CS3113fa17 Project 1C Polishing your shell

Due 11/21/2017 11:45pm CST (100 points)

Summary

In this project you will put the finishing touches on your shell and create a document that summarizes your project one.

In summary you will:

- A `readme` file describing you process.
- Create an option for your shell to read and execute batch files.
- Add extra internal commands `echo`, `pause`, and `help`.
- Background command execution with the `&` command.

Batch files

Make the shell capable of taking its command line input from a file whose name is provided as a command line argument. i.e. if the shell is invoked with a command line argument:

```
./1c batchfile
```

then `batchfile` is assumed to contain a set of command lines for the shell to process. When the **end-of-file** (EOF) is reached, the shell should exit. When a batch file is provided, the shell should provide no prompt.

Obviously, if the shell is invoked without a command line argument it should solicit input from the user via a prompt on the display as before.

You will probably need to have a look at the difference between the `gets` and `fgets` functions and insert a test for *end-of-file* for when reading from a file (`int feof(FILE*)`). You may find it easier to use `fgets` for both types of input using the default *FILE* stream pointer `stdin` when no batch file is specified. Note the use of `fgets` in the `strtokeg.c` example.

echo

Calling this will print the command line on the display (without the word echo). Remember that `<comment>` can be multiple words. This command always prints to the stdout (even if you try and redirect).

pause

Calling this will display "Press Enter to continue..." and pause operation of the shell until **'Enter'** is pressed (ignore any intervening **non-'Enter'** input). Turning off echo of keyboard input is possible and there is also a system function to read a buffer from the current **tty** input without echo - this is left as a research exercise for the student!

help

This command that will print out your **readme** file. Note that you may not actually still be in the 'startup' directory! Although you can assume that the **readme** file will be in the current working directory when the shell is started. More information about the contents of the **readme** file is below.

Background Execution of External Commands

Having put in the `wait` function to ensure that the child process finishes before the shell gets its next command line, you will now have to provide the option of 'background' operation. i.e. the child is started and then control by-passes the `wait` function to be immediately passed back to the shell to solicit the next command line.

An ampersand (`&`) at the end of the command line indicates that the shell should return to the command line prompt immediately after launching that program. You can assume that `&` will be surrounded by white space

(`' '` , `'\t'` , or `'\n'`) and will be the last non-white space character in the line.

Polishing

You code should exist in the `/projects/1c/` directory. Upon completion, observe and answer the following points, add your observations to the `readme` file.

1. Check that arguments exist before accessing them.
2. What happens when Ctrl-C is pressed during a. command execution? b. user entry of commands? c. would `SIGINT` trapping control this better?
3. The code should check the existence/permissions of input redirection files?
4. Check makefile works

readme

For this project, create a readme file (all lowercase, no `.txt` extension). The `readme` file is a manual describing how to use the shell. The manual should contain enough detail for a beginner to UNIX to use it. For example, you should explain the concepts of I/O redirection, the program environment, and background program execution. The manual MUST be named `readme` and must be a simple text document capable of being read by a standard Text Editor `vim` , `emacs` , etc.

For an example of the sort of depth and type of description required, you should have a look at the online manuals for csh and tcsh (man csh, man tcsh). These shells obviously have much more functionality than yours and thus, your manuals don't have to be quite so large.

The source code **MUST** be extensively commented and appropriately structured to allow your peers to understand and easily maintain the code. Properly commented and laid out code is much easier to interpret, and it is in your interests to ensure that the person marking your project is able to understand your coding without having to perform mental gymnastics!

Again, the readme should contain enough detail for a beginner to UNIX to use the shell. For example, you should explain the concepts of I/O redirection, the program environment and background execution.

In summary, we expect the following from the `readme` :

1. Summary of how to compile and use the shell.
2. Description of *each* internal command line feature.
3. Description of known bugs.
4. Answers to polishing questions.

Grading

The point breakdown is below. Any errors and insufficiencies in any part of the criteria will result in a loss of points. Both the write-up and code should be submitted. Please submit only the text files (.c, .h, .txt) or .pdf files of your write up and code. There is no need to include makefile or build files. Please update your makefile to run an executable `./1c`. Include this data as a `.tar.gz` file and upload to canvas (along with your instance ip address.)

Task	Percent
`readme` created and detailed	60%
Each new feature working correctly	40%
Total	**100%**