Scripting a Run

CS3411 Spring 2020
Program Four
Due: April 14, AoE

In this project, you will be developing an equivalent program to Unix *script* command. In order to distinguish it from the Unix version, we'll call ours *prose*.

Prose should execute a given program with the supplied arguments, intercept all output from that program, display the output on its standard output and standard error and also save the outputs in two files. The syntax for the execution of the *prose* program is given by:

`prose <script file name>  <program name> <arguments>`.

In order to accomplish this goal *prose* should fork then exec the target program, and direct the output coming from the standard output of the program to the file `<script file name>.stdout` and to the standard output of *prose*, and the output coming from the standard error of the program to the file `<script file name>.stderr` and to the standard error of *prose*. Both output streams must be translated before they are written to the standard output (error) of prose by displaying any non-printable characters in their ascii hex form in angle brackets. For example, the null character should be displayed as `<00>` on the screen, but should be written as is to the corresponding script file. For your reference, ASCII values between 32 and 126 (inclusive) are printable. Any other byte value should be displayed in angle brackets in hex.

In order to accomplish this goal, there are certain requirements:

1. No stdio routines are permitted, except sprintf.
2. No external libraries can be used.
3. All I/O must be done through kernel read and write functions.
4. Perform all reads and writes in chunks of 256 bytes, using a character buffer of that size.
5. Malloc and its friends are not permitted. Do not dynamically allocate memory for this program.
6. The program MUST use UNNAMED pipes to intercept the traffic.
7. The program MUST use execve kernel call to initiate the argument program.
8. The program MUST use select kernel call.
9. The program cannot fork/exec any program other than the one passed as argument.

I strongly suggest you follow the procedure below to develop this program:

1. Assume you will at least have 5 pts if all your submitted programs compile correctly.
2. Develop a small program called step1.c which opens for output the first argument with the added suffix stdout and writes the remaining arguments into that file and exits (10pts).
3. Make a copy of step1.c called step2.c, then add the capability so that it forks, then execs the target program and correctly passes the rest of the arguments (15pts).
4. Make a copy of step2.c called step3.c, then open the files `<script file name>.stdout` and `<script file name>.stderr`, place these descriptors to the standard output and the standard error of the child before executing the exec call. Executed program's stdout should appear in the first file and its stderr should appear in the second file (30pts).
5. Develop an independent program which uses select kernel call to wait for input from the stdin, and a specified timeout value. When the input becomes available it should read it and write it to the standard output. If the timeout passes it should write *tick* to the standard output. This step is necessary to familiarize yourself with the call. Call this program step4.c (No points - will be discussed in the online meeting).
6. Make a copy of step4.c and call it step5.c. Modify it so that it opens a named pipe called "transit" created using a mkfifo command in your directory. Add functionality so that it displays any input on stdin as well as this descriptor. Executing this program and output directing anything to the named pipe from a different terminal should cause your step5 program to display that output, as well as anything typed in to the standard input (15pts).

7. Make a copy of step5.c and call it step6.c. Modify it so that you introduce two pipe calls, modify the select so that it waits on them and the input sides of these pipes are placed to be the stdout and stderr of the child before it execs (10pts).
8. Make a copy of step6.c and call it prose.c. Modify it so that non-printable characters are appropriately translated before displaying them and the two streams are written to the appropriate output files by the main program.

Grading

We will grade prose.c. If it correctly functions, you will receive 95 pts (+5 for compiling). If it is completely dysfunctional, we will grade the individual steps above for partial grading.

Bonus A prose implementation which creates the output files `<script file name>.stdout` and `<script file name>.stderr` only if the target program has performed a write to that stream receives 5 extra points. Bonus points are awarded if and only if the program receives the full points during the evaluation.

Submission

Use Canvas to submit a tar file named `prog4.tgz` that contains:

1. Your final program source prose.c.
2. A Makefile.
3. Any of the *step* files *step1.c*, *step2.c* etc. If you do not submit any of the steps, you will not be able to get partial credit for it!
4. A README file detailing which steps work.

Your makefile should include "all", and "clean" labels. When I type `make` or `make all` in the directory containing your recovered submission, a binary file named `prose`, as well as the binaries step1, step2, .. step6 should be created. Typing `make clean` should remove all the object files and the created binary `prose`.

In order to tar the files, store all your files into the directory project-4, cd into this directory and use the command:

```
tar -czvf prog4.tgz *
```

Ground Rules and Restrictions

This assignment may be provided with updates as the project goes on. Make sure that you check Canvas for announcements watch the class mailing list.
Your program MUST work on colossus or guardian (or other IT supplied remotely accessible lab machine equivalent). Any program which was not tested on these machines will not be graded. Please contact us ASAP if you have trouble with connections to these remote servers, or need advice on how to program on these servers remotely.
You may not borrow or reuse any code from other resources, and all the code must be your own. In addition, the following rules apply:

1. You may discuss the program with others.
2. You may not bring any printed/written or otherwise recorded material into the discussion with you. (You may not show anyone your code; you may not view the code of anyone else. This includes others both enrolled in the course and others not enrolled in the course.)
3. You may generate recorded material during the discussion, but it must be destroyed immediately afterwards. Don't save copies of anything from the discussion.
4. If you are in doubt about the acceptability of any collaboration activity, I expect you to check with me before engaging in the activity.