

---

# 61-PROJ2DESIGNDOC

## SHELL-ENHANCEMENT

---

THIS IS A DESIGN DOCUMENT TO BE SUBMITTED FOR CS302 @ SUSTECH

**Group Member\***  
11610613 Zheng Shuxin  
11711421 Jiang Yifan  
11712819 Yu Zhiyang

May 30, 2021

### ABSTRACT

This document was for the final project of CS302, 2021 spring. In this doc, the details about our project 2 development was introduced. This task was developed in RealEvo kit including IDE and Simulator, which was offered by the Yihui tech.

## 1 Environment Setting

First about the developing environment: RealEvo-IDE, and RealEvo-Simulator. We can get all the source code of the SylixOS by new a base project in the RealEvo-IDE, and then modify it. After compiling, we then need to build a Bsp project base on the former base project, and compile it directly after build. When finished, we will get a .bin file under the "Debug" or "Release" directory(note if you use "x86" structure, .elf instead). The final file structure was like fig1-1.

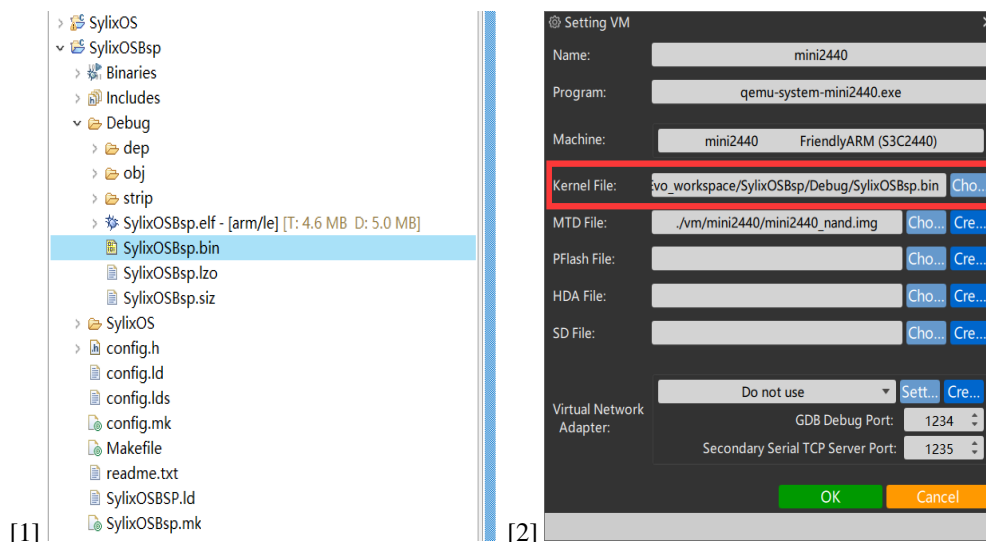


Figure 1: software setting

After build the binary file, we put the file into the simulator as the kernal like fig1-2, than run it!

---

\*the source code is now open sourced on github : [https://github.com/lannnnn/OS2021\\_project2](https://github.com/lannnnn/OS2021_project2)

## 2 Implementation

Our main implementation mainly includes the following three part:

- The auto-completion of command line keywords
- The associative help for command line keywords and arguments
- Support find, head, sed utility commands

We will introduce the developing details in this section.

### 2.1 associative help for command line keywords and arguments

If *tab* is detected, shell will first search for a completely same keywords and display the usage of the command if found. A function *tshellCmdMatchFull* is added in */libsylixos/SylixOS/shell/ttinyShell/ttinyShellLib.c*. It is called by function *tshellCharTab* developed in */libsylixos/SylixOS/shell/ttinyShell/ttinyShellReadline.c/*. Function *tshellCmdMatchFull* calls for function *\_\_tshellKeywordFind* which was developed originally used by command help. Then if found a match keyword, the function *\_\_tshellKeywordFind* will give us a pointer point to the a structure which contains the target keyword. We print the necessary message in the shell and return. For case not found, the function goes on.

