

1.2 Result analysis

Since SylixOS has already implemented filename and path completion, we mainly implemented **tab** key cauto-ompletion for command. When there is only one command that can be matched by the prefix entered by the user, the command will be completed directly, and when there is more than one command matched, the list of all commands will be printed for the user to select.



1.3 Implementation

The original way SylixOS stored keywords was a hashmap, which was difficult to traverse, so I added an array to store the registered commands. When the method that registered the keyword is called, it stores the keyword in the array. When SylixOS detects that the user has pressed the tab key and the first parameter has not been entered, it triggers the command auto-completion, which iterates through the array of stored keywords to see if the content entered on the command line is prefixed with the registered command.

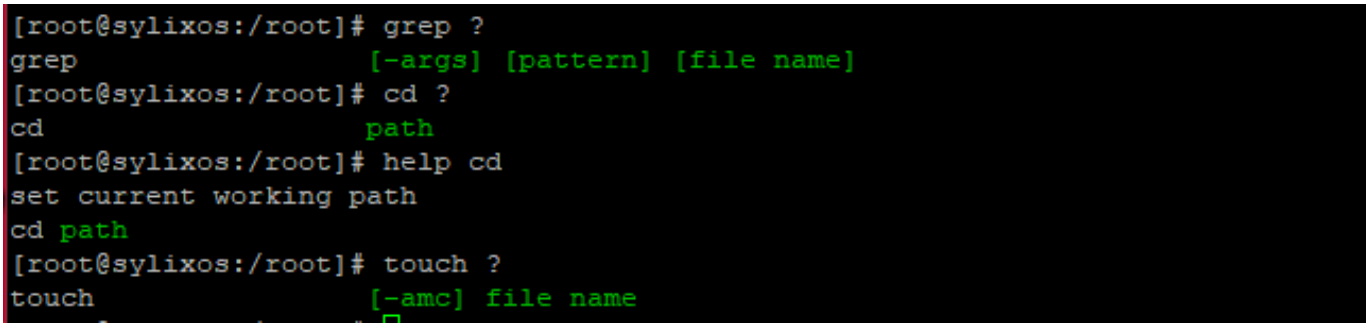
2 Command association help

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 Result analysis

After the user enters "**command+?**". Then press the Enter key to print the parameters that can be used.



2.3 Implementation

`__tshellWrapper` function in `ttinyShellLib.c` is used to handle a command from keyboard input. And `__helpPrintKeyword` function in `ttinyShellSysCmd.c` is used to print the help information for a keyword. After we detect "**command+?**", we can then call `__helpPrintKeyword` to print the corresponding description.

3 Pipeline command operator

3.1 Significance

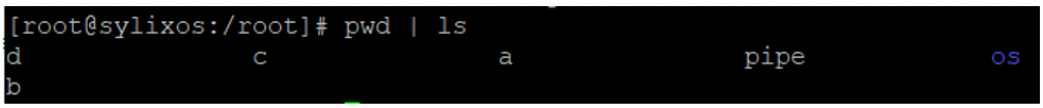
The pipeline command operator takes the standard output that the previous command was supposed to output to the screen as the standard input for the next command. This allows the CLI to do more without having to save the data to a file, greatly improving the CLI's functionality. In addition, using pipeline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

3.2 Result analysis

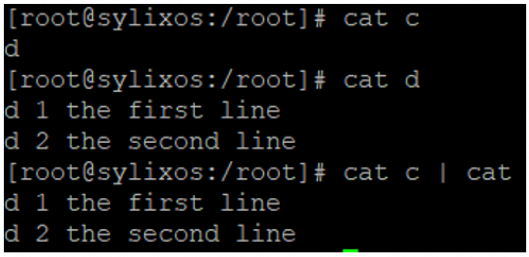
```
command A | command B
```

Pipe：

pwd | ls



cat c | cat



3.3 Not achieved and difficulties

Use pipeline multiple times: the difficulty is use pipeline multiple times, I need to divide the complete command into multiple pieces then deal with them one by one, which is much more complex then use pipeline only one time. I can not reach the goal by the design of my realization of one pipeline.

4 Find

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

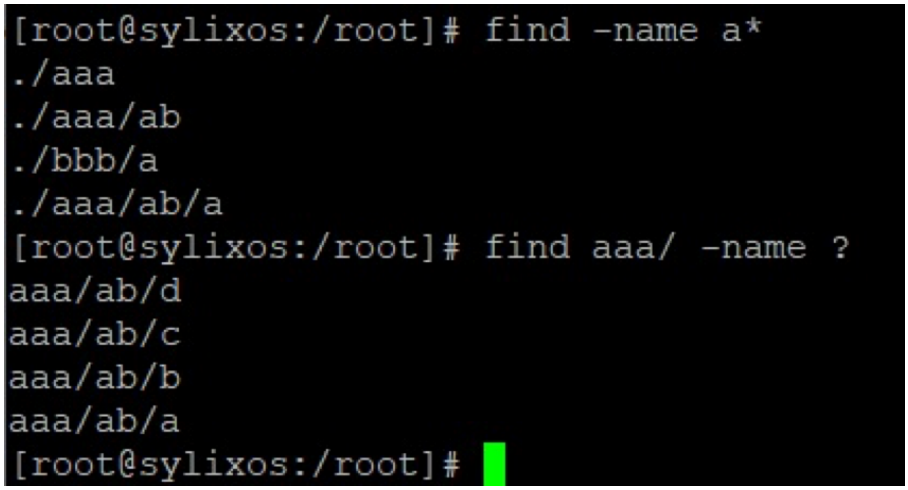
4.2 Result analysis

Implement part of the functionality of find to search for files with multiple parameters and print the searched files on the command line.

A total of 8 parameters are currently supported:

- **-name** Search files by filename, support * and ? both wildcard characters
- **-type** Search files by filetype
- **-links** Search files by the number of hard links
- **-size** Search files by filesize, support exact or range of file size search and input in b, kb, mb
- **-empty** Search empty file or folders
- **-perm** Search files by file permissions
- **-mtime** Search files by the last modification time, Unit in days
- **-mmin** Search files by the last modification time, Unit in minutes

All parameters can be used in combination instead of supporting one parameter at a time alone. Searching for files is done recursively instead of just searching for a folder.



```
[root@sylixos:/root]# find aaa/ -type d
aaa/ba
aaa/ab
[root@sylixos:/root]# find aaa/ -type f
aaa/ab/d
aaa/ab/c
aaa/ab/b
aaa/ab/a
```

```
[root@sylixos:/root]# find bbb/ -links 1
bbb/b
bbb/a
bbb/c
[root@sylixos:/root]# find bbb/ -links +1
[root@sylixos:/root]#
```

```
[root@sylixos:/root]# find -size +1k
./aaa
./bbb
./aaa/ba
./aaa/ab
./bbb/c
[root@sylixos:/root]#
```

```
[root@sylixos:/root]# find -empty
./aaa/ba
./bbb/a
./bbb/c
./aaa/ab/d
./aaa/ab/c
./aaa/ab/b
./aaa/ab/a
[root@sylixos:/root]#
```

```
[root@sylixos:/root]# find bbb/ -mtime -1
bbb/b
bbb/c
[root@sylixos:/root]# find bbb/ -mtime +1
bbb/a
[root@sylixos:/root]# find bbb/ -mmin -300
bbb/b
[root@sylixos:/root]#
```

4.3 Difficulties

1. When recursive folders are used, their full paths must be saved, otherwise they cannot be opened properly
2. The parameters entered by the user and the way the system saves do not match and cannot be compared directly, we need to convert

4.4 Implementation

- First iterate through the folder, and output it if it matches the parameters.
- Second if a folder is encountered during the iteration, it is added to a queue.
- When the current folder is finished, a new folder is taken out of the queue and the process is repeated until there are no new folders in the queue.

5 Grep

5.1 Significance

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.

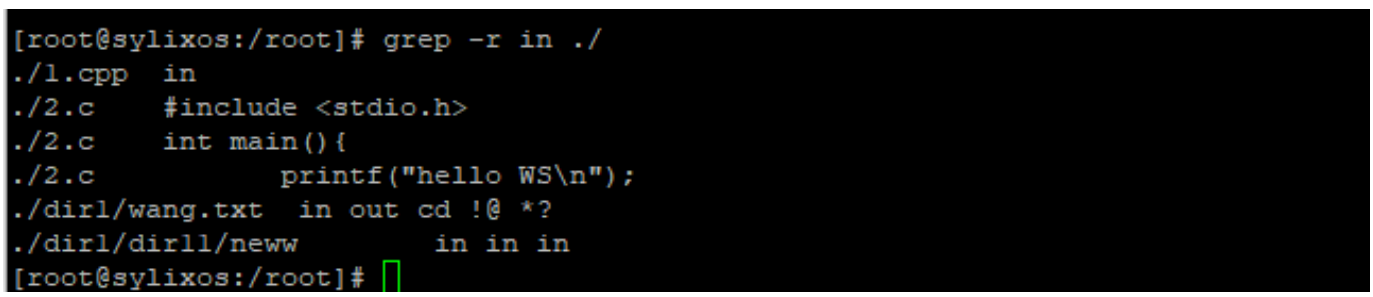
The **grep** command is a powerful text search tool that searches for text using regular expressions and prints out the matching lines.

