CSE 109: Systems Programming

Fall 2017

Program 6: **Due on Wednesday, November 8th at 9pm on CourseSite.**

Collaboration Reminder:

1. You must submit your own work.
2. In particular, you may not:
   (a) Show your code to any of your classmates
   (b) Look at or copy anyone else's code
   (c) Copy material found on the internet
   (d) Work together on an assignment

Assignment: Preparation

1. Make a *Prog6* directory in your class folder.
2. You will be writing a *Command.cpp*, *Command.h*, *ExecutionPair.cpp*, *ExecutionPair.h*, *Piper.cpp* and a *Makefile* file for this assignment.
3. All source code files must have the comment block as shown at the end of this document. All files must be contained in your *Prog6* directory.
4. You are expected to check return values (except for printf/scanf and their variants) for errors via errno and report them (usually aborting when this happens).

Assignment:

In this assignment, you will have to create a program that will execute other programs that are provided by a file. You will do management of the communication between these programs and report on their status. Read the entire layout of the program first, some things are referred to before they are defined. Basic things such as constructors and destructors are assumed at this point. Make sure not to memory leak!

Assignment: Piper.cpp:

1. This will contains your main function. The executable name will be *piper*.

2. Ensure that we've received one commannd line argument (the file that tells us what other programs to execute). If not, output an error message to standard error and return 1.

3. Open the file for binary I/O. If an error occurs, report it and return 2.

4. The first 2 bytes of the file represent an *unsigned short* telling you how many cases exist in the file.

5. Each case is an *Execution Pair* and the organization of that data is explained in that section.

6. For each *Execution Pair* create a *ExecutionPair* object.

7. Execute each *ExecutionPair* object.

8. Wait for all child processes to die and report on their behaviors. See "Waiting" section.

Assignment: ExecutionPair:

1. Each ExecutionPair object will consist of two Command objects (referring to the commands that comprise the pair (or each case of the input file).

2. ExecutionPair will contain 2 *Command* objects that reflect the pair of commands that must be run.

3. ExecutionPair will contain the data related to the Communication Model.

4. *execute(ExecutionPair *pair)*: See "Execution" section.

Assignment: Command:

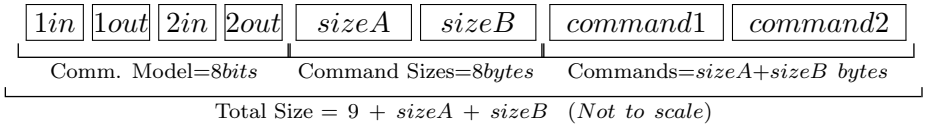
1. Command will contain at least the string representing the entire command that will be run.

2. Command should contain some way to acquire a *char** argv* suitable for use with *exec*.

3. Command must not contain the data related to the Communication Model - it should not be aware of it.

Assignment: Execution Pairs

Execution Pairs are variable-sized chunks of data consisting of 8 components. The first 4 components (*1in, 1out, 2in, 2out*) are 2 bits each (1 byte total, the first byte of the *Execution Pair*), these indicate the communication model. The next 8 bytes are two unsigned integers, telling you the size of the two commands that comprise this *Execution Pair*, call these *sizeA* and *sizeB*. The next *sizeA* bytes is the string representing the first command, it is not null terminated. The next *sizeB* bytes is the string representing the second command, also, not null terminated.



1in is the least significant 2 bits, then 1out, 2in and 2out is the most significant 2 bits.

The commands are the two programs (the pair) to be executed - this also includes command line arguments (there will be no multi-word command line arguments).

The critical aspect of this is the communication model which determines what happens to standard input and standard output for each of the commands. *1in* and *1out* refer to the standard input and standard output for the first command, likewise with *2in* and *2out*. The value of each of these elements indicates an attachment. Specifically, the value of *1in* tells us if refers to standard input or if it refers to the output of command 2.

Comm Model ID	2-Bit Values			
	00	01	10	11
1in	stdin	invalid	invalid	stdout2
1out	invalid	stdout	stdin2	invalid
2in	invalid	stdout1	stdin	invalid
2out	stdin1	invalid	invalid	stdout

You will not be given invalid values for the communication model. For each input/output of each command, there are two valid values for the communication model. One of these values indicates the default case (nothing

to do on your side). The other value indicates that we connected to the other command. This connection is reciprocial. If *1in* is connected to standard output of the second command, then *2out* must be connected to the standard input of the first command.

Assignment: Execution:

ExecutionPair objects can be executed to run their pair of associated commands, with command line arguments, with appropriate pipes allowing them to communicate with each other, if applicable.

1. If any communication occurs between the two commands, as per the communication model, create *pipe*s to distribute to the two child processes created by the commands. You will need to come up with some way for the children to know what to do with these *pipe*s before they *exec*.
2. Use fork to create the two child processes.
3. Parent does not wait, you can just return to your caller (Piper).

   Make sure the parent closes any *pipe*s it may have open.
4. If a child needs to use a *pipe*, close/replace your file descriptor with the appropriate end of the *pipe*. Make sure to close the unused end of the *pipe*.
5. Use execvp to execute the command associated with the child with its corresponding command line arguments. If execvp fails, print to standard error and exit(3).

Assignment: Waiting:

After Piper executes all of the *ExecutionPair*s, it must wait for all child processes to terminate.

1. Use waitpid and repeat until you receive an error.
2. If that error is ECHILD (no waiting children), do not report the error. Clean up any memory and unclosed files and finish.
3. If the error is not ECHILD, report it and exit(5).
4. If a child terminates, print to standard output the status of the child:

(a) Examples, match as best as possible:

(b) Child X was terminated abnormally with signal #.

(c) Child X was terminated abnormally.

(d) Child X returned #.

Where X is the pid of the value and # is some numeric value.

Testing:

1. Some test files may be provided in the *p6student* directory.

   (a) *p6ref* will be the reference executable.

   (b) *file#* are the test cases. More may be added later.

   (c) Some debug messages are leftover, you can hide them by redirecting stderr. However, these may give you a clue as to the issues that I encountered writing the reference executable.

Submission:

1. Once ready to submit, you can package up the assignment as a .tgz file

   tar -czvf Prog6.tgz Prog6

   You must use this command in the directory that contains the *Prog6* folder, not within the directory.

2. Transfer *Prog6.tgz* to the Program 6 submission area of CourseSite.

Comment Block:

```
/*
    CSE 109: Fall 2017
    <Your Name>
    <Your user id (Email ID)>
    <Program Description>
    Program #6
*/
```