1 Introduction

In this homework, we replicate some the functionality of the standard Unix shell, including running programs with arguments, redirecting stdin, stdout and stderr to files, and pipelines.

2 The Unix shell

Your assignment is to replicate the behavior of a standard Unix shell, in a Rust program.

To get started, create a new project called hw2 using the cargo utility: `cargo new hw2`. This creates a project folder, including `hw2/src/main.rs` which is the file you will be editing. To run your program, `cd` into the hw2 folder, then run `cargo run`. This should produce the output `Hello world!`.

After running `cargo run` or `cargo build`, the binary executable created is `target/debug/hw2`. This is the program we will be testing.

2.1 Grading criteria

For this assignment, we will use one external dependency, the `nix` crate. Add the line `nix="*"` at the end of your `Cargo.toml` file. Grading will be done by entering input in your shell and comparing it against expected behavior.

Below are some expectations on your shell:

- displays a suitable command line prompt including the current working directory, such as `/home/jakob/>`
- executes commands as provided, with or without argument(s)
- executes multiple `;`-separated commands in one line sequentially.
- waits for the last command to finish before displaying the next prompt
- correctly handles the $>$, $<$ and $2>$ operators for redirecting stdin, stdout and stderr to file.
- correctly executes pipelines that include the "|" character, redirecting stdout/stdin as expected
- exits when the "exit" command is given
- does not panic for any input (being very careful about using unwrap() goes a long way here)
- not exit unless the "exit" command or Ctrl-C is issued. Not using the "?" operator in the main function is a good tip, as it would exit on Err()/None.
- Ctrl-C does not exit the shell while the shell is running a program.

For the redirection operators, compare your shell to how bash operates on Linux, particularly when > and | appear next to each other, such as `echo hello > /tmp/hello | grep h`.

2.2 Parsing vs. Executing

There are many moving parts in this homework, which can make it confusing. To reduce complexity, separate your implementation into two pieces: parsing, which takes a command string and produces a data structure (intermediate representation) describing the work to be performed, and execution, which takes this clear data structure and executes it. During development, use debug formatting to print out your intermediate representation, to verify that you parsed your input correctly. *Note: this separation is not a requirement, just advice.*

2.3 Additional suggestions

Most of the functionality you will need have been, or will be demonstrated in class before the assignment due date. Specifically, you will need to use std::process::Command to launch your processes. To connect stdout of one process to stdin of another, use Stdio::piped() and Child.stdout.

Use `nix::sys::signal::sigaction` to configure Unix signals. In particular, you'll want to ignore `SIGINT` while the child is running. Test this by running `sleep 10` in your shell, and hitting `Ctrl-C` during the 10 second wait. The sleep should exit, but your shell should continue running.

To print a regular, inline prompt (that isn't on a separate line), use `print!` instead of `println!`, then use stdout().flush() to make sure the prompt doesn't stay in the output buffer instead of appearing.

2.4 Bonus: parenthesis for better redirection control (20%)

For a 20% additional bonus, implement parentheses for better control of I/O redirection. For example, in bash, `time echo hello 2> /tmp/times` does not redirect the stderr from `time`, but only the stderr from echo (nothing). To save the output from `time` we must use `(time echo hello) 2> /tmp/times` instead, making clear that we want all stderr output within the parenthesis to be redirected, not just the last one.

2.5 Double-check your submission

Before turning in your homework, verify that your program builds and runs correctly after downloading the submitted version, and that it works the way you expect it to.

2.6 turn-in instructions

In your Cargo.toml, please include your name and UIC email address in the authors field, to identify your submission. Submission is via github classroom. Use this link to create your turn-in repository `https://classroom.github.com/a/65I6Zef-`