Programming Asssignment (PA) - 1
(CLI Simulation in C)

CS307 - Operating Systems

17 March 2022
DEADLINE: 28 March 2022, 23:55

1 Introduction

In this assignment, you will implement a command line interpreter(CLI) or, as it is more commonly known, a shell. In this section, we will provide rudimentary information about command-line interface, commands, redirection and background jobs. Please note that this section is to give you a general idea. The shell is not limited to the functionality described here. We recommend you to check the user manual for more details and try the commands on your terminal. Moreover, your implementation will not include everything described here. Exact details of the implementation is provided at Section 2.

1.1 Command Line Interface or Shell

Command Line Interface. The shell is just a user program that enables users to interact with the OS by typing commands. In a loop, it first shows you a prompt and then waits for you to type your command (and any arguments). After getting the input command, the shell decodes the command and creates a child process that is responsible for executing the command you entered. While the child process is executing the command, the parent process (called the shell process) waits for the command to complete. When the child process terminates, the shell process starts the new iteration by prompting again for the next user command.

There are many shells and each kind of shell has its own set of commands and functions. For UNIX based OSes, `bash`, `zsh` and `tcsh` are popular choices.

1.2 Built-in Commands

A shell command is basically a program linked to the shell. The shell processes the commands internally. Some fundamental commands are:

`ls [path]` prints the listing of a specific directory. If no path is given, it lists the current working directory

`cd [path]` changes the working directory. Without any arguments the shell changes to the user's home directory.

`man [command]` is used to display the user manual of a given command. Manual pages are great resources to learn about commands, flags and arguments.

`grep [pattern] [file]` searches for the given pattern in the file. If no file name is given, it searches the whole directory recursively.

`wait [id]` waits for each process identified by an ID report its termination status. If no ID is given it waits for all active child processes.

`wc [file]` counts the number of lines, words, characters, and bytes of each given file and prints it in a newline.

1.3 Options/Flags

Most of the commands have options, also known as flags, that allows to specify the command behaviour. These flags usually shown with a dash followed by one or more letters.

In case of `ls` command supported with `-a` option displays all files including the hidden files.

```
1 ls -a
```

You can use more than one flags and some cases flags may have their own arguments. (Please note that, multiple flags are not in the scope of this assignment however, it is a good practice to know them.)

For instance `grep` command used to search strings in a given file. This command is more powerful when it is combined with the flags. Check out the example below.

```
1 grep -B 2 -A 4 -i "Adleman" lovelace.txt
```

`grep` without arguments displays only the lines containing matching case. However you may want to display the lines before and after the match case.

When you pass the `-B` flag and the number of the lines to be displayed to the command, it displays the matched line and previous lines. Similarly, you can pass the `-A` argument to display the lines after the match. Another handy flag `-i` allows for case-insensitive search.

In the example above grep searches any "adleman" occurence case-insensitively. Then displays 2 lines before the match, the line containing the match, and 4 lines after the match.

1.4 Redirectioning

The standard input/output device of programs is the screen (console). The shell provides mechanisms to redirect console streams (input or output) to files. The `>` symbol is used for redirecting the output, whereas `<` symbol redirects the input.

```
1 ls -a > allfiles.txt
2 grep -i "Adleman" < lovelace.txt
```

In this example, the first command prints names of the all the files and directories, including the hidden ones, to `allfiles.txt` file. If the file does not exist, it is first created. If it already exists, then the file is overwritten. The second command searches the string `adleman` (in a case insensitive way) in the file `lovelace.txt`.

1.5 Background Jobs

The shell allows you to run multiple jobs concurrently (at the same time). This feature is implemented by the ampersand (&) operator. By putting an & to the end of a command, the user requests the shell to run this command in the background meaning that the shell does not wait for this command to terminate before getting its next input from the user. By using this feature, the user can submit multiple commands that run together. If the user wants to wait until one or all of background jobs finish, s/he can use the `wait` command described before.

