ASSIGNMENT 2

CS 361: Systems Programming / Spring 2023

Description

In this assignment, you will implement a shell that allows users to launch processes and redirect input/output. Each command should run in a separate process with a combination of `fork` and `execv`.

Interactive shell

A shell is an interactive program that reads a line from the user, which should contain a command; runs that command; and repeats. The user can exit the shell by pressing <CTRL+C> (i.e., send a SIGINT signal) or <CTRL+D> (i.e., send an EOF character). Here is an example interaction with the shell you will build:

```
cs361sh> echo Hello
Hello
cs361sh> /usr/bin/date
Fri Jan 27 12:41:36 PM CST 2023
cs361sh> thisCommandDoesNotExist
cs361sh: command not found: thisCommandDoesNotExist
cs361sh> <CTRL+D>
<shell exits>
```

Built-in `echo`

Your shell should recognize the command `echo` and handle it specially by printing each argument to that command in a separate line. Example interaction:

```
cs361sh> echo Hello
Hello
cs361sh> echo Hello World
Hello
World
cs361sh> echo This is a long line with many arguments
This
is
a
long
line
with
many
arguments
```

Path search

Your shell will work with full paths for the commands it supports (e.g., `/usr/bin/date`). Your shell should also work with just the name of the executable (e.g., `date`). To find the executable, your shell should search in the contents of the environment variable named `PATH`. You can get the contents of environment variables with function getenv, which takes the name of the variable and returns its contents:

```
char *getenv(const char *name);
```

For instance, if `PATH` is `/usr/bin`, the following interaction should be supported:

```
cs361sh> /usr/bin/date
Fri Jan 27 12:41:36 PM CST 2023
cs361sh> date
Fri Jan 27 12:41:36 PM CST 2023
```

Environmental variable `PATH` may contain many paths, separated by the character `':'`. For instance, consider that `PATH` is `/bin:/usr/bin:/local/bin:.`

In this case, for command `date`, your shell should try to run the following executables in this order:

1. `date`
2. `/bin/date`
3. `/usr/bin/date`
4. `/local/bin/date`
5. `./date`

Your shell should run the first executable that it finds in this list, or print an error message if it cannot find any executable in any path.

Sequencing commands

Your shell should support sequencing commands separated with the character `';'` such that the first command is executed to completion, then the second command is executed to completion, then the shell returns to the user. Here is an example of such interaction:

```
cs361sh> date ; echo Hello World
Fri Jan 27 12:41:36 PM CST 2023
Hello
World
cs361sh>
```

Input/Output redirecting

Your shell should support redirecting standard in, out, and err as follows:

- If there is argument 2> then your shell should redirect the standard err of the command before the argument into the file named after the argument. Example:

```
cs361sh> cat doesnotexist 2>output.err
cs361sh>
```

After this command, file `output.err` should exist with contents `cat: doesnotexist: No such file or directory` which is the error message that `cat` prints to the standard error when it cannot find a file. Note that the message is not printed back to the user in the shell.

- If there is argument > then your shell should redirect the standard out of the command before the argument into the file named after the argument. Example:

```
cs361sh> date >date.txt
cs361sh>
```

After this command, file `date.txt` should exist with contents `Fri Jan 27 12:41:36 PM CST 2023` which is message that `date` prints to the standard. Note that the message is not printed back to the user in the shell.

- If there is argument < then your shell should redirect the standard input of the command before the argument from the file named after the argument. . Example, consider that file input.in exists with the following contents:

```
This
Is
A
File
With
Many
Lines
```

Then, the command `wc -l` which counts how many lines the standard in has, should print:

```
cs361sh> wc -l <input.in
7
cs361sh>
```

- It is possible to combine all redirections in a single command, as long as they appear in the order specified in this document. Assuming file `02.in` exists and the result of calling md5sum on it would be `a9dc8dbff40f4ba4903ee45584243dbc`, here is an example:

```
cs361sh> md5sum 01.in - 03.in 361.in <02.in >test5.out 2>test5.err
cs361sh>
```

After this command, file test5.out should exist with the following contents:

```
7457f155f6824ffcb2d8f9a26ceffcd4  01.in
a9dc8dbff40f4ba4903ee45584243dbc  -
76cc2cbd6954651a8101f0e876504407  03.in
```

And file test5.err should exist with the following contents:

```
361.in: No such file or directory
```

Pipelining

Your shell should support pipelining the output of one process into the input of another, via the character `|`. For instance, consider the following example:

```
cs361sh> echo Hello World
Hello
World
cs361sh> echo Hello World | wc -l
2
cs361sh>
```

In this case, the output of command `echo Hello World` is redirected as the input of process `wc -l` which counts lines, resulting in the output `2` being shown to the user.

Entry Point

You will modify file `a2.c` with your implementation. This file has TODO comments to guide where you should add code.

Due Date and Resubmission Policy

This assignment is due on **February 18th 2023** (Saturday) at **5pm CST**. There is no late policy.

The code and date used for your submission is defined by the last commit to your Git repository.

To resubmit this assignment, you are required to have attendance to Lab Sessions 2, 3, and 4. You can resubmit your assignment until **February 25th 2023** (following Saturday) at **5pm CST**. Together with your resubmission, you will have to submit a written description of what you changed from the original submission (on Gradescope).

Bonus Points

This assignment has a total of **25% bonus points**:

- 10% can be earned by using Piazza as described in the syllabus. Your posts should be public, tagged with the `assignment2` label, and non-anonymous to the instructors to count towards the bonus. You can claim bonus points through **a Gradescope quiz**.
- 15% can be earned by completing Lab Sessions 2, 3, and 4.

Submission and Grading

This assignment is submitted through Github, and has an automatic grade component of 100%. You can check your current grade at any point by submitting your code and checking the autograder. The automatic grade is determined by 10 tests, that will check if your submission outputs the expected result. Each test is worth 10%.

Errors and Omissions

If you find an error or an omission, please post it on Piazza as soon as you find it.

Hardcoding and Academic Integrity

Any hardcoding will result in a 0% grade. Hardcoding is when you submit code that detects which test is being run, and simply outputs the expected result. For instance, detecting that test 22 is running, and replacing the usual execution of your submission with `printf("expected result").`

The academic integrity policy described in the syllabus applies to this assignment. You are responsible for writing all the code that you submit. We will use an automatic tool that detects plagiarism on all submitted code, and we will investigate all instances where plagiarism is more than likely.

Please refer to the syllabus for the full academic integrity policy.