Lab 3: The Shell and more C Review

Objectives

- Understand how processes work on UNIX-type systems
- Practice relevant C system calls to work with processes – fork, wait, exec.
- Get more experience writing C code.
- Review OS concepts.

You may complete this lab individually or with one colleague from this class using pair programming rules (see the course Canvas page).

1 fork - wait – exec (14 pts)

These questions are taken from Chapter 5 of the textbook (http://pages.cs.wisc.edu/~remzi/OSTEP/). From that chapter and your notes from lecture, you should be able to complete the following tasks.

1. **Fork** Write a function that calls `fork()`. Before calling `fork()`, have the main process access a variable (e.g., `x`) and set its value to something (e.g., 100). What value is the variable in the child process? What happens to the variable when both the child and parent change the value of `x`?

2. **Fork and Open** Write a function that opens a file (with the `open()` system call) and then calls `fork()` to create a new process. Can both the child and parent access the file descriptor returned by `open()`? What happens when they are writing to the file concurrently, i.e., at the same time?

3. **Fork and Printing** Write another function using `fork()`. The child process should print "hello", and the parent should print "goodbye". You should try to ensure that the child process prints first. Can you do this without calling `wait()` in the parent?

4. **Fork and Exec** Write a program that calls `fork()` and then calls some form of `exec()` to run the program `/bin/ls`. See if you can try all of the variants of `exec()`, including `execl()`, `execle()`, `execlp()`, `execv()`, `execvp()`, and `execvpe()`. Why do you think there are so many variants of the same basic call?

5. **Wait** Now write a function that uses `wait()` to wait for the child process to finish before continuing in the parent. What does `wait()` return? What happens in you use `wait()` in the child?

6. **Waitpid** Write a slight modification of the previous function, this time using `waitpid()` instead of `wait()`. When would `waitpid()` be useful?

Lab 3: The Shell and more C Review

7. **Standard Out** Write a function that creates a child process, and then in the child process closes standard output (`STDOUT_FILENO`). What happens if the child calls `printf()` to print some output after closing the descriptor?

TIP: In each function, make sure to have the CHILD process exit using the `exit()` system call. (Otherwise, what will happen when that function returns back to main and the main function calls on another of your functions?)

2 Glossary (6 pts)

This is a class-generated glossary which contains terms used in operating systems studies: policy, mechanism, system call, API, file descriptor, interrupt, trap, trap table, stack pointer, kernel mode, kernel stack, standard output.

Your role is to provide definitions for these terms and to supