VE482 — Introduction to Operating Systems

Project 1 (Specifications)

Yihao Liu, Modified by TAs later
— UM-JI (Fall 2021)

Goals of the guide

- Clarify the descriptions in the project
- Show some test cases

1 Format

1.1 Input

We will test your shell with a carefully coded driver, so the input format in the test cases will be taken as the driver's `stdin`.

There are three kinds of commands in the input:

- `WRITELN <line>`: Write `line` to your shell ended with a `\n`.
- `CTRL <C|D>`: Send `SIGINT` with `CTRL C` or `EOF` with `CTRL D`.
- `WAIT <T>`: Wait for `T` ms.

1.2 Output

All of output should be flushed if it wasn't flushed automatically by `\n`. If you don't do so, the order of the output of different processes can be arbitrary, so you will get wrong answer.

The newline character (`\n`), tab character (`\t`) and spaces will be considered as the same in the output.

We will redirect your `stdout` and `stderr` to the same file in the driver, so please do not print debug information to `stderr`. The reason why we are doing it is that a shell should print prompts and error messages to `stderr`, though our project doesn't have this requirement. Some students may follow this rule but some will not, so we have to consider them together.

1.3 Examples

The examples will show input in the left and output in the right. Most of them are from milestone 1 and 2, and the real test cases before. We publish these because we are testing your shell functionalities instead of sticking to your carefulness on I/O formats.

2 Tasks

2.1 Write a working read/parse/execute loop and an exit command; [5]

The loop time should be reasonable (<10ms), we will test on your overall performance in this part.
Hint: Don't forget to print `exit` when exiting your shell.

Example (Milestone 1 Case 1)

```
WRITELN exit
```

```
mumsh $ exit
```

2.2 Handle single commands without arguments; [5]

Hint: We only have three files in the working directory, they are `driver`, `mumsh`, `mumsh_mmeory_check`.

Example (Milestone 1 Case 2)

```
WRITELN ls
WRITELN exit
```

```
mumsh $ driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.3 Support commands with arguments; [5]

We have seen someone write programs like `cat`, `ls` themselves, which is totally wrong in a shell. So we have these examples with package managers now.

Example (Milestone 1 Case 5)

```
WRITELN ls -a
WRITELN exit
```

```
mumsh $ .
..
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.4 File I/O redirection: [5+5+5+2]

This task is based on Task 2 and 3.

Only one output redirection and one input redirection is allowed in a command. The test cases will strictly follow this rule.

2.4.1 Output redirection by overwriting a file [5]

Example (Milestone 1 Case 7)

```
WRITELN echo 321 > 1.txt
WRITELN echo 123 > 2.txt
WRITELN cat 2.txt
WRITELN cat 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ mumsh $ 123
mumsh $ 321
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.4.2 Output redirection by appending to a file; [5]

Example (Milestone 1 Case 9)

```
WRITELN echo 321 >> 1.txt
WRITELN cat 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ 321
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.4.3 Input redirection; [5]

Example (Milestone 1 Case 11)

```
WRITELN echo 123 > 1.txt
WRITELN cat < 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ 123
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.5 Support for bash style redirection syntax; [8]

The bash style redirection syntax is very complex, we only require you to implement part of it. The following example actually shows all of the requirements:

- An arbitrary amount of space can exist between redirection symbols and arguments, starting from zero.
- The redirection symbol can take place anywhere in the command.

Example (Milestone 1 Case 18)

```
WRITELN echo 123 > 1.txt
WRITELN echo 222 > 2.txt
WRITELN echo 321 > 4.txt
WRITELN <1.txt>3.txt cat 2.txt 4.txt
WRITELN cat 1.txt
WRITELN cat 2.txt
WRITELN cat 3.txt
WRITELN cat 4.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ mumsh $ mumsh $ mumsh
↪  $ 123
mumsh $ 222
mumsh $ 222
321
mumsh $ 321
mumsh $ 1.txt
2.txt
3.txt
4.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.6 Pipes: [5+5+5+10]

2.6.1 Basic pipe support; [5]

Example (Milestone 2 Case 1)

```
WRITELN ls -a | grep mum
WRITELN exit
```

```
mumsh $ mumsh
mumsh_memory_check
mumsh $ exit
```

2.6.2 Run all 'stages' of piped process in parallel; [5]

We simply limit your total runtime in 1s in the test.

Example (Milestone 2 Case 3)

```
WRITELN sleep 0.6 | sleep 0.6
WRITELN echo success
WRITELN exit
```

```
mumsh $ mumsh $ success
mumsh $ exit
```

A tricky test of executing the shell itself.

Example (Milestone 2 Case 4)

```
WRITELN echo exit | ./mumsh
WRITELN exit
```

```
mumsh $ mumsh $ exit
mumsh $ exit
```

2.6.3 Extend 6.2 to support requirements 4. and 5. ; [5]

Input redirection can only be found in the first command, and output redirection can only be found in the last command, and each of them can occur only once. The test cases will strictly follow this rule.

