Iteration VII - Using Pipes

Due: April 13th, 11:59:59 pm

Useful man pages

dup, dup2, setpgid, select

NOTE:

Object I made

```
Parser       -   Parses the input into tokens
Executor     -   Uses the tokens to create a Process
Process      -   Object that symbolizes a process in the Operating System
ProcessGroup -   Object that represents a group of processes
Signal       -   Object that symbolizes a signal in the Operating System
ProcControl  -   List container of Processes
```

Overview

We can now signal signal processes, be it to make them suspend, continue or die. Now we must get processes to communicate with one another. To do this we introduce the concept of the pipe.

* Using pipes to direct information

Description:

You are to extend your previous program by adding the ability to do the following:

1. All operations done in the previous iterations must work. This means that all tests performed on previous iterations need to continue to work.

2. The lines of input will look like the following:

   ```
   command1 arg1 arg2 | command2 arg1 arg2 ...
   ```

3. Each "|" represents a UNIX pipe between the surrounding programs. That is the stdout of the program on the left should be connected to the stdin of the program on the right. This pattern continues until the last command is reached.

4. Each line represents ONE process group. That is, while there may be more than one program on one line - there should only be one group assigned to all of them.

5. In order to send a signal to the whole group, you need only send the signal to the group leader (i.e. the first program that formed the group).

6. All signals that are received to merely be forwarded the foreground group. This includes the handling of SIGINT, SIGTSTP and any other signal you wish to handle. DO NOT cause SIGTSTP to suspend a process group.

7. Interactive programs should now work, though it is possible that they might not. This is because you are not handling all the signals that might need to be dealt with and the fact that you might be waiting on a process (and not watching the file descriptors that want input).

```
    Also, if you would like to use threads you may.  Another way to
    accomplish the monitoring of multiple file descriptors is to use the
    select() function.

8.  Remove ALL debugging data.  This means that the argument tables
    should NO LONGER be printed.  The only output should be information
    to the user that is pertinent.

Example (this is NOT output)
============================

Input:  ls| grep c

Program 1: ls
Program 2: grep c

The stdout of ls is connected to the stdin of grep.

So the path that the data takes should be:

User --> Shell --> ls --> grep --> Shell --> Screen

The following path is also acceptable:

User --> Shell --> ls --> grep --> Screen
```