Programming lab 2: runPipe

Submission deadline: May 31, 2015

Introduction

The aim of this programming assignment is to write a C/C++ program that given a YAML description executes a set of programs chaining their outputs and inputs using pipes. Some requirements are:

- Each job described in the input file has to be spawned as an individual process.
- All processes should be run concurrently.
- The standard input and output of each process can be redirected (or not) to either a file or some other process.
- Each pipe can be assumed to be disjoint(nonoverlapping) with all the other pipes.
- If one or more processes were not mentioned explicitly in any pipes they are assumed to be in a default pipe arranged as in the original jobs array.
- Only the standard output has to be redirected.

Assignment (80%)

- (30%) Write a program `runPipes` that receives a file with a list job descriptions and pipes between them (using YAML) as input, and run each job in his corresponding pipe.

  An example of input would be:

```
$ cat desc.yml
Jobs :
 - Name : "job1"
   Exec : "echo"
   Args : ["you", "gotta","be","kidding"]
 - Name : "job2"
   Exec : "tr"
   Args : [" ","_"]
 - Name : "job3"
   Exec : "yes"
   Args : ["No","I'm", "not!"]
 - Name : "job4"
   Exec : "head"
   Args : []
Pipes :
 - Name : "pipe1"
   Pipe : [ "job1","job2" ]
   input : "stdin"
   output : "stdout"
```

An example output/execution of `runPipes` would be:

```
$ runPipes desc.yml
## Output pipe1 ##
you-gotta-be-kidding
## pipe1 fished Successfully ##
## Output default-pipe ##
no I'm not!
no I'm not!
no I'm not!
no I'm not!
no I'm not!
no I'm not!
no I'm not!
no I'm not!
no I'm not!
no I'm not!
## default-pipe fished Successfully ##
```

This is equivalente to (the order is irrelevant):

```
$ echo you gotta be kidding | tr " " _ &
$ yes No "I'm" "not!" | head &
```

- (10%) Annotate your source code in a way that explains your solution for the assignment.
- (5%) Use GNU`/make` with at least this targets

  build (Default) Compiles the code, and places the resultant executable on `bin/`

  run Runs your program with some examples placed on `examples/`

  clean Delete files from compilation.

- (5%) Use the following folder structure

```
├── bin/
├── examples/
├── src/
├── README
└── Makefile
```

- (30%) Attend a revision, for the grading of the following topics:
  - Code understating.
  - Teamwork.
  - Concepts used in the solution.

Date: June 1 2015

Clean Code (10%)

Before submitting your code, clean it up! Clean code:

- Does not have long lines (at most 80 columns).
- Has a consistent layout.
- Has no junk (unused code, commented lines of code, unnecessary code).
- Has no overly complicated function definitions.
- Does not contain any repetitive code.
- Has no unnecessary spaces at the end of a line, or empty lines at the end of a file.
- Has no Tabs! (Except for the `Makefile`)

Submission (10%)

The submission will be through Bitbucket, please create a private repository and grant read access to jfcmacro (teacher) and agomezl (TA) and follow these guidelines:

- Do not include binaries.
- Include a README (or README.md), a file containing at least the following information.
  - Your name(s).
  - A general description of your program.
  - Information on how to use your program.
- The last commit before the deadline is the one that will be graded, no further commits will be accepted.

Plagiarism policy

Feel free to discuss with your classmates about the lab, but please do not use code that is not yours. Any plagiarism will be graded with 0 for all students involved and may imply disciplinary sanctions.

Formats

`jobRun` job description (Input file)

A job description is a YAML object with fields `Jobs` and `Pipes` each of which contains a list of jobs and pipes respectively.

```
Jobs :
 - Name : <Job Name>
   Exec : <Executable>
   Args : [<Arguments>]


Pipes :
 - Name : <Pipe Name>
   Pipe : [<Jobs>]
   input  : <Input>
   output : <Output>
```

`<Job name>` Is a descriptive name for the job.

`<Executable>` Is the name of the program that will be run, either with an absolute or relative path (this is optional).

`<Arguments>` Is a list of arguments for the program.

`<Pipe name>` Is a descriptive name for the pipe.

`<Jobs>` The list of jobs in that pipe. note that the order in the list is the actual redirection order.

`<Input>` Whether to read from standard input (stdin) or from a file.

`<Output>` Whether to write to standard output (stdout) or to a file.

Output*

```
## Output <Pipe Name> ##
<Program output>
## <Pipe Name> fished <Exit Status>  ##
```

`<Pipe Name>` Is the name of the pipe being executed (same as in the input file)

`<Program output>` Is any output the pipe may produce. (Could be none)

`<Exit Status>` Can be one of:

- `Successful`
- `Unsuccessful(Err:<errcode>)`

with `<errcode>` being the exit code of the last job in the pipe.

* one per pipe.

Notes

- The programming lab may either be solved on your own, or jointly with one other classmate.
- Follow the yaml spec.
- It is required to use `x86_64 GNU/Linux` as target architecture.
- This lab will be run on a Fedora 21 machine, please be as clear as possible with the required dependencies of your program.

- Feel free to use any yaml parsing library of your liking, just be aware that such library should be free and publicly available. Also don't forget to properly document his usage and installation.

Recommendations

- Use your common sense.
- Don't assume things.
- If you don't know something just ask.
- Be pragmatic.