### 2.2 auto-completion of command line keywords

If *tab* is detected and input command is incomplete according to the matching result in previous step, shell will search for possible keywords and display the result list with parameters. A function *tshellCmdMatchPart* is added in */libsylixos/SylixOS/shell/ttinyShell/ttinyShellLib.c*. It is called from function *tshellCharTab* of */libsylixos/SylixOS/shell/ttinyShell/ttinyShellReadline.c/*, where shell detects *tab* pressed. Function *tshellCmdMatchPart* first get keyword list of Sylix shell by calling function *tshellKeywordList*, then match input string with keyword list. To print keywords and their parameters, function *printHelpKeyword* in */libsylixos/SylixOS/shell/ttinyShell/ttinyShell-SysCmd.c/* is used to simplify the implementation. To reduce number of matching result, only substrings in the head of keywords are considered. And this could be easily adjust to substrings at any position by removing a "break" in the while loop. Since overall keywords in Sylix shell are no more than hundreds, advanced algorithm like KMP is unnecessary. There is also a convenient operation, which is that the input of next command line will automatically be filled with the content in previous one.

### 2.3 Support find, head, sed utility commands

#### 2.3.1 find

When the command *find* is detected, shell will find files according to specific requirements in current path. *Find* command support users to find files by similar name, specific type or specific size in Bytes. The command format in general is **find path [operation] [content]**. We add a new function *\_\_tshellFsCmdFind* in */libsylixos/SylixOS/shell/ttinyShell/fsLib.c*. to handle the find commnad. In *\_\_tshellFsCmdFind*, the first element in format is ".", which represent the current path, and further development is needed to support find in recursion. In this version, we use *opendir()* and *readdir* to open the current directory and get a *dirent* struct and a *stat* struct of each file, that contains files information. Then the last two elements in format are [operation] and [content], which contains the requirement that files found must satisfy. Our function switch to different branches according to different [operation] and [content].

When [operation] is *-name*, [content] represent the *file\_name*. We compare the *file\_name* with each file in directory and print all file that share same name or contain *file\_name* in its name. We use the *\_\_similarLen* function which can return the number of same char in two string to solve the problem.

When [operation] is *-type*, [content] represent the *file\_type*. Our find support d for directory file,c for char file,b for block file, f for general file,l for link file and s for socket file. We use the macro definition provide by SylixOS to enable the find function.

When [operation] is *-size*, [content] represent the *file\_size*. Our find use file information in *stat* struct of the file to complete the match process.

### 2.3.2 head

The command head was realized by referring the command cat, which includes a file reading operation. I modified the reading flow to read the content char by char, if the char was 'n', it was noticed as the finish of one line. Then by counting the needed line, we can output the needed messages.

```

1  do {
2      cBuffer[0] = '\0';
3      sstNum = 0;
4      do {
5          rntNum = read(iFd, rBuffer, 1);
6          sstNum += rntNum;
7          lib_strlcat(cBuffer, rBuffer, MAX_FILENAME_LENGTH);
8      } while (rBuffer[0] != '\n' && rntNum != 0);
9      fprintf(stdout, "%s", cBuffer);
10     bLastLf = (cBuffer[sstNum] == '\n') ? LW_TRUE : LW_FALSE;
11     NumLine++;
12 } while (sstNum > 0 && NumLine < Tline);

```

### 2.3.3 sed

Similar with head, we can now split the contents line by line. So when an operation given, we can made the corresponding modification on the target contents. However, the development among such bottom level led to tens of thousand errors when we try the file operations. So as a limited debug ability, we finally realized a *fake* sed, only change the output content. But though the sed was a fake one, the parameter analysis was roughly keep the same. This means the process will split the command into three kind: operation only, operation with one line number and operation with two line number. After analyse the line numbers, we will set two variables as the start line(0 if not given) and the final line(INT\_MAX if not given). As long we have get the necessary messages, we then make the operations using the given content. If content was not given or the given operation was not support, error notice will given on the shell.

## 3 Result

In this part, we will exhibition the result and usage of our final implement.

