Advanced Programming: Yet Another Shell (ysh)

Emilio E. G. Cantón Bermúdez

15/10/2019

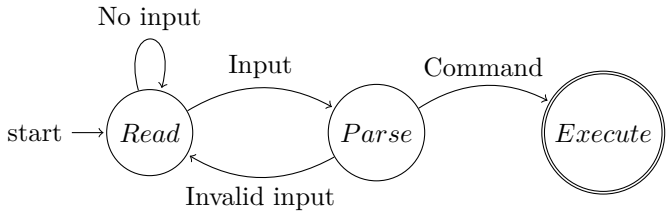
Description

Many of us programmers do not wonder how many of the peculiarities within a shell that we use day to day pass without us thinking how it is implemented. The main problem to be solved with this project is to confront this gap between my knowledge of how to use a shell and how is implemented. The problem may be broken into the following pieces:

- Input parsing
- Running program
- Implement pipelines
- IO Redirection
- Environment variables
- Background processes
- Global expressions
- Job handling (SIGINT, SIGSTOP and SIGCONT)

Solution

With some research I have found that the main process for a shell is the following:



On each step in the process there are certain deviations that can be taken (e.g. multiple piped commands, io redirection, etc. . .). In the following paragraphs I'll explain the functionalities and broad explanation of how each one will be taken care of.

Input parsing

Maybe the most exhausting task of the process, input parsing, will take a line from the input, remove the shell specifications (current working directory, username, among others) and determine the following:

```
- Command is valid or invalid
- Special characters' presence:
    - Pipes
    - Redirection
    - Global expressions
```

Running program

Once we know who much programs, pipes, redirections or other special characters we have, we will `fork` the process accordingly in order to execute the command/s that are specified using the `execvpe` function. In this step we get any errors from the actual execution of the command with the arguments and feedback will be returned to the user if the command was incorrectly ran.

Implement pipelines

In the case we have a number of pipes present we will have the number of processes to be executed and will create a pipe for each '|' character specified. With this, we will use the `pipe` function in order to create a pipe between each command and duplicate the file descriptors from the commands given in order to pass the **stdout** (with fd 1) and **stdin** (with fd 0) into the pipe and generate the desired output.

IO Redirection

For the IO redirection we will be using the special characters '<', '>', '<<' and '>>' in order to `open`, `write` or `read` from the given input or output. With this we will accomplish effects like `ls > somefile.txt` in which the stdio is written to a file.

Environment variables

Using the `getenv` function we will be able to parse and use environmental variables in order to use them in certain programs.

Background processes

We will identify if the char '&' is present in order to send the process to the background and don't `wait` on the child.

Global expressions

In order to parse expressions like `cat *.txt` in which the global expression *one or more* (shell specific) is specified by the character '*', we will implement the functions `glob` and `globfree` along with the `struct glob_t`.

Job handling (SIGINT, SIGSTOP and SIGCONT)

Finally we'll implement a similar function in any other shell for the signals **SIGINT**, **SIGSTOP** and **SIGCONT** in which they react as follows:

```
- SIGINT: Interrupt (terminate) program
- SIGSTOP: Send to sleep program (add to jobs)
- SIGCONT: Resume stopped program in `fg` or `bg`
```

Concept specific explanation

Dynamic memory

This topic will be specifically implemented in many of the necessary variables such as the **pipes** that must be saved on the run.

Pointers

As with any `struct`, pointers will be used for `struct glob_t` and any other that are needed.

Process creation

Needless to say a shell is nothing without process creation, therefore `fork` will be an essential part of the project/program.

Inter process communication

The IPC will be managed mainly with pipes due to the nature of them. I'll be considering in which way I can use sockets as it's a topic that I personally liked very much, but I do not know at the moment of any specific implementation that can be used within a shell.

Signals

Finally, signals are a core theme within a shell as seen in the Job handling section. Therefore signals will definitely be used in order to manage communication between processes executed and jobs.