Did You Order this Product? revisited (one last time!)

In this assignment you will implement the product problem using an extension to your client/server paradigm of PA2. The assignment is constructed in two parts. The first part is required. The second part is required for CSCE 851 students and optional for CSCE 451 students (33% bonus). In the first part, you will modify your client/server implementation of the product problem so that the server creates a child process to perform the necessary calculations. The server parent/child processes communicate using Unix pipes. In the second part, you will use multiplexed I/O in the client and the server (an extension of Part II of PA2).

Part I: Multi-threaded Server

This part is a combination of the first and second programming assignments in that it uses the same client-server framework of PA2 to organize its operation, but it differs in that the work is performed by your program from PA1. In many real network-based servers, rather than having the server itself handle the incoming requests (e.g., to solve the product problem, as above), the server process will fork() another process (a worker process), passing it the socket descriptor to be used for communication back to the client process. The role of the server then becomes one of simply opening and accepting connections from clients and then handing the request off to a worker process. In this project we will have the server process maintain communication with the client process, but have it fork off a subordinate process to do the real work.

The work in this case is performed by your implementation of the product problem from PA1. The server creates a slave process to execute your program from PA1 by using fork(), pipe(), and exec(). This means that you will have to look at the use of pipe in the example code (available from the web page) to see how the stdin and stdout of the parent (server) and child (your program from PA1) processes are connected to one another.

You should also take great care to gracefully terminate the server/client connection. You should not leave an open socket or server available for someone else to connect to, and you should not leave the machines cluttered with useless processes. Thus, you should always check your process listing with the ps -u userid command before logging out to make sure you are not leaving processes behind – if you do not know how to do this, please ask.

IMPORTANT NOTE: Make sure that you do not try to read from a pipe that has had its write end closed. This means that the parent should not read from the pipe to the child after it has told
the child to quit. If you do read from an orphaned pipe (one that has had its write end closed),
your process will receive a SIGPIPE signal. The default action for this signal is to terminate your
process. When this happens, your shell will print a "Broken pipe" (or similar) message.

Part II: Multiplexed I/O and Modified Product Problem
Required for CSCE 851 students. 33% bonus for CSCE 451 students!

Create a new solution to PA1 that ignores all cases in which the first integer is 1. Modify your
server program to fork and exec this modified program. Neither the client nor the server is
allowed to examine the input before sending it to the worker process. Thus, the client and server
never know when they will receive results from the worker process.

Both your client and your server will need to multiplex input processing so that they can read
from either stdin or the socket/pipe. The major problem here is not knowing when a formatted
line of output will be ready for display. Thus, I/O must be multiplexed. Another problem is
coordinating the shutdown of all of the processes.

Grading Policy for Programs
The programs you hand in should work correctly and be documented. When you hand in your
programming assignment, you should include:

1. A program listing containing in-line documentation (for each program).

2. A separate (typed) document of approximately two pages describing the overall program
design, a verbal description of "how it works" including the basics of what the system is
doing underneath, and design tradeoffs considered and made. Also describe possible
improvements and extensions to your program (and sketch how they might be made).

3. A description of the tests you ran on your program to convince yourself that it is indeed
correct. Also describe any cases for which your program is known not to work correctly.
This information can be in a separate test document or in your design document.

4. A make file that compiles your program(s).

Please hand in your source files for all parts of this project.

The program should be neatly formatted (i.e., easy to read) and structured and documented
according to the guidelines distributed in class. Use the handin program to submit your
program(s) for grading. This is assignment 3. Your grade will be determined as follows:

Program Listing
   works correctly 40%
   in-line documentation 15%
   quality of design 25%
Design Document 15%
Thoroughness of test cases 05%

START EARLY. THIS IS HARDER THAN IT LOOKS!