*Notice: All the corresponding figures are in the appendix*

### 3.1 associative help for command line keywords and arguments

if the tab was detected with a existing complete command, it will show the possible parameters and usage as for help, and resume the command again(figure 2).

### 3.2 auto-completion of command line keywords

In the first situation, when input is head of some keywords and *tab* is pressed, only matching results that start with input string will be displayed(figure 3). In the second situation, when input is substring from any position of some keywords and *tab* is pressed, all matching results that include this substring will be displayed(figure 4).

### 3.3 command find

Figure 5 show how *find* perform when find the files with specific name. There are four files, one directory in /root, and **find . -name test** means we want to find files that contains *test*. And the results prints three files with *test* in its name.

Figure 6 show how *find* perform when find the files with specific type. There are four general files, one directory and one link file in /root, and **find . -type f** means we want to find general files. And the result prints four general files information. **find . -type l** means we want to find link files. And the result prints link file *link\_file*.

Figure 7 show how *find* perform when find the files with specific size. **find . -size n** means we want to find files which is n Bytes. And the result prints information of files that has the specific size.

### 3.4 command head

The head command will show the first n lines if given the -n parameter, else print out the whole file. The usage and the result was shown on the following.

### 3.5 command sed

The sed command was not a real 'sed'. It only change the output into the screen but not the file itself. The operation add(a), insert(i), delete(d) and replace(c) was temporarily realized. The usage and the result was shown on the figure 8.

## 4 Conclusion

This project should be a challenge for our group. This section will including the conclusion about:

- Advantages and Disadvantages
- Difficult and Gains
- Reflection and Summary

### 4.1 Advantages and Disadvantages

Our project was developed by only using the bottom functions without calling for other user accessible functions like `__tshellSysCmdHelp()`, which was directly called by help command. In another word, we keep the structure the same as the original project, so the realization keeps easy to read for other developers.

However, the realization for the `sed` command is inefficient and not perfect. That's because the development was according to the modification about memory, which will cause lot's of bugs makes us exhausted. Also, some commands has limited functions, which was due to the limited time and our poor ability.

### 4.2 Difficulties and Gains

The compiling process is a huge time-consuming project, it increase the difficulty for positioning bugs. Due to unknown reason, we encounter misleading problems when using the simulator that stout was not working properly, or different CPU type selected in simulator lead to different execution results. Also, the bug location in such bottom level is such a difficult job that the sometimes even the print function can be a cause of the bugs. For example, incorrect usage of pointer could cause serious bugs, once happened, we got confusing results. Besides, the documents offered in the github was structured in a mass, that's like a goto in a function, but the target was missed.

In all, this project enhanced our ability to search the document and the bug shooting, and build up our capacity for patience. Also, it's worth mentioning that we had also learnt how to read and understand the huge amount of source code in short time during the development, which I think is the most valuable thing.

### 4.3 Reflection and Summary

This project enhanced our ability not only on reading the huge mount of source code, but also the bug shooting, document searching and group cooperation. Our final result is not perfect, but it's a satisfying one for we really gained a lot during the development.

**The lines of code we add and modify for each part varies in hundreds. But we think lines of code is not an appropriate measurement to value our work, as we need to delve into a number of instruction documents and to gain an insight of the details of each function and then try to write code with high compatibility, which we think is more important. To some extent, knowing where to write code at requires no less work and time than knowing what to write.**

It's a lucky thing to have this project as the last one of our bachelor's career.



```

[root@sylixos:/root]# el
All matching results:
qosruledel      [netifname] [rule sequence num]
npfruledel      [netifname] [input | output] [rule sequence num]
gdel            group_name
udel           name
dosfslabel      [[vol newlabel] [vol]]
loglevel        [level]
vardel          [variable]
help            [-s keyword]
shell           [tty device[:hwopt]] [nologin]
[root@sylixos:/root]# mk
All matching results:
mkgrub          [block I/O device]
mkfs            media name
mkfifo          [fifo name]
mkdir           directory