5.2 Result analysis

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

1. Support wildcard characters match.
2. Use KMP algorithm to improve matching performance.
3. Support recursive search.
4. Support searching for a specific directory.
5. Support some regular expression operators.

```
grep [-r -R] [pattern] [filename / dirname]
```



```
[root@sylixos:/root]# grep -r in ./
./1.cpp  in
./2.c    #include <stdio.h>
./2.c    int main(){
./2.c            printf("hello WS\n");
./dirl/wang.txt  in out cd !@ *?
./dirl/dirll/neww        in in in
[root@sylixos:/root]# grep -r in /root/
/root/1.cpp      in
/root/2.c        #include <stdio.h>
/root/2.c        int main(){
/root/2.c                printf("hello WS\n");
/root/dirl/wang.txt      in out cd !@ *?
/root/dirl/dirll/neww    in in in
[root@sylixos:/root]#
```

```
[root@sylixos:/root]# grep -r -R in.* ./
./2.c    int main(){
./1.cpp  in
./dirl/wang.txt  in out cd !@ *?
./dirl/dirll/neww        in in in
[root@sylixos:/root]#
```

5.3 Difficulties

1. How to read far larger file.
2. Memory management.
3. Output alignment.

5.4 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.

6 Sed

6.1 Significance

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.

6.2 Result analysis

sed function:

```
sed file
sed [address]s/pattern/replacement/flags file
sed [address]d file
sed [address]p file
sed [address]a (or i) \new text file
sed [address]c\new text file
sed [address]y/inchars/outchars/ file
sed [address]w filename file
sed [address]r filename file
sed [address]q file
```

1.sed :

```
[root@sylixos:/root]# cat a
first line
second line
third line
4th line
```

```
[root@sylixos:/root]# sed a
first line
second line
third line
4th line
```

2. sed s replace
Format :
sed [address]s/pattern/replacement/flags file

```
[root@sylixos:/root]# sed 's/i/1/' a
f1rst line
second line
third line
4th line
```

```
[root@sylixos:/root]# sed 's/i/1/g' a
f1rst l1ne
second l1ne
th1rd l1ne
4th l1ne
```

```
[root@sylixos:/root]# sed 's/i/1/2' a
first l1ne
second line
third l1ne
4th line
```

```
[root@sylixos:/root]# sed 's/f/1/p' a
1irst line
```

3. sed d
delete lines that satisfy params
[address]d

```
[root@sylixos:/root]# sed 'd' a
[root@sylixos:/root]# sed '2,3d' a
first line
4th line
[root@sylixos:/root]# sed '2d' a
first line
third line
4th line
[root@sylixos:/root]# sed '3,"$"d' a
first line
second line
```

4. sed p
Search for lines that meet params and print them

[address]p

```
[root@sylixos:/root]# sed '2,3p' a
first line
second line
second line
third line
third line
4th line
[root@sylixos:/root]# sed '2p' a
first line
second line
second line
third line
4th line
```

5. sed a i
a append a line to specific line
i insert a line before specific line
[address]a (i) \ text

```
[root@sylixos:/root]# sed '3a\addedline' a
first line
second line
third line
addedline
4th line
[root@sylixos:/root]# sed '3i\addedline' a
first line
second line
addedline
third line
4th line
```

6. sed c
replace all the content with text
[address]c\text

```
[root@sylixos:/root]# sed '3c\addedline' a
first line
second line
addedline
4th line
```

6.3 Not achieved and difficulties

1. The options of sed function: I think the options of the sed function is not difficult, but it takes time. And I spend time on solving pipeline, then I have no time to complete the sed function.

Summary

In the process of completing the project, we had a preliminary experience of developing an operating system.

Developing an operating system requires a much better overall design than developing an application. Specifically, we observe that SylixOS defines its own data types, data structures, and some basic functions. These customizations are generally stored in the lib folder.

I noticed that SylixOS has two ways of implementing reading files. One is to read the entire file into a buffer, like the `cmp` command. The other is to read the contents of the file into the buffer in fixed-length segments, such as the `cat` command. I think the latter method is safer. Because the maximum file supported by the file system may be larger than the memory available to the operating system, reading the entire file into the buffer may lead to a memory shortage, which in turn may cause the system to crash.

In addition, using c to program requires special attention to memory management, especially for programs that need to run continuously like operating systems. Some memory management mechanisms and functions have been implemented in SylixOS. When the function of the built-in methods of the C language is the same as that of the corresponding methods of the SylixOS, the methods provided by the SylixOS should be used in preference.

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