Project 1

ESE 333 Real-Time Operating System, Spring 2018
Instructor: Prof. Fan YE
fan.ye@stonybrook.edu

---

1 Introduction

As we introduced in class, the four functional characters (<, >, |, &) are for input redirection, output redirection, pipe and background jobs respectively. By default, they can be interpreted correctly by a UNIX terminal, but not by a C/C++ program. In a UNIX terminal if you type in "echo hello > file1", you will find a file named "file1" created which has string hello stored inside, and thats because the terminal interpreted ">" correctly and redirected the output of string hello from screen to file1.
In our Project 1, we want you to write a C/C++ program to interpret all these four functional characters as a UNIX terminal does.

2 Description

When your program is running, we expect it to allow the user to type in a command line, and if the user types in

```
echo hello > file1
```

, the same thing as in a UNIX terminal should happen. This is only the test for output redirection, and we expect the other three functional characters to be interpreted correctly as well. Note that function "**system**" cannot be used in your program because it will do everything for you. For example, system (echo hello > file1); can give the right result without needing you to do anything. Instead, we want you to use system calls for process management and file systems, such as **fork, waitpid, execvp, exit, pipe, dup, open, close**, etc. First, your program should read in the users command line and store it in an array. Then it checks if any of the four functional characters is in the array and will do corresponding work if so.

Take as an **EXAMPLE** the "output redirection", if we input

```
echo hello > file1
```

, the C/C++ program should read them in, and store them in an array whose name is args, with args[0] having echo, args[1] having hello, args[2] having > and args[3] having file1. Then the program checks if any of the four functional characters exits, and yes, > is found in args[2]. After this the program can do corresponding work: since > is in args[2], the following argumentargs[3]should be the output redirection destination, thus the program creates a file called file1 (using system call create) and redirects standard output to file1 (using system call "$dup2$", there can be multiple solutions). Finally, the program executes the command by calling

```
execvp(args[0], args);
```

. You can assume that the input will consist of one command per line and have at most one pipe. For the usage of system calls, besides the ones we discussed in class, you may look

at the on-line manual in UNIX system called "*man page*" by typing shell command "*man system_calls_name*", for example, "man fork".
Your code should be well documented so that others may read and understand your code. You should give instructions on how to run your program, and indicate the machines you used. The TA will test your program on UNIX Lab machines. Note that this is an **INDIVIDUAL** project. You should work independently and submit your own program. You should email your program to the TA with subject title "ESE333 Project 1" before deadline.

3 Commands that your program needs to support

command name	symbol	test example	points
Output redirection	>	*echo hello* > *file*	%25
Input redirection	<	*cat* < *file*	%25
Pipe	|	*man date* | *grep os*	%25
Background	&	*sleep* 100 &	%25

Extra credits: other commands like "cd" (%10)