```

Figure 4: command line auto-completion(any position matching)

```

[root@sylixos:/root]# ls
directory      link_file      similartest    testfind      testfile
testfile2
[root@sylixos:/root]# find . -name test
-rw-rw-rw- root    root    Sat May 29 14:13:29 2021    1 B, testfind
-rw-r--r-- root    root    Sat May 29 11:26:23 2021    14 B, testfile
-rw-rw-rw- root    root    Sat May 29 14:13:15 2021    1 B, testfile2
total items: 3

```

Figure 5: find file(s) contain similar name

```

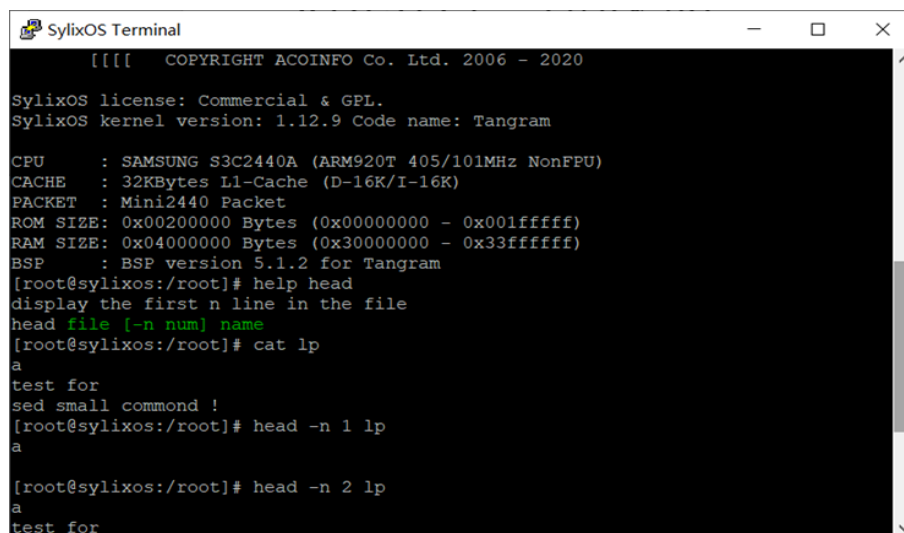
[root@sylixos:/root]# ll
drwxr-xr-- root    root    Sat May 29 17:11:03 2021    directory/
lrwxrwxrwx root    root    Sat May 29 17:09:13 2021    link_file -> .
-rw-rw-rw- root    root    Sat May 29 15:35:46 2021    1 B, similartest
-rw-rw-rw- root    root    Sat May 29 14:13:29 2021    1 B, testfind
-rw-r--r-- root    root    Sat May 29 11:26:23 2021    14 B, testfile
-rw-rw-rw- root    root    Sat May 29 14:13:15 2021    1 B, testfile2
total items: 6
[root@sylixos:/root]# find . -type f
-rw-rw-rw- root    root    Sat May 29 15:35:46 2021    1 B, similartest
-rw-rw-rw- root    root    Sat May 29 14:13:29 2021    1 B, testfind
-rw-r--r-- root    root    Sat May 29 11:26:23 2021    14 B, testfile
-rw-rw-rw- root    root    Sat May 29 14:13:15 2021    1 B, testfile2
total items: 4
[root@sylixos:/root]# find . -type l
lrwxrwxrwx root    root    Sat May 29 17:09:13 2021    link_file -> .
total items: 1

```

Figure 6: find file(s) of specific type

```
[root@sylixos:/root]# ll
drwxr-xr-- root    root    Sat May 29 17:11:03 2021    directory/
lrwxrwxrwx root    root    Sat May 29 17:09:13 2021    link_file -> .
-rw-rw-rw- root    root    Sat May 29 14:13:29 2021    1 B, testfind
-rw-rw-rw- root    root    Sat May 29 15:35:46 2021    1 B, similartest
-rw-rw-rw- root    root    Sat May 29 14:13:15 2021    1 B, testfile2
-rw-r--r-- root    root    Sat May 29 11:26:23 2021    14 B, testfile
      total items: 6
[root@sylixos:/root]# find . -size 0
      total items: 0
[root@sylixos:/root]# find . -size 1
lrwxrwxrwx root    root    Sat May 29 17:09:13 2021    link_file -> .
-rw-rw-rw- root    root    Sat May 29 14:13:29 2021    1 B, testfind
-rw-rw-rw- root    root    Sat May 29 15:35:46 2021    1 B, similartest
-rw-rw-rw- root    root    Sat May 29 14:13:15 2021    1 B, testfile2
      total items: 4
[root@sylixos:/root]# find . -size 14
-rw-r--r-- root    root    Sat May 29 11:26:23 2021    14 B, testfile
      total items: 1
```

