CSCE 313.506-F18:
Programming Assignment 2

Tristan Seifert

Due: Sunday, Oct. 7, 2018

---

1 Parsing

User input is parsed in four stages – first, by splitting the string by pipes; then, parsing input/output redirection, followed by finding an ampersand to put processes in background, and lastly, splitting the process arguments by spaces. During each of these stages, quoted strings are treated as a single token – so a pipe or an ampersand in a quoted string will be ignored, for example.

At the end of parsing, the user's inputted command line is transformed into one or more `Fragment` classes, which are then executed by the `Shell` class.

`Shell` checks if the program name is one of the built-ins – `cd, exit, jobs` – and if it is, simply runs a function to handle it. If the program name cannot be executed without launching another process, `execvp()` is used to launch it, and any other processes.

2 Piping

When the shell determines that a single command line resulted in two or more `Fragment`s, it sets up n number of pipes, where n is equal to the number of fragments.

```
// Create pipes (one per fragment; [0] is stdin, [1] is stdout)
int pipes[fragments.size()][2];

for(int i = 0; i < fragments.size(); i++) {
  err = pipe(pipes[i]);
}
```

The standard output of the first fragment connects to the input of the second, and so forth, until all fragments have had their `stdin`/`stdout` redirected. If needed, the first process' input may be read from file, while the last one's output may be written to a file.

```
// connect stdin
int stdinIndex = (i - 1);

if(stdinIndex >= 0) {
  err = dup2(pipes[stdinIndex][0], STDIN_FILENO);
}

// connect stdout for all but the last process
int stdOutIndex = (i - 0);

if(i < (fragments.size() - 1)) {
  err = dup2(pipes[stdOutIndex][1], STDOUT_FILENO);
}
```

Note that while the pipes are created in the shell's process, connecting them to the `stdin`/`stdout` takes place in the child process: e.g. after `fork()` returns zero; additionally, error handling was omitted in this excerpt.

3 Redirection

When IO redirection is desired, the specified files are opened for reading or writing, and replace the existing standard input or output using the `dup2()` syscall:

```
// redirecting standard input
int newStdin = open(it->stdinFile.c_str(), O_RDONLY);
dup2(newStdin, STDIN_FILENO);

// redirecting standard output
int newStdout = open(it->stdoutFile.c_str(), (O_RDWR | O_TRUNC | O_CREAT), 0644);
dup2(newStdout, STDOUT_FILENO);
```

Note that this redirection takes place in the child process: e.g. after `fork()` returns zero; additionally, error handling was omitted in this excerpt.

When executing a single process, both `stdin` and `stdout` can be redirected; but when pipes are involved, the first process can only have its `stdin` redirected, the last process can only have its `stdout` redirected, while any other process between the first and last cannot have file redirection.

4 Customization

The shell can be customized with a `.kushrc` file in the directory from which it is executed. This file is structured like an INI file, and might look something like this:

```
[prompt]
# Text to show as the prompt. You can substitute $STATUS, $USER, $HOST, $DIR
# $DATE and $TIME. A trailing space is automatically added.
# To colorize the status code (0 is green, any other color is red) you can
# use $COLORSTATUS.
text = {$COLORSTATUS} \e[31m$USER\e[0m@\e[34m$HOST\e[0m - \e[32m$DIR\e[0m ($DATE $TIME) $
```

In this case, the ANSI escape sequences (`\e[31m` and so forth) are used to colorize the prompt. This example prompt might look like the following:

```
{0} tristan@Bird-Machine.local - /Users/tristan/TamuCS/CSCE313/PA2 (10/07/18 19:21:34) $
```

The shell will substitute `$COLORSTATUS` with a colorized version of the `$STATUS` variable – green if the status code is zero, red otherwise.