IPC Using Pipes and Signals

Program Four
CS 3411 Spring 2022

Overview

You are required to create a system of four communicating concurrent processes which does some relatively lightweight manipulation of text files. The goal of this assignment is to give you realistic experience with process creation, signals and IPC through pipes. The system is comprised of four processes: the driver, the scanner and even and odd. These are described in more detail below.

Scanner

The *scanner* process accepts ascii data on its standard input and detects *words*. A word is defined to be any consecutive sequence of non-whitespace characters, where a whitespace character is a SPACE, a TAB, or a NEWLINE. The scanner then transforms words in accordance with two rules:

- all punctuation characters (as defined by the `ispunct()` library routine) are stripped out of the word, and
- all characters in the word are mapped to lower case (as defined by the `tolower()` library facility).

When the scanner detects a word which contains an even number of characters, it sends that word down a pipe to a process called *even*. It then terminates that word by sending a space down the pipe to even. It performs analogously when a word containing an odd number of characters is detected—the only difference is that a process called *odd* gets the space-terminated word down its pipe.

On detecting EOF on the standard input, the scanner will close its output pipes and enter an infinite loop in which it sleeps for 1 second and then prints a single asterisk on its standard output. When it receives a SIGTERM signal from even, it enters a phase in which it expects to receive the output of even and odd (see below) and to display that information to the user by writing onto its standard output. The final output might appear as:

```
**********************************
Words with even letters:
   farp     2
   twit     6
Words with odd letters:
   fubar    1
   zip      8
```

Even and Odd

The processes even and odd expect to have standard input which appears as

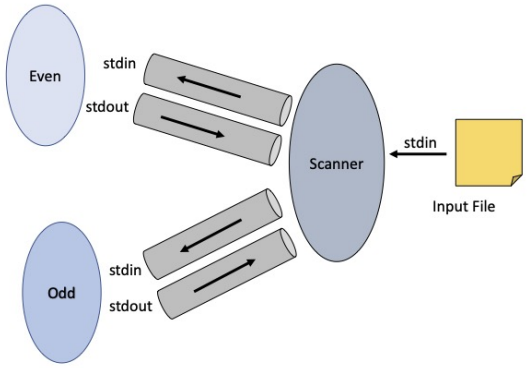
word word . . . word word

Each constructs a table of (word,count) pairs, updating on incoming words. When **even** detects EOF on its **standard input**, it will wait for 10 seconds, signal the scanner as specified above, and will then write (to its **standard output**) a sequence of (word,count) pairs. The process called even will do exactly the same as odd, **but it will not signal the scanner**. In the output generated by even and odd, the words will be null-terminated character sequences, and the counts will be *binary* integers. The scanner receives and outputs those pairs in human-readable form. It is imperative that the standard output of even and odd be hooked up to the appropriate pipes which feed the scanner during its output phase.

Driver

The *driver* is the process responsible for creating and interconnecting the other three processes—it's the ultimate ancestor in this system.

The driver is responsible for taking a single command line argument which specifies the file to be processed by this system. If that file can be opened, it must then create the three sub-processes: scanner, even, and odd. It must arrange for the standard input of scanner to come from the designated file. The standard output of the scanner should be inherited from the driver. The driver must also arrange for the pipes which interconnect the processes in keeping with the figure below. The driver is not shown in the figure because it has no real role in processing the data; it must, however set up all components of the system.



Additional Requirements

You may not make assumptions about the size of the input file. This means that you should dynamically "grow" your data structures as input is processed.

Notes

- Be sure to use low level (binary) input output operations when having the processes interact via the pipe(s). Specifically, you should be invoking the kernel entries `read` and `write` instead of things like `putc` and `fscanf` in order to do pipe I/O. You may communicate with the user (i.e, stdin and stdout of the scanner) using whatever I/O constructs you choose (but you would be wise to use the standard I/O library). The `fscanf` function does a particularly nice job of picking out the words in a text file.
- EOF is never detected on a pipe unless the writer closes its write descriptor for that pipe. Until the input side is closed, a read will hang indefinitely.
- Be sure to terminate this multi-task system gracefully. You should formally `exit` in the three children of driver, and driver should formally `wait` for their termination.
- You must submit three source files. The source for driver should be called `driver.c`; the scanner, `scanner.c`. One program, `evenodd.c` should be written to be the even and odd children. Have the process use the value of `argv[0]` to adapt it's personality to be even or odd: the only real difference in behavior is that even will signal the scanner while odd will not. In the directory where the binaries of the system will reside, there should be binaries for driver, scanner, and evenodd. There should be a link from evenodd to even, and another link from evenodd to odd in order to provide the illusion of separate binaries for the even and odd processes.

Collaboration

You may work in teams of two on this project. It is allowed to work alone, but if you choose this freely, then no allowance will be made in the project requirements or due date. Get in touch with me by March 16 at 5pm if you want a partner but cannot find one. **If you don't contact me by this time, you have freely chosen to work alone.**

Due

No later than **11pm on Friday April 1**.

Submit three files named `driver.c`, `scanner.c`, and `evenodd.c` using Canvas. Submit a makefile that produces `driver`, `scanner`, and `evenodd` binaries. Your makefile should support a directive `clean` that removes all binary and object code. (The makefile should assume the code it operates against is in the same directory as the makefile.). The grader will run `driver` to test your code.

Also submit a README that indicate the number of slip days that were used and describes the contribution of each person in teams of two.