

IE411 (OS), W22, Assignment 1: IO redirection with descriptors

In this lab assignment you will be creating your own mini-shell, that is, your program will read in command lines, distinguish between program name, command-line arguments, < and > for redirection, and launch new processes matching the command. It will require you to use fork and exec (hopefully that's obvious) and it will also require you to change standard input or standard output when you receive commands like "mname < foo.txt" or "mname > foo.txt". This we'll do at the file descriptor level. So ask yourself: Which file descriptor is "standard in", "standard out"? How do I change that file descriptor so that it refers to a connection to some other file?

You do NOT need to deal with background jobs: you can assume the '&' character will never appear on your minishell command line. Of course, a shell like bash does have to deal with these things, but for this assignment, we're interested in you showing how the shell prepares for and does an exec, not in you showing how to parse the command line.

Here is a link to [example exec calls](#) (and other details about the exec functions).

What you need to do:

- Pre-lab Task (3 pts): Download the tar archive into your system. Develop a working solution to Part 1 below before the upcoming lab session on Jan 28th.
- In-lab Task (2 pts): During the lab session on Jan 28th, demonstrate to the TA that your solution to Part 1 works as expected.
- Post lab Task (5 pts): Complete parts 2, 3 and 4 and submit your complete working minishell.c via classroom. This is due 3rd Feb, 11:55 pm.

Background

The tar archive contains code for a utility called mname (min-max-average-median). This utility will read the input from stdin, i.e. reading until end-of-file (^D, CONTROL-D) is encountered, then printing the results:

```
$ mname
1 2 3
^D
1.000 3.000 2.000 2.000
$
```

mname reads from stdin
the user enters data
the user indicates end-of-input
results are printed
back to the shell prompt

The mname utility will also read from a file named on the command line, but after the file has been processed, the program continues reading from standard input! Note that results are not printed until the user enters ^D. This allows the user to continue entering data from stdin after mname first reads data from the named file:

```
$ mname test_data.dat
1 2 3
^D
1.000 32767.000 16601.027 16617.000
$
```

mname reads data from test_data.dat
the user continues to enter values
the user then indicates end-of-input
results are printed
back to the shell prompt

mname behaves differently when the shell redirects stdin to read from a file, as follows:

```
$ mname < test_data.dat
55.000 32767.000 16650.824 16633.000
$
```

mname reads from stdin (redirected from test_data.dat)
results are printed
back to the shell prompt

Notice that ^D is not entered by the user when IO redirection is handled by the shell.

The command `make mmame` will compile-and-link the program. Do that, and experiment with it a little bit.

Part 1: Minimal minishell

For Part 1 of the project you are to create a program named `minishell`. You are given skeleton code for `minishell` in a file named `minishell.c`. This is your starting point. If you give the `make` command, it'll be compiled-and-linked ... although the version you're given doesn't do anything. Read the source code and follow the numbered comments! `minishell` works as follows:

1. The `minishell` prompt is `"IE411> "`
2. `minishell` reads "minishell commands" from `stdin`, but it is a very simple shell: it "understands" only how to run the program named on its command line. We're testing the `minishell` with the `mmame` program, but you may NOT assume the user won't try to run a program other than `mmame`.
3. Your `minishell` should read user input, then `fork/exec` as appropriate to execute `mmame`.

Here are some examples of the `minishell` in action for a correct Part 1 solution:

```
$ minishell
IE411> mmame
30 25 60 30
^D
25.000 60.000 36.250 30.000
IE411>
```

Start the minishell
IE411 is the minishell prompt. Here we're running `mmame` program
User enters data (`mmame` reading from `stdin`)
User indicates end-of-input with the ^D character
This is `mmame` output
`mmame` terminated: back to the minishell prompt

```
$ minishell
IE411>
IE411>
```

User enters nothing on the command line: just presses the ENTER key
(returns the user to the minishell prompt)

```
$ minishell
IE411> grok
IE411> minishell: grok: command not found
IE411>
```

User enters a command that can't be executed
`minishell` prints an error message
...and returns the user to the minishell prompt

Part 2: Reading direct from file

Add to a correctly functioning Part 1 the ability to execute `mmame` reading directly from a file. Do not attempt Part 2 unless Part 1 works completely and correctly. Here are two examples of the `minishell` in action for a correct Part 2:

```
$ minishell
IE411> mmame test_data.dat
^D
55.000 32767.000 16650.824 16633.000
IE411> mmame test_data.dat
1 2 3
^D
1.000 32767.000 16601.027 16617.000
IE411>
```

Part 3: Input Redirection

Add to a correctly functioning Part 2 the ability to execute mmame using input redirection. Do not attempt Part 3 unless Part 2 works completely and correctly. Here's an example of the minishell in action for a correct Part 3:

```
IE411> mmame < test_data.dat
55.000 32767.000 16650.824 16633.000
```

You may assume that only well-formed commands are entered on your minishell command line. A well-formed command looks like this:

```
mmame < filename
```

You DO NOT have to worry about handling lines like these:

```
mmame foo.dat <      (< In The Wrong Location)
mmame <foo.dat        (Missing Space After <)
mmame <               (Missing Filename After <)
< mmame foo.dat       (mmame not first)
```

Part 4: Output Redirection

Add to a correctly functioning Part 2 the ability to execute mmame using output redirection. Do not attempt Part 4 unless Part 2 works completely and correctly. If the file to which output is redirected already exists, the old version will be overwritten. Here's an example of the minishell in action for a correct Part 4:

```
IE411> mmame > foo.dat
1 2 3
^D
IE411> ^D
$
```

You may assume that only well-formed commands are entered on your minishell command line. A well-formed command looks like this:

```
mmame > filename
```

You DO NOT have to worry about handling lines like these:

```
mmame filename >     (> In The Wrong Location)
mmame >filename       (Missing Space After >)
mmame >              (Missing Filename After >)
> mmame filename     (mmame not first)
```