The maxshell in C project

A shell that supports redirection and pipes

Prerequisites

Before attempting this project, you should complete the following prerequisites:

(1) shell in C
(2) io in C

Function summary

This lesson will introduce the following functions:

dup2 Duplicate a file descriptor

pipe Create pipe

waitpid Wait for a process to change state

Lesson

The most fundamental interface between Unix and normal users is the shell. Examples include the (Ken) Thompson shell, the (Stephen) Bourne shell, the C shell, the Bourne-again shell (BASH), and Zsh. Shells allow users to express commands to execute, and they typically provide easy access to system facilities such as file redirection, pipes, exit codes, and so on. Read Chapters 4 and 5 of *Operating Systems: Three Easy Pieces* [1]. This reading will introduce you to the notion of a process, along with the system call interface Unix provides to control processes.

Assignment

Setup

Please register or sign in to complete this project.

Copy the `Makefile` and the `shell.c` from the shell project into this project's repository. Your submission of this project should support the requirements of preshell and shell in addition the requirements laid out here.

Specification

Implement a shell that reads space-delimited command lines and executes each command with its given arguments. Your shell must also support the redirection of standard input and standard output as well as creating a single pipe between two commands.

Standard input redirection causes a command to consider a file as its standard input. Here the shell arranges things so the command `cat` takes is standard input from the file `/etc/fedora-release`:

```
$ cat < /etc/fedora-release
Fedora release 36 (Thirty Six)
child exit code: 0
```

(The text you would type as input to the shell is red.)

Standard output redirection causes a command to write its standard output to a file. Here the shell arranges things so the command `echo` creates the file `f` and writes to it. Notice the use of `cat` to confirm the contents of `f`:

```
$ echo foo > f
child exit code: 0
$ cat f
foo
child exit code: 0
```

The shell must set things up so that if `f` already exists at the time of running `echo`, then `echo` would try to truncate and overwrite the contents of `f`.

Finally, pipes associate the standard output from one command with the standard input of another. In this example, `tr` removes the 'a' characters from the output of `echo`:

```
$ echo foo bar baz | tr -d a
foo br bz
child exit code: 0
```

Your shell should reject bad command-line syntax. Here are some examples:

```
$ echo >
cannot use '>' without subsequent file
$ echo > > x
cannot use '>' twice
$ echo <
cannot use '<' without subsequent file
$ echo < < x
cannot use '<' twice
$ echo |
cannot use '|' without subsequent command
$ echo | | x
cannot use '|' twice
```

Hints and special considerations

Error handling Each of the system calls (except `exit`) you are to use returns a -1 if it encounters an error. Your shell should write a message to `stderr` (i.e., file descriptor 2) if it encounters an error. Given an error condition, the `perror` function will print a nice string describing what caused it.

Forking The parent shell should fork each process involved in a pipeline, and it must wait twice before prompting for the next command in the case of running a two-process pipeline.

Compiling your program using GCC with GCC's `-Wall`, `-Wextra`, and `--fanalyzer` flags must not produce errors.

Submission

Complete this project using the C programming language. Aside from your source files, you must provide a Makefile in your submission's root directory that builds the executable shell when run with `make`. Please register or sign in to complete this project.

References

[1] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau (2018-08) *Operating systems: three easy pieces*. 1.00 edition, Arpaci-Dusseau Books. External Links: Link Cited by: Lesson.