Exercise #2 - Simple Shell Implementation

Moti Geva

1 Introduction

You are required to implement a simple shell. Your shell should be able to execute both foreground and background applications, as well as IO redirections. Efficiency is important in your implementation, hence you should refrain from making unnecessary system calls or inefficiently calling function calls.

2 Input

- You are to display dollar sign (`$`) followed by a single space whenever waiting for input. Don't add a newline (`\n`) at the end.
- Read a single line from `stdin`.
- Parse the line similar to `bash` (see `man bash` for details), i.e.:
  - `app > file` - redirect `stdout` of *app* to *file* (write output to file)
  - `app < file` - redirect `stdin` of *app* from file (read input from file)
  - `app >> file` - redirect `stdout` of *app* to *file* (append output to file)
  - `app1 | app2` - redirect `stdout` of *app1* to `stdin` of *app2*
  - `app &` - means that *app* should be executed in the background
- Execute the required *apps* with proper redirections and arguments (`argv` list). Your program must look for the application in the `PATH` environment variable.
- If an error occurs
  1. Write the error to `stderr`, in a readable form using the standard error strings (see `man pages` for `errno, strerror` and `perror`)
  2. Log the erroneous command and error string in a log file named '`errors.log`' as a single line in the format:
     $< command\,line >$: $< error\,string >$
     where $< command\,line >$ is the input command, followed by '`: `' (a colon and a space) and the $< error\,string >$.
- Repeat the above process until either the command is '*exit*' or `stdin` closes

3 Notes

1. A single line may contain multiple pipes (|).
2. A single line may contain multiple ampersands (&).
3. Unlike `bash` you may assume:
   (a) Only a single smaller-than sign can appear in the line (<)
   (b) Only a single larger-than sign (>) or alternatively a single double-larger-than sign (>>) can appear in the line.
   (c) An argument for `argv` does not contain any special characters such as white spaces, quotation marks etc.

4 Examples

- Executing *ls* to print the content of *directory/*

```
$ ls direcroty/
aaa  ccc  eee  ggg  iii  kkk  mmm  ooo  qqq  sss  uuu  www  yyy
bbb  ddd  fff  hhh  jjj  lll  nnn  ppp  rrr  ttt  vvv  xxx  zzz
$
```

- Executing *ls* to print the content of *directory/* to a file named *dir.out*

```
$ ls direcroty/ > dir.out
$
```

- Executing *ls* to print the content of *directory/* and redirecting the output to *grep* (which prints only lines containing '*r*').

```
$ ls direcroty/ | grep r
rrr
$
```

- Executing *ls* to print the content of *directory/* and redirecting the output to *grep* and redirecting the output to a file named *dir.out*

```
$ ls direcroty/ | grep r > dir.out
$
```