CSCI 315 Lab 2 Exercise

February 4, 2010

Last week you investigated the properties of the `fork` command and found that when a process forks a child, the child process starts out with a copy of the parent's environment (code and data). You should also have learned that once the child is created, then there is no communication with its parent. This week's lab is in two parts. First, you will investigate how the `fork` command affects the child in regards to file access. Second, you will investigate the use of the `pipe` command for communication between processes.

Note: Files are also available in ~cs315/Labs10/Lab02.

References: UNIX `man` pages for the `fork` and `pipe` and the pipes handout.

Lab Problems
Write your answers in a hand-in file as you go through the exercises. Copy-and-paste code and output where possible.

Fork

1. Copy this code to a file (call it `fork.cc`). In addition, copy the file text.txt to your directory to use as an input file. [Note: You should be able to use the copy-and-paste method to copy the above files to your editor window.] Compile and run the program using text.txt as the command-line argument.

2. Modify `fork.cc` so that the program will fork two children. Have each child call `ReadFile` with a unique letter identifier and the same file descriptor. Adjust the delay loop to insure that the siblings read no more than 10-20 characters in a time-slice. **Include your modified program in your handin file. What conclusion about how the two programs access the file do you draw from the output of the program? Write your answer in the handin file along with enough of the output to make your point (e.g., show 2 turns for each process).**

3. Modify the program so that one sibling closes the file after about 20 characters are read. **What is the effect on the other process? Include your answer in the handin file.**

4. Modify the program so that the file is opened in `ReadFile` rather than in the parent code. **In the handin file, include the new version of your code. Run your program and explain what happens in the light of the output. Contrast this with the results of the previous experiments.**

Pipes

1. This program creates two sibling processes; copy it to a file (call it `pipes.cc`).

   You are to use this program as a base: be sure to read the pipes handout and study the first three diagrams and the accompanying programs. Add code to `pipes.cc` to create a pipe from the first child to the second.

   The siblings should do the following: the first child should write a character string (your name) *one character at a time* to the pipe. The first child will then generate an end-of-file (EOF) on the pipe by closing the pipe. The second child should read the characters from the pipe and print them to the screen until the EOF is read. The following hints will be helpful:

   - The `read()` and `write()` operations both take the following arguments:
     (a) a file descriptor (type `int`; see the pipes handout);
     (b) a pointer to a `char` object (type `char*`); this can indicate either a single character or a string (you'll want the former in this program); and
     (c) the number of bytes to read or write.
   - Since you'll be passing multiple characters, you'll need to put the `read()` and `write()` calls in a loop. The termination of the `write()` loop is easy. The `read()` loop should terminate when EOF is encountered. The `read()` operation returns a positive value if a character is successfully read, and 0 when it fails to read because of the EOF.
   - Open the pipes in the correct place relative to the `fork()` calls; remember that they are a common resource to both children.
   - In each process (parent and both children) **close the ends of the pipes that will not be used** as the handout describes. Remember that you need to close the write end of the pipe in every process where it is open for the EOF to be generated.

   Get the program working and demonstrate that it works.

2. Modify your program so that two pipes are used in the following way: The first child writes to the second child via the first pipe (no change here). The second child reads characters from that first pipe, writes them to the screen, and then converts them to upper case and writes them to the second pipe. When done with its original task, the first child reads the data from the second pipe and displays the results.

   Note that each pipe needs a pair of pipe IDs (an integer array of size two). Also note that when the job is done, all pipes have to be closed or the receiver will think more would be coming from the open pipe. **Important:** remember that the parent process also has access to the pipe, so you will need to close both ends in the parent *after* it has started *both* children. **Copy your output and your revised code into your handin file.**

3. If you complete the preceding exercises before lab is over, begin work on the homework assignment. It involves using pipes and cooperating processes to solve Conway's problem: read a text-file and print it out in lines of 25 characters after squeezing out redundant spaces, which include tabs and new-line characters. **Note** that the homework assignment is to be handed in separately.

Hand in Hand in the answers and code you've included in your handin file by next Monday.