Operating Systems 2018-B
Assignment 2: Implementing a Shell

Deadline: Monday, 9/04/2018

This assignment should be submitted as a *.tar.gz* archive, with C++ and README files inside. The archive should not contain generated object or executable files.

The assignment should be submitted in groups of 2 students. Names and IDs of the students in the group must be written inside a README file in the archive, and *also* listed in the submission comment in Moodle.

Here is how you can create a flat submission archive of directory ex2 that contains files ex1/README, ex1/shell.cpp, ex1/utils.cpp, and so forth:

```
tar -C ex2 -czvf ex2.tar.gz .
```

Questions about the assignment should be asked in Piazza. Late submissions are penalized 10 points per day, and assignments can be submitted at most 5 days late. Special requests should be directed to your lab TA.

Introduction

In this assignment, you will implement a shell that reads and evaluates commands given by the user on standard input. The code should be written in C++, spread over one or more *.cpp* files.

Overall, the shell will support:

- a prompt that prints the current working directory, and compresses the home directory part of it into "~";
- exiting the shell via an exit command or via end-of-file on input;
- changing the current working directory via a cd command;
- substitution of "~" home directory references, "$?" last command exit status references, and "$VAR" environment variable references in input command line;
- running a single command with parameters, while waiting for its completion or running it in background using "&" suffix;
- reaping zombies — consuming background child processes that have completed their execution, and printing their exit status.

Your shell must compile without errors *or warnings* in a fully updated 64-bit Ubuntu 16.04 LTS distribution using the following command:

```
g++ -o shell *.cpp -std=c++11 -Wall -Wno-vla -pedantic -march=core2 -Os -pipe
    -fstack-protector-all -g3 -Wl,-O,1,-z,combreloc
```

The only casting allowed in the code is const_cast<> (no C-style casts or reinterpret_cast<>). Using glob() and wordexp() functions for variable expansion is not allowed.

You should go over the man page of each external C function that you use (e.g.: man 2 waitpid), and pay special attention to the values that it may return. All errors must be handled; it is advised to take advantage of the perror() function in case of errors.

Part 1 — Shell REPL, built-ins, and variable expansion

Implement a basic read-eval-print loop of the shell. At this stage, you do not need to support running external commands — only cd and exit commands need to be handled. Note that end-of-file should be also handled as a request to exit the shell. This is important for testing the shell by redirecting a file to its standard input. You can send an EOF using Ctrl-D sequence in Linux.

Here is an example of running the compiled shell, interacting with it, and exiting back to Bash (user input is shown in *italics*):

bud-**bundy**:~/ex2$ *./shell*
Welcome to OS SHell!
OS SHell: ~/ex2> *cd ../misc*
OS SHell: ~/misc> *exit*
C ya!
bud-**bundy**:~/ex2$

Exiting upon end-of-file is also supported:

bud-**bundy**:~/ex2$ *echo cd nosuchdir | ./shell*
Welcome to OS SHell!
OS SHell: ~/ex2> OS SHell: cd: No such file or directory
OS SHell: ~/ex2> C ya!
bud-**bundy**:~/ex2$

You can observe that the shell prints the current working directory in the prompt. The current directory can be extracted with getcwd() function, and changed with chdir() function. Note also that the home directory part is compressed into "~" in the prompt. The home directory can be extracted from HOME environment variable using getenv() function.

The recommended method of handling user input is getline() followed by converting the input into a list of tokens, via splitting on whitespace delimiters. Each argument can be then processed separately. The amount of whitespace between arguments is arbitrary.

Add "~" and variable expansion to arguments processing. Expansion should work as follows:

- "~" at the beginning of command or argument should be expanded into user's home directory.
- "$?" inside a command or an argument should expand into last exit status. At this point, only cd command may set non-zero exit status (set it to 1 if cd fails).
- "$VAR" inside a command or an argument should expand into the value of environment variable VAR, or an empty string if there is no such environment variable. VAR contains only alphanumeric ASCII characters or "_" (underscore), and cannot start with a digit.
- All other "$" characters should be left as-is. E.g., "$$HOME$" becomes "$/home/bud$".

Example:

bud-**bundy**:~/ex2$ *mkdir 'yea1$$'; export MyVar33=yea; ./shell*
OS SHell: ~/ex2> *cd ~/misc*
OS SHell: ~/misc> *cd $HOME/ex2/$MyVar33$?$$*
OS SHell: cd: No such file or directory
OS SHell: ~/ex2> *cd $HOME/ex2/$MyVar33$?$$*
OS SHell: ~/ex2/yea1$$> *[User types Ctrl-D]*C ya!

Part 2 — Running external programs

Implement running external commands in foreground using fork(), execvp() and waitpid() functions. Using execvp() from the exec() family of functions enables search for executables in the list given by PATH environment variable.

After fork(), the parent process needs to wait for the child process to terminate using waitpid(), and extract the exit status via WEXITSTATUS() and WTERMSIG() macros, according to whether the child process terminated normally, or was terminated by a signal. In the latter case, the exit status is 128 + the number of the signal that caused the child process to terminate.

Note that if execvp() fails, the child process continues to run. In this case, an error needs to be shown, and the child process must terminate explicitly with exist status 127.

Examples (note that glob patterns are not supported, hence the error on second ls):

```
OS SHell: ~/ex2> echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
OS SHell: ~/ex2> ls
build  shell  shell.cpp  test  test.c
OS SHell: ~/ex2> ls -l test*
ls: cannot access test*: No such file or directory
OS SHell: ~/ex2> echo $?
2
OS SHell: ~/ex2> ls -l $HOME/ex2/test.c ~/ex2/test
-rwxr-xr-x 1 bud bud 8682 Mar 23 01:03 /home/bud/ex2/test
-rw-r--r-- 1 bud bud  369 Mar 23 01:03 /home/bud/ex2/test.c
```

Add support for running external commands in background (while printing their process ID) by appending "&" after the last argument. When running such a command, waitpid() is not called, and last exit status is reset to 0. However, when such a command ultimately terminates, it becomes a zombie process, and needs to be collected by the parent process:

```
OS SHell: ~/ex2> sleep 5 &
[324]
OS SHell: ~/ex2> ps fh
32343 pts/3    Ss     0:00 -bash
  323 pts/3    S+     0:00  \_ ./shell
  324 pts/3    S+     0:00      \_ sleep 5
OS SHell: ~/ex2> ps fh
32343 pts/3    Ss     0:00 -bash
  323 pts/3    S+     0:00  \_ ./shell
  324 pts/3    Z+     0:00      \_ [sleep] <defunct>
```

Implement zombie reaping by calling waitpid() with WNOHANG option in order to check if any child process has terminated. For all such processes, print their process ID and exit status:

```
OS SHell: ~/ex2> sleep 60 &
[407]
OS SHell: ~/ex2> kill 407
[407] : exited, status=143
OS SHell: ~/ex2> pgrep sleep
OS SHell: ~/ex2>
```