In the below example, command in Line 2 can be entered while command in Line 1 is running. Hence, `grep` command can execute while `ls` is running. Similarly, both `grep` and `ls` might be running when the user enters the `wait` command. After the `wait` command is entered, the shell blocks until both `grep` and `ls` commands terminate.

```
1 ls -a > allfiles.txt &
2 grep "sample" -i < cs.txt &
3 wait
```

2 Problem Description

2.1 Brief Overview

You are expected to implement a C program that provides a subset of functionalities of a typical Unix Shell. However, your shell does not take its input from the console. The commands are provided in a file called `"commands.txt"`. In this file, each line corresponds to a single shell command simulating a command entered by the user through the console. Your program is expected to execute these commands one-by-one in the given order. For each command, you have to first decode (parse) it and then create a new child process (using the `fork` system call) and execute this command on the child process (using the `execvp` command). The child processes must terminate as soon as the associated command finishes its execution. The parent process (shell process) must terminate after all the commands in the `commands.txt` terminates. If the command does not contain an & at the end, the shell process must wait until the command terminates.

In order to give more detailed information about your implementation, we first need to provide the format of commands you should expect in `"commands.txt"` file.

2.2 Command Format

Each line in `"commands.txt"` is a shell command obeying the following format:

$$cmd\_name\ [input]\ [option]\ [> \mid <\ file\_name]\ [\&]$$

In this format, parts inside brackets ("[ ]") are optional and "|" corresponds to a logical `or` operation. For instance, the expression $[> \mid < file\_name]$ should be interpreted as $> file\_name$, $< file\_name$ or nothing. Then,

- *cmd_name* is one of the following commands: `ls`, `man`, `grep`, `wait`, `wc`.
- *input* is an optional input to the command. Possible inputs of commands can be found in Section 1.2. In the scope of this project, each command can take at most one input. For `grep` command, you can assume that only string pattern can be given as an input. You can also assume that search string is a single word(i.e. it doesn't contain white spaces.). The input file can be provided through redirectioning. For `wait` command, you can assume that it has no input since there is no way to know the `pid` of a background job before running the program.
- *option* starts with a dash (-) symbol, followed by a single letter. After that there can be an optional string following a single white space. For instance, "`grep -i "sample" < cs.txt`" and "`ls -l`" are commands with valid options for this assignment. However, you should not expect commands like "`ls -la`", "`ls -l -a`" although they are perfectly valid commands for any other shell.
- $file\_name$ is a valid path to an existing file. Note that the format allows at most one redirectioning per command. Either the input or the output stream can be redirected to a file, but not both. For more information on redirectioning, see Section 1.4 for more detail.
- Optional "&" operator at the end enforces shell to run this command at the background. See Section 1.5 for more detail.

Below are some sample input files with commands obeying the conditions above:

```
ls -a
man -f grep
grep "sample" -i < cs.txt
```

```
ls -l > out1.txt &
man cat
grep "rwx" < out1.txt &
wait
```

2.3 Parsing

You can safely assume that each line in `"commands.txt"` obeys the format explained in Section 2.2. In `"commands.txt"`, there will not be any blank lines. Also, for every command line, there will be exactly one blank space between every token. However, you still need to parse each line to extract commands, input, options, redirectioning and background operator information. For parsing, you can use the calls `strtok()` and `strchr()`. Check out this page to find out different string functions. After parsing the command, the shell process must print the obtained information to the console in the following format:

First you need to output 10 dash(-) symbols to indicate you are starting to execute a new command from `"commands.txt"`. After that, you need to give information about the command, its input, its option, whether redirection is used or not and whether the command will work as a background job. Every information should be written in a new line. In the end, you need to output 10 dash(-) symbols again.

You only need to write the command name -without input and option- in the `"Command"` output. For `"Input"` and `"Option"` output, if there is no input or option given, you should not output anything specific. If there are not any redirection symbols in the command, you need to put a dash(-) symbol in the `"Redirection"` output. If there exists a redirection symbol, you need to output that symbol. `"Background Job"` part can only get two outputs: 'y' and 'n'. If there exists an ampersand(&) in the command, you need to

output 'y'; otherwise 'n'.

For example, if the command is `"grep "main" -i < input.txt &"`, then your shell process must print the following to the console:

```
----------
Command: grep
Inputs: "main"
Options: -i
Redirection: <
Background Job: y
----------
```

You can check the sample runs for more information.

Warning: Write operations to a file or to the console might not immediately take effect. The OS might wait them for a while and do batch processing for efficiency. We will talk about this in more detail during the *Persistence* section of the course. In order to print the formatted command information to the console immediately, we recommend you to use `fsync()` or `fflush()` system calls after the print statements so that the effects of these print statements immediately becomes visible on the console. Otherwise, the OS might reorder some lines. Even if you use `fsync()` or `fflush()` after print statements, there might be some background jobs running concurrently with the shell process. Hence, their print statements might interleave with shell process' print statements and the console output might be garbled. This is OK for the scope of this PA. We will consider these possibilities while grading your PAs.

2.4 Implementing Command Executions

After parsing, your program (shell process) should *fork* a new child process called command process. If the command does not contain any redirectioning or background operator part, the command process puts parsed components of the command in an array. This array becomes the input for the `execvp` that is executed next by the command process. See the program `p3.c` in your textbook (Section 5.3 of OSTEP) which does a similar task.

There are corner cases you need to consider while giving arguments to `execvp`. Some commands take options as strings which are surrounded by double quotes (`"`). Moreover, options start with a dash character (`-`). All of these characters (dash or double quote) are **special characters** and they must be **escaped** by using some other special characters. It is your task to identify and find ways to escape these characters. In the report you will submit, please write what are the special characters you identified and how you escaped them.

IMPORTANT NOTE: You are **not** allowed to use **system()** library function to create processes and run commands.

2.5 Implementing Redirectioning

If the command contains a redirectioning part like `"> file.txt"` or `"< file.txt`, your command process must redirect the output (resp., input) to the `"file.txt"` file. Note that you cannot do it by changing the command program like opening a file and then replacing every `printf` or `scanf` statement with a `write` or `read` operation to a file. The only clean way you can do this is to first close the standard output (resp., input) and then open `"file.txt"` using the `open` system call. If you do these two steps consecutively, UNIX based OS will associate the `"file.txt"` with the file descriptor of the standard output (resp., standard input) of the console. See the program `p4.c` in your textbook (Section 5.4 of OSTEP) which performs a similar task. In your report, please explain how your program implements the redirectioning in a clear way.

Commands your program will execute might contain at most one redirectioning part. Only the input or the output could be redirected but not the both.

2.6 Implementing Background Jobs

If there is no & at the end of the command, then the shell process waits for the child process executing the current command to terminate. Otherwise it

continues and fetches the next line from `"commands.txt"`.

If the current command is `wait`, the shell process does not create a child process for this command. It waits for **all** background jobs to terminate. In order to achieve this task, shell process should keep track of all the child processes that are executing the background commands. In your report, you are expected to explain your program does this bookkeeping.

When all commands inside `"commands.txt"` file is processed by the shell process, it must also wait for all the continuing background jobs to finish i.e., all the children processes to terminate. You can assume as if there is an invisible `wait` statement at the end of every `"commands.txt"` file.

3 Submission

You are expected to submit a zip file named *<YourSUUserName>_PA1.zip* until 28 March 2022, Monday, 23:55.

The content of the zip file is as follows:

- `report.pdf`: Your report that explains the design decisions you have made for your implementation. You should clearly explain how you handled the escape characters, redirectioning and background jobs (wait method).
- `cli.c`: Your entire implementation must be in this file. We will not consider or compile any other `".h"` or `".c"` files even if you provide them in your submission. We will also not take any other `".c"` file with different name into consideration.

4 Grading

Some parts of the grading will be automated. If automated tests fail, your code will not be manually inspected for partial points. Some students might

be randomly called for an oral exam to explain their implementations and reports.

Submissions will be graded over 100 points:

1. **Compilation (10 pts):** Your source code can be compiled and built without any problems.
2. **Report (20 pts):** Your report clearly explains the design decisions you have made and they are compatible with your implementation.
3. **Parsing (15 pts):** For each command, your shell process prints the command name, input, option, redirectioning and background operator information in the format explained before.
4. **Command Execution (20 pts):** Your program can execute basic commands (except wait) with valid input and option parts. For this grading item, commands will not contain any redirectioning or background operator parts.
5. **Redirectioning (20 pts):** Your program successfully performs input and output redirectioning for commands.
6. **Background Jobs (15 pts):** Your program successfully handles the background jobs and the wait command.

For all grading items $i \in \{3, 4, 5\}$, i is a precondition for the grading item $i + 1$. Grading item 1 is a precondition for every other grading item. If a grading item x is a precondition for another grading item y, it means that you will get 0 from y unless you get full points from x.

5 Sample Runs

We provide two sample runs for you to check and correct your output formatting.

Warning: Using `fsync()` or `fflush()` after printing the formatted commands will ensure that order of these print statements will be preserved and they will immediately take effect. However, if there are background jobs running concurrently with the shell process, their print operations might interleave with shell's print operations and final console view might be different than the sample outputs presented here. While grading your submissions, we will consider the nondeterminism inherent to your programs and you will not lose any points due to valid but unexpected interleavings of processes.

Sample Run 1

```
ls -a
grep varius < input1.txt
```

commands.txt

```
----------
Command: ls
Inputs:
Options: -a
Redirection: -
Background: n
----------
. .. a.out commands.txt hw1.c
----------
Command: grep
Inputs: varius
Options:
Redirection: <
Background: n
----------
Duis feugiat varius arcu,
```

Terminal

```
Lorem ipsum dolor sit amet,
Consectetur adipiscing elit.
Duis feugiat varius arcu,
Eget tincidunt urna tempor et.
```

input1.txt

Sample Run 2

```
man ls > output1.txt
grep description -i < output1.txt
ls -l
wc output1.txt
wait
ls &
```

commands.txt

```
----------
Command: man
Inputs: ls
Options:
Redirection: >
Background: n
----------
----------
Command: grep
Inputs: description
Options: -i
Redirection: <
Background: n
----------
DESCRIPTION
----------
Command: ls
Inputs:
Options: -l
Redirection: -
Background: n
----------
total 40
drwxrwxr-x 2 deniz deniz  4096 Mar 16 21:47 .
drwxr-xr-x 3 deniz deniz  4096 Mar 12 10:38 ..
-rwxrwxr-x 1 deniz deniz 16792 Mar 16 19:25 a.out
-rw-rw-r-- 1 deniz deniz    73 Mar 16 16:22 commands.txt
```

```
-rw-rw-r-- 1 deniz deniz  5163 Mar 16 19:25 hw1.c
-rwx------ 1 deniz deniz  8048 Mar 16 21:47 output1.txt
----------
Command: wc
Inputs: output1.txt
Options:
Redirection: -
Background: n
----------
 211  957 8048 output1.txt
----------
Command: wait
Inputs:
Options:
Redirection: -
Background: n
----------
----------
Command: ls
Inputs:
Options:
Redirection: -
Background: y
----------
a.out commands.txt hw1.c output1.txt
```

Terminal

```
LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILEs (the current
```

```
11      directory by default). Sort entries alphabetically
12      if none of -cftuvSUX nor --sort is specified.
13  ...
14
15
16  (Rest of the "man ls" page)
```

output1.txt