Lab 3

5 February 2021

v20210201-1200

This week's lab is about practicing three things in C: file I/O with `FILE*`, fork/wait, and signal handling.

Black box spec

Your program will take as its command line argument a partial filename, to which the program will append `.in` and `.out` to generate the actual filenames used for the file-based I/O. (You may print an error message, but should not crash, if the provided partial filename is longer than 20 characters.) On startup, it will print a message to the screen, then create one child process to interact with the user and another to interact with the provided files.

For both the user interaction and the file interaction, the behaviour should be that it repeatedly reads in a single positive integer, pauses to think for one second, and then prints the number twice as large as that. If the input is zero, the interaction ends (a "normal" end). If the input is negative, the interaction prints an error message and ends (an "error" end).

If both interactions end normally, the original process prints a message to that effect and exits. If either interaction ends with an error while the other is still running, the still-running interaction process immediately prints that the other interaction had an error and then exits itself; and then the original process prints a message that says which interaction ended with an error, and exits. If the error happens after the other process is done, the original process simply prints its error message and exits.

Example

Contents of `test1.in`:

```
3
42
7
0
```

Interaction, including initial command prompt and both input and output:

```
shannon -> ./a.out test1
Welcome
8
16
1000
2000
17
34
2401
4802
0
Both interactions normal
```

Contents of `test1.out` after execution:

```
6
84
14
```

Example 2

Contents of `errex2.in`:

```
8
13
-5
2
0
```

Interaction, including initial command prompt and both input and output:

```
shannon -> ./a.out errex2
Welcome
123
246
2
Error in other interaction
File interaction ended with error
```

(Note that the exact result/length of the interaction depends slightly on how fast/slow the user types.) Contents of `errex2.out` after execution:

```
16
26
Error
```

Example 3

Contents of `third.in`:

```
1
2
3
4
5
0
```

Interaction, including initial command prompt and both input and output:

```
shannon -> ./a.out third
Welcome
25
50
-1123
Error
User interaction ended with error
```

Contents of `third.out` after execution:

```
2
4
Error in other interaction
```

(Note that the exact contents of the file depend slightly on how fast/slow the user types.)

Internal and other requirements

In case it's not clear, the one-second pause is a purely artificial requirement to make this basically testable (otherwise the file-based interaction would go much too fast). This can be done with the `sleep` function.

The indication from a child process as to whether it is ending normally or with an error should be through its exit condition: 0 for normal, 1 for an error. The original process will check for this condition to decide how to respond to it.

The signal from the parent process to the other child after one child errors should be a SIGTERM which is handled appropriately.

The actual work of read-pause-print should be done by a function that takes `FILE*` parameters, so that it can work with actual files or with standard input and output.

Other design aspects are largely up to you, but you should use functions appropriately and observe principles of good design.

As before, your handin should include a readme documenting how to compile, run, and test your program. You may use a makefile; you should not rely on `compile`. (If you wish to use unci, i.e. `.u` files, to test any functions you write, you can use `uncic` directly to compile the `.u` files into `.o` files. See me for details if you need help.)

Programs that do not compile will get few or zero points.

Use test cases to show me what works.

Duedate and handin

The lab is due Friday at 4pm.

Hand it in using the handin script:

```
handin cmsc242 lab3 dir_of_stuff/
```