University of Pennsylvania / CIS Department
Spring 2012
CIS-505 (Software Systems)
Matt Blaze (blaze at-sign cis.upenn.edu)
Homework #1 due: Wednesday, January 25, 2012 (2359 local time)

Description

In this *individual* (non-group) assignment, you will use the standard UNIX-style system call interface to create and manage processes. Processes creation and management is the most basic form of inter-process communication, and programming in C under Unix/Linux will be an important foundation for software you design and build in the rest of this course.

Submission instructions will be posted shortly. Watch this space.

Note: The submission system is still being set up; please don't try to submit your work until next week.

Part 1 (50% credit)

Using only system calls, write a short C program `quickshell` that runs other programs, much as the standard Unix shell does but more impatiently. Each line of standard input to your `quickshell` should contain the name of a program file to execute. Loop to read each input line (until EOF) and run the program file if it exists, waiting for each child process to exit before starting the next. If the child process finishes within 2 seconds, print the message "Wow, that was fast!" on standard output. If it finishes after 2 seconds but within 5 seconds, print the message "That wasn't very fast" after it terminates. If a child takes more than 5 seconds, print the message "This is taking much too long!" on standard output. If it then finishes before 10 seconds, print the message "What took so long?" on standard output after the process terminates. If the process is still running after 10 seconds, print the message "I've had enough of this!" on standard output and kill the process by sending it a signal. These features should be implemented entirely in the `quickshell` program and not depend on any special features of the child process (other than the time it takes to run).

Each input line to quickshell can be assumed to contain the full path name of a program file to execute (one per line, separated by newline characters). There is no need to parse or process command line arguments or do extensive string processing on the input, although you may need to eliminate the trailing newline character before calling the exec function. You may assume that the standard input is an interactive terminal, and you may limit input lines to some (reasonable) maximum length. (Document any other assumptions you make).

Then write four (simple) programs that run for 1, 3, 7, and 12 seconds for testing your quickshell. Each test program should write a short message to standard output right after it starts, sleep() for the given number of seconds, and then print a final message to standard output before terminating. These programs will serve as your test cases.

Important: You may use *only* UNIX system calls -- those documented in section 2 of the UNIX man pages, plus any of execl(), waitpid(), sleep(), signal(), kill(), exit() and select() (which are not, strictly speaking, system calls on some systems) -- to implement the quickshell. You may *not* use any standard library functions (higher-level functions that are documented in section 3 of the manual, such as printf(), etc).

All potential errors, *especially* those indicated by system call return values, should be handled robustly.

Make especially sure to sensibly handle the case where the called program file doesn't exist. Use good coding practices (including appropriate comments, indentation, functional decomposition, etc).

Test and run your quickshell (with each of your four test programs) on the `speclab` Linux cluster.

Hint: Install a signal handler for an alarm signal and schedule an alarm in the parent after you fork. Don't forget to cancel the alarm(s) if the child terminates before 5 (and 10) seconds and to reset the timer each time you loop to run a new child process.

Part 2 (50% credit)

Give a numbered listing of your `quickshell` source code and write a detailed (English language) description (in your own words) of the program's operation. It should be aimed at a reader familiar with the C language but perhaps not with the UNIX system call interface or process management. Pay particular attention to explaining which processes are running when and when your shell program is and isn't in a blocked state,

This part of the assignment need not be lengthy, but it should be clear; two or three paragraphs is probably sufficient.

Important: Your description should be submitted in *PLAIN ASCII* format. No Word files or other proprietary document formats will be graded.

Course home page: http://www.crypto.com/courses/spring12/cis505/

17 January 2012;