Figure 7: find file(s) of specific size of Bytes

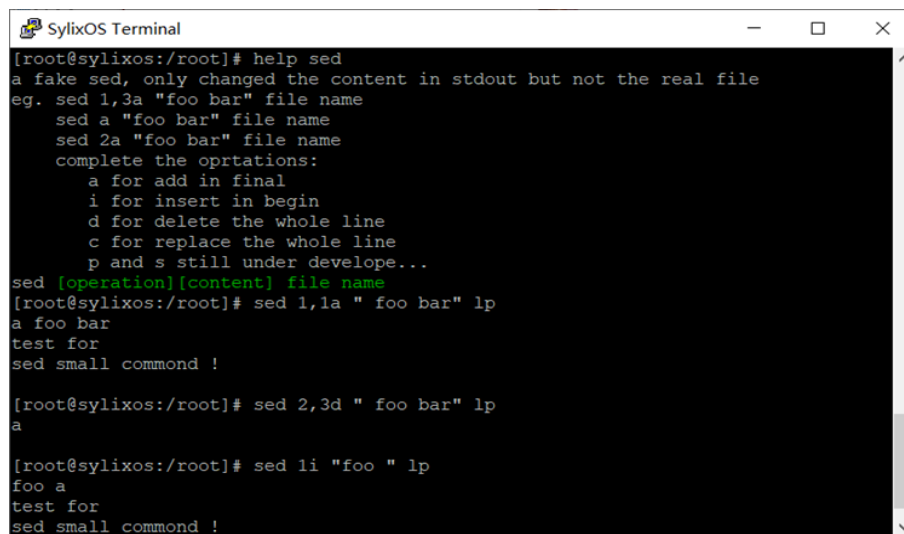


```
SylixOS Terminal
[[[[[  COPYRIGHT ACOINFO Co. Ltd. 2006 - 2020

SylixOS license: Commercial & GPL.
SylixOS kernel version: 1.12.9 Code name: Tangram

CPU      : SAMSUNG S3C2440A (ARM920T 405/101MHz NonFPU)
CACHE    : 32KBytes L1-Cache (D-16K/I-16K)
PACKET    : Mini2440 Packet
ROM SIZE: 0x00200000 Bytes (0x00000000 - 0x001fffff)
RAM SIZE: 0x04000000 Bytes (0x30000000 - 0x33ffffff)
BSP      : BSP version 5.1.2 for Tangram
[root@sylixos:/root]# help head
display the first n line in the file
head file [-n num] name
[root@sylixos:/root]# cat lp
a
test for
sed small command !
[root@sylixos:/root]# head -n 1 lp
a
[root@sylixos:/root]# head -n 2 lp
a
test for
```

Figure 8: command head



```
[root@sylixos:/root]# help sed
a fake sed, only changed the content in stdout but not the real file
eg. sed 1,3a "foo bar" file name
    sed a "foo bar" file name
    sed 2a "foo bar" file name
complete the oprtations:
    a for add in final
    i for insert in begin
    d for delete the whole line
    c for replace the whole line
    p and s still under developpe...
sed [operation][content] file name
[root@sylixos:/root]# sed 1,1a " foo bar" lp
a foo bar
test for
sed small commond !

[root@sylixos:/root]# sed 2,3d " foo bar" lp
a

[root@sylixos:/root]# sed 1i "foo " lp
foo a
test for
sed small commond !
```

Figure 9: command sed