Example (Milestone 2 Case 5)

```
WRITELN echo 123 > 1.txt
WRITELN echo 321 > 2.txt
WRITELN cat < 1.txt | cat >> 2.txt
WRITELN cat 1.txt
WRITELN cat 2.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ mumsh $ mumsh $ 123
mumsh $ 321
123
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.6.4 Extend 6.3 to support arbitrarily deep "cascade pipes" ; [10]

Do you like the cats in the example? (Miao -,-)

Example (Milestone 2 Case 8)

```
WRITELN echo 123 > 1.txt
WRITELN echo 321 > 2.txt
WRITELN cat < 1.txt | cat | cat | cat
    ↪ | cat | cat | cat | cat | cat |
    ↪ cat | cat | cat | cat | cat | cat
    ↪ >> 2.txt
WRITELN cat 1.txt
WRITELN cat 2.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ mumsh $ mumsh $ 123
mumsh $ 321
123
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.7 Support CTRL-D (similar to bash, when there is no/an unfinished command); [5]

If there is an unfinished command, nothing should happen.

If the command line is empty, print `exit` and exit the shell.

2.8 Internal commands: [5+5+5]

2.8.1 Implement pwd as a built-in command; [5]

Do not use the builtin `pwd` command. Instead, you should implement it yourself.

2.8.2 Allow changing working directory using cd; [5]

You should consider these:

- Execute `pwd` , then execute `cd ..` followed by `cd .` and another `pwd`
- Execute `cd /etc/../etc/./././../etc`
- Execute `cd` alone
- Execute `cd` with more than 1 argument
- Execute `cd ~`
- Execute `cd -`

2.8.3 Allow pwd to be piped or redirected as specified in requirement 4.; [5]

Example (Milestone 2 Case 13)

```
WRITELN pwd > 1.txt
WRITELN cat 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ /out/package
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.9 Support CTRL-C: [5+3+2+10]

Your shell is likely to get TLE (Time Limit Exceeded) if CTRL-C is not handled correctly.

You should NOT use `kill(0, signal)`, it may destroy the judger. You will be sent to the honor council if you intentionally use it.

2.9.1 Properly handle CTRL-C in the case of requirement 4. [5]

Example (Milestone 2 Case 15)

```
WRITELN sleep 10
CTRL C
WRITELN echo success
CTRL C
WRITELN exit
```

```
mumsh $
mumsh $ success
mumsh $
mumsh $ exit
```

2.9.2 Extend 9.1 to support subtasks 6.1 to 6.3; [3]

Example (Milestone 2 Case 16)

```
WRITELN sleep 10 | echo 123
CTRL C
WRITELN echo success
WRITELN exit
```

```
mumsh $ 123

mumsh $ success
mumsh $ exit
```

2.9.3 Extend 9.2 to support requirement 7., especially on an incomplete input; [2]

Example (Milestone 2 Case 17)

```
CTRL C
CTRL C
```

```
mumsh $
mumsh $
mumsh $ exit
```

2.9.4 Extend 9.3 to support requirement 6.; [10]

Example (Milestone 2 Case 18)

```
WRITELN sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | sleep 10 | echo 123
CTRL C
CTRL C
WRITELN echo success
CTRL C
WRITELN exit
```

```
mumsh $ 123

mumsh $
mumsh $ success
mumsh $
mumsh $ exit
```

2.10 Support quotes: [5+2+3+5]

2.10.1 Handle single and double quotes; [5]

Hint: the program name can also be quoted.

Example (Final Pretest Case 2)

```
WRITELN "echo" 'abc def' "xm""fxh"
WRITELN exit
```

```
mumsh $ abc def xmfxh
mumsh $ exit
```

2.10.2 Extend 10.1 to support requirement 4. and subtasks 6.1 to 6.3; [2]

Example (Final Pretest Case 3)

```
WRITELN echo 123 > 1.txt
WRITELN "echo" "<1.'txt'" < 1.txt > "2.""txt"
WRITELN cat 1.txt
WRITELN cat 2.txt
WRITELN ls
exit
```

```
mumsh $ mumsh $ mumsh $ 123
mumsh $ <1.'txt'
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

Example (Final Pretest Case 4)

```
WRITELN "echo" "123" "|" | cat >
↪ 1.txt
WRITELN cat 1.txt
WRITELN ls
exit
```

```
mumsh $ mumsh $ 123 |
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.10.3 Extend 10.2 in the case of incomplete quotes; [3]

This part is similar to some functions in task 11. Consider them together.

You should output "> " after the user hits enter.

Example (Final Pretest Case 6)

```
WRITELN echo 123 > 1.txt
WRITELN echo "<1.'txt'
WRITELN " < 1.txt >> 2."'txt'"
WRITELN cat 1.txt
WRITELN cat 2."'txt'"
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ > mumsh $ 123
mumsh $ <1.'txt'

mumsh $ 1.txt
2.'txt'
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.10.4 Extend 10.3 to support requirements 4. and 6., together with subtask 9.3; [5]

Usually you do not need to modify many things for this task if you design your shell well. It is not tested in the final pretest, but will be tested in the final shell.

2.11 Wait for the command to be completed when encountering >, <, or |: [3+2]

You should output "> " after the user hits enter.

2.11.1 Support requirements 3. and 4. together with subtasks 6.1 to 6.3; [3]

Hint: "»" does not need to be supported.

Example (Final Pretest Case 7)

```
WRITELN echo 123 >
WRITELN 1.txt
WRITELN cat 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ > mumsh $ 123
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

Example (Final Pretest Case 9)

```
WRITELN echo 321 > 1.txt
WRITELN cat <
WRITELN 1.txt
WRITELN ls
WRITELN exit
```

```
mumsh $ mumsh $ > 321
mumsh $ 1.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

Example (Final Pretest Case 11)

```
WRITELN echo 321 |
WRITELN cat
WRITELN exit
```

```
mumsh $ > 321
mumsh $ exit
```

2.11.2 Extend 11.1 to support requirement 10.; [2]

Hint: Actually 10.3 already has this functionality, so you only need to extend to 10.1, 10.2, 10.4.

Example (Final Pretest Case 13)

```
WRITELN echo 123 > 1.txt
WRITELN "echo" "<
WRITELN 1.'txt'" <
WRITELN 1.txt >
WRITELN "2.""txt"
WRITELN cat 1.txt
WRITELN cat 2.txt
WRITELN ls
exit
```

```
mumsh $ mumsh $ > > > mumsh $ 123
mumsh $ <
1.'txt'
mumsh $ 1.txt
2.txt
driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.12 Handle errors for all supported features. [10]

There are eight kinds of errors to be handled. In the test cases of other tasks, none of these errors will be included. You will always get a correct command then. At most one error will occur in a command, so you do not need to consider about the order of errors.

The output format of your errors should strictly follow the format listed below. It is very similar to the bash style error output format.

1. Non-existing program
   - input: `non-exist abc def`
   - input: `echo abc | non-exist`
   - output: `non-exist: command not found`
2. Non-existing file in input redirection
   - input: `cat < non-existing.txt`
   - output: `non-existing.txt: No such file or directory`
3. Failed to open file in output redirection
   - input: `echo abc > /dev/permission_denied`
   - output: `/dev/permission_denied: Permission denied`
4. Duplicated input redirection
   - input: `echo abc < 1.txt < 2.txt`
   - input: `echo abc | grep abc < 1.txt`
   - output: `error: duplicated input redirection`
5. Duplicated output redirection
   - input: `echo abc > 1.txt > 2.txt`
   - input: `echo abc > 1.txt >> 2.txt`
   - input: `echo abc > 1.txt | grep abc`
   - output: `error: duplicated output redirection`
6. Syntax Error
   - input: `echo abc > > > >`
   - output: `syntax error near unexpected token `>'`
   - input: `echo abc > < 1.txt`
   - output: `syntax error near unexpected token `<'`
   - input: `echo abc > | grep abc`
   - output: `syntax error near unexpected token `|'`
7. Missing program
   - input: `> abc | | grep 123`
   - output: `error: missing program`
8. `cd` to non-existing directory
   - input: `cd non-existing`
   - output: `non-existing: No such file or directory`

2.13 A command ending with an & should be run in background. [10+5]

2.13.1 For any background job, the shell should print out the command line, prepended with the job ID [10]

Hints:

- Do not print the process ID.
- Do not print complete information or "mumsh $" after the job is finished (like 'bash' or 'zsh').

– The job ID starts from 1.

Example (Final Pretest Case 23)

```
WRITELN /bin/ls &
WRITELN sleep 0.2
WRITELN /bin/ls | cat &
WRITELN sleep 0.2
WRITELN exit
```

```
mumsh $ [1] /bin/ls &
mumsh $ driver
mumsh
mumsh_memory_check
mumsh $ [2] /bin/ls | cat &
mumsh $ driver
mumsh
mumsh_memory_check
mumsh $ exit
```

2.13.2 Implement the command jobs which prints a list of background tasks together with their running states [5]

Hint: Print all of done and running background processes in this shell.

Example (Final Pretest Case 23)

```
WRITELN /bin/ls &
WRITELN sleep 0.2
WRITELN sleep 0.2 &
WRITELN jobs
WRITELN sleep 0.4
WRITELN exit
```

```
mumsh $ [1] /bin/ls &
mumsh $ driver
mumsh
mumsh_memory_check
mumsh $ [2] sleep 0.2 &
mumsh $ [1] done /bin/ls &
[2] running sleep 0.2 &
mumsh $ mumsh $ exit
```