VE482— Introduction to Operating Systems

Project 1

Manuel — UM-JI (Fall 2017)

Goals of the project

- Write a simple shell
- Run the shell in Linux
- Run the shell in Minix 3

1 The `ve482sh` shell

The main task of a shell is to wait for some user input, parse it and execute a command requested by the user. It should also provide support for input/output redirection and pipelining from a program into another one.

A shell simply consists of a loop that parses the user input. When waiting the shell displays a prompt, here we want our shell to display "`ve482sh $ `".

When a command is input by the user it should be launched in a new process and the shell should block, waiting for the command to end.
A command line is composed of a command followed by some arguments. Arguments are space separated. The shell should exit when the user inputs `exit`. In case of error, such as when a command that does not exists is input, the shell should output an error on the standard error output (e.g. `Error: no such file or directory`).

The shell can be tested by comparing its behaviour to the result of the same commands in the regular Linux shell (e.g. sh, bash, zsh). Note that those commands are only for testing purpose, therefore they are far from being optimal and do not encompass all the features that need to be implemented.

```
ve482sh $ echo 123 | grep 2
ve482sh $ echo 123 > 1.txt
ve482sh $ echo 456 >> 1.txt
ve482sh $ cat < 1.txt
ve482sh $ cat < 1.txt | sort -R > 2.txt
```

The `ve482sh` shell is expected to be running in both Linux and Minix 3.

Hints:

- Useful system calls: `fork()`, `execvp()`, `wait/waitpid()`, `dup2/dup()`, `pipe()` and `close()`.
- A command line is not expected to be longer than 1024 characters.
- The use of the command `system` or of `lex` and `yacc` is prohibited.

2 Grading policy

A total of thirteen requirements, some with sub-tasks, will be considered when grading. For each requirement the awarded marks are display in bold into square brackets.

Important notes:

- In case of a final grade larger than 100, the extra marks will be saved for a bonus;
- A 50% penalty will be applied if commands are not launched in a new process;
- Any work that is not pushed onto ve482 git server will be ignored;

In the following description "requirement x" stands for requirement x, including all its sub-tasks, if any. A requirement having dependencies is considered completed if and only if it is completed together with all it's dependencies.

1. Write a working read/parse/execute loop and an `exit` command; **[5]**
2. Clean exit, no memory leaks in any circumstance; **[5]**
3. Handle single commands without arguments (e.g. `ls`); **[5]**
4. Support commands with arguments (e.g. `apt-get update` or `pkgin update`); **[5]**
5. File I/O redirection: **[5+5+5+2]**
   5.1. Output redirection by overwriting a file (e.g. `echo 123 > 1.txt`);
   5.2. Output redirection by appending to a file (e.g. `echo 465 >> 1.txt`);
   5.3. Input redirection (e.g. `cat < 1.txt`);
   5.4. Combine 5.1 and 5.2 with 5.3;
6. Support for bash style redirection syntax (e.g. `cat < 1.txt 2.txt > 3.txt 4.txt`); **[8]**
7. Pipes: **[5+5+5+10]**
   7.1. Basic pipe support (e.g. `echo 123 | grep 1`);
   7.2. Run all 'stages' of piped process in parallel. (e.g. `yes ve482 | grep 482`);
   7.3. Extend 7.2 to support requirements 5 and 6 (e.g. `cat < 1.txt 2.txt | grep 1 > 3.txt`);
   7.4. Extend 7.3 to support arbitrarily deep "cascade pipes" (e.g. `echo 123 | grep 1 | grep 1 | grep 1`)

   Note: the sub-processes must be reaped in order to be awarded the marks.
8. Support `CTRL-D` (similar to bash, when there is no/an unfinished command); **[5]**
9. Internal commands: **[5+5+5]**
   9.1. Implement `pwd` as a built-in command;
   9.2. Allow changing working directory using `cd`;
   9.3. Allow `pwd` to be piped or redirected as specified in requirement 5;
10. Support `CTRL-C`: **[5+3+2+10]**
    10.1. Properly handle `CTRL-C` in the case of requirement 5;
    10.2. Extend 10.1 to support subtasks 7.1 to 7.3;
    10.3. Extend 10.2 to support requirement 8, especially on an incomplete input;
    10.4. Extend 10.3 to support requirement 7;

11. Support quotes: **[5+2+3+5]**
    11.1. Handle single and double quotes (e.g. `echo "de'f' ghi" '123"a"bc' a b c`);
    11.2. Extend 11.1 to support requirement 5 and subtasks 7.1 to 7.3;
    11.3. Extend 11.2 in the case of incomplete quotes (e.g. Input `echo "de`, hit enter and input `cd"`);
    11.4. Extend 11.3 to support requirements 5 and 7, together with subtask 10.3;
12. Wait for the command to be completed when encountering >, <, or |: **[3+2]**
    12.1. Support requirements 4 and 5 together with subtasks 7.1 to 7.3;
    12.2. Extend 12.1 to support requirement 11;
13. Handle errors for all supported features. **[10]**
    Note: a list of test cases will be published at a later stage. Marks will be awarded based on the number of cases that are correctly handled, i.e. if only if:
    - A precise error message is displayed (e.g. simply saying "error happened!" is not enough);
    - The program continues executing normally after the error is identified and handled;