CSCI 1730 Breakout Lab 12

Interprocess Communication Using Pipe and Dup2

Learning Outcomes

- Design and implement programs that use pipe and dup2 for input/output redirection and interprocess communication.
- Design and implement programs that use system calls and signal handling (fork, exec, wait, etc.) and run concurrently.

Problem / Exercise

WARNING: DO NOT abuse the fork system call for this lab!

The purpose of this lab is to get experience using the C programming language to make UNIX system calls to pipe and dup2 for interprocess communication between a parent process and child process that run concurrently. You will write a C program that uses a single call to the system call pipe (without using a | character), and A SINGLE CALL TO fork to split itself into a parent process and a child process that communicate through a pipe. The parent process will wait for the child process to run the executable named in the second command line argument and any of its command line arguments to the left of the `-pipe` flag as shown in the Examples section. After the child process is finished running, the parent process will run the executable named in the command line argument after the `-pipe` flag and any of its command line arguments to the right of it as shown in the examples in the Examples section. The standard output of the child process should be piped into the standard input of the parent process without using a | character anywhere in your source code.

Requirements

1. Read through the instructions and examples in this document and section 15.2 in your Advanced Programming in the UNIX Environment, Third Edition (Stevens & Rago) textbook.
2. Your program must use a single system call to pipe. Do not use a | character anywhere in your source code. Instead, use the pipe system call. If one or more | characters are found anywhere in your source code, then you will receive a grade of zero on this assignment.
3. Your program must use A SINGLE CALL to fork. Points will be deducted if you call fork more than once (do not place it inside a loop nor any other repetition statement). Abusing the fork system call may result in a conduct violation for our course, which may have heavy consequences to your final grade in our course.
4. Your program must call wait, dup2, and execvp where appropriate. The parent process and child process should each call the C library function execvp.
5. Your program CANNOT use a call to `system`. If your program uses a call to `system`, then you will receive a grade of zero for this assignment.
6. Implement this lab in a C source file called `lab12.c`, and use our provided Makefile to create an executable called `lab12.out`. Do not modify our Makefile. Your source code must compile correctly (no warnings) without any modifications to our Makefile.
7. Your program's I/O must adhere to the instructions in the Examples section.
8. You may assume valid input for this lab. Also, you may assume that inputs for this lab will contain only one `-pipe` flag.
9. If a function call fails in your program, then your program should print (to standard output) a reasonable error message and terminate gracefully.

Examples

After you finish implementing the `lab12.c`, the I/O from running `lab12.out` should match our examples exactly when run on a vcf node. The words.txt file is provided for you on eLC, and it should be put in the same folder on a vcf node as `lab12.out`. You may assume all executables and command line arguments are going to be passed to your program as valid inputs on a vcf node. Your program must work correctly with any valid executables, command line arguments, and files we choose to use for grading, which might be different than these examples.

```
./lab12.out head -4 words.txt -pipe grep r
pointer
reference
```

```
./lab12.out echo -e Your program should handle any number of valid command line arguments. -pipe grep -i P
Your program should handle any number of valid command line arguments.
```

```
./lab12.out sort -r words.txt -pipe uniq -i
value
system call
struct
signal
shell
reference
pointer
PIPE
object
netstat
long
library call
ipc
int
function
double
declaration
assignment
```

```
./lab12.out cat words.txt -pipe wc -l
23
```

```
./lab12.out cat words.txt -pipe sort
assignment
declaration
double
function
int
int
ipc
library call
long
netstat
object
pipe
pipe
PIPE
pointer
reference
shell
signal
signal
struct
```

```
struct
system call
value

./lab12.out ps --version -pipe cat -E
procps-ng version 3.3.10$

./lab12.out uname -pipe cat
Linux
```

1 C Program

1.1 General Requirements

1. Place the files for this assignment in a directory called `lab12`.
2. Your program's source file(s) must compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`).
3. Place your C source code for this assignment in a file named `lab12.c`. See the Examples section to see what your C program should output. Compile your C program using the provided Makefile for this assignment. Do NOT modify any commands, targets, or dependencies that are associated with compiling source code in the provided Makefile for this assignment. If there are any warnings or errors from the compiler, then you should fix them before you submit this assignment to us for grading.
4. Your program cannot contain a memory leak when executed. You are responsible for using valgrind to test your program to ensure it does not contain a memory leak, and valgrind should be run for every test case for your program. A program that contains one or more memory leaks (definitely, indirectly, or possibly lost) identified by valgrind will lose points.
5. The only include directives that you are authorized to use in your source code for this assignment are listed below.

   ```
   #include <stdio.h>
   #include <string.h>
   #include <stdbool.h>
   #include <sys/wait.h>
   #include <unistd.h>
   ```

   You CANNOT use any other include directives anywhere in your source code.
6. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

1.2 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.
4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
5. Every line of source code must be indented properly and consistently.

1.3 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Lab 12. Submit only the following files.

1. All source files required for this lab: lab12.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own copy of the provided Makefile to compile your program. We only need the files mentioned above with the file extensions aforementioned.

3 Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

4 Grading: 7 points

If your program does not compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`), then you'll receive a grade of zero on this part of the assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin or a vcf node	7 points
README.txt file missing or incorrect in some way	-1 point
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
One or more compiler warnings	-2 points
Not adhering to one or more coding style requirements	-2 points
Valgrind identifies a memory leak (definitely, indirectly, or possibly lost)	-2 points
Not submitting this assignment before its late period expires	-7 points
Source code contains one | character or more than one | characters	-7 points
Penalty for using a call to `system`	-7 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin or a vcf node with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.