A Simple Shell[2]

Prof. Darrell Long

Dr. Karim Sobh

Computer Science Department
Jack Baskin School of Engineering
University of California, Santa Cruz

Spring 2017

Assigned: Tuesday 11th April *at 15:00*

Due: Tuesday 25th April *at 15:00*

Purpose

The primary goal of this assignment is to warm up your programming skills that may have atrophied over the inter-quarter break, and to gain some familiarity with the system call interface. A secondary goal is to use some of the programming tools provided in the FreeBSD environment. In this assignment, you are to implement a FreeBSD shell program. A shell is simply a program that conveniently allows you to run other programs.

Basics

You are provided with the files `lex.l` and `myshell.c` that contain code that calls `getline()`, a function provided by `lex.l` to read and parse a line of input. The `getline()` function returns an array of pointers to character strings. Each string is either a word containing the letters, numbers, period (`.`), and forward slash (`/`), or a character string containing one of the special characters: `(`, `)`, `<`, `>`, `|`, `&`, or `;` (which have syntactical meaning to the shell).

To compile `lex.l`, you have to use the `flex` command. This will produce a file called `lex.yy.c`. You must then compile and link `lex.yy.c` and `myshell.c` to get a running program. In the link step, you also must use `-lfl` compiler switch to get everything to work properly. Use `cc` for the compilation and linking.

[2] Although the idea of writing a shell as the first assignment in the operating systems course is an ancient one, dating at least to the venerable Prof. Jehan-François Pâris, I have borrowed liberally from the assignment designed by Prof. Scott Brandt. Even so, it differs in key aspects, which will be observed by the discerning student.

Details

Your shell must support the following:

1. The internal shell commands `exit` and `cd`.
    a. *Concepts*: shell commands, exiting the shell, changing current working directory.
    b. *System calls*: `exit(), chdir()`
2. A command with no arguments.
    a. *Examples*: `"$ ls", "$ ps" etc.`
    b. *Details*: Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect. This holds for *all* command strings in this assignment.
    c. *Concepts*: Forking a child process, waiting for it to complete, synchronous execution.
    d. *System calls*: `fork(),execvp(),exit(),wait()`
3. A command with arguments.
    a. *Examples*: `"$ ls -l", "$ ps aux" etc.`
    b. *Details*: Argument zero is the name of the command other arguments follow in sequence.
    c. *Concepts*: Command-line parameters.
4. A command, with or without arguments, whose output is redirected to a file.
    a. *Examples*: `"$ ls -l > file", "$ cat file > newfile" etc.`
    b. *Details*: This takes the output of the command and put it in the named file.
    c. *Concepts*: File operations, output redirection.
    d. *System calls*: `close()` and some variety `dup()`
5. A command, with or without arguments, whose input is redirected from a file.
    a. *Examples*: `"$ sort < file", "$ cat < file"`
    b. *Details*: This takes the named file as input to the command.
    c. *Concepts*: Input redirection, file operations.
    d. *System calls*: `close()` and some variety of `dup()`
6. A command, with or without arguments, whose output is piped to the input of another command.
    a. Examples: `"$ ls -l | grep file", "$ ps aux | grep sshd" etc.`
    b. Details: This takes the output of the first command and makes it the input to the second command.
    c. Concepts: Pipes, synchronous operation
    d. System calls: `pipe(),close()` and some variety of `dup()`
7. A command that is any combination of the above.

You must check and correctly handle all return values. This means that you need to read the manual pages for each function and system call to figure out what the possible return values are, what errors they indicate, and what you must do when you get that error. You must the specified system calls, and use of other functions/system calls *instead* of the specified ones will be heavily penalized.

The `getline()` function already provides you an input list in a neat array of strings. Parse that array to make sense of the entire line and execute the specified commands in your shell.

Deliverables

Push your project directory (`asgn1`) to your `git` repository (to the `asgn1` branch) including your `README.txt`, design document (`Design.pdf/Design.txt`) and `Makefile`. *Do not* commit any binary files. Include a Readme file to explain anything unusual to the teaching assistant and graders. Your code and other associated files must be in a single directory (`asgn1`) so they will build properly in the submit directory. Your `Makefile` must call `flex` to generate the `lex.yy.c` file and then compile your code, linking with this generated file as the default target. Also provide a target, `clean,` to delete the generated files and executables.

Do not submit object files, assembler files, or executables, or in this case, the `lex.yy.c`. Any file in the submit directory that could be generated automatically by the compiler or assembler will result in a small deduction from your programming assignment grade.

Your design document should be called `DESIGN.txt` (if in plain text), or `DESIGN.pdf` (if in Adobe PDF) and should reside in the project directory with the rest of your code. Formats other than plain text or PDF are not acceptable; please convert other formats (Word, LaTeX, HTML, ...) to PDF. Your design should describe the design of your assignment in enough detail that a knowledgeable programmer could duplicate your work. This includes descriptions of the data structures you use, all non-trivial algorithms and formulas, and a description of each function including its purpose, inputs, outputs, and assumptions it makes about the inputs or outputs.

Deliverables Summary:

1. Design Document (.txt or .pdf) [`git`]
2. README.txt [`git`]
3. Source code (`myshell.c, lex.l, Makefile` and any other C or header files you created for this assignment). [git]
4. Git Commit ID. [eCommons]

Note: Do not copy/paste code from here, either type it manually in or download it from the assignment page in the course web page.

Appendix I: `lex.l`

```
%{
int   _numargs = 10;
char  *_args[10];
int   _argcount = 0;
%}

WORD        [a-zA-Z0-9\/\.-]+
SPECIAL     [()><|&;*]

%%
      _argcount = 0; _args[0] = NULL;

{WORD}|{SPECIAL} {
        if(_argcount < _numargs-1) {
          _args[_argcount++] = (char *)strdup(yytext);
          _args[_argcount] = NULL;
        }
      }

\n    return (int)_args;

[ \t]+

.

%%

char **getline() { return (char **) yylex(); }
```

Appendix II: `myshell.c`

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>

extern char **getline();

int main() {
  int i;
  char **input;
  while (1) {
    input = getline();
    for (int i = 0; input[i] != NULL; ++i) {
      // The input list that must be parsed.
      printf("Item %i of input: %s\n", i, input[i]);
    }
  }
  return 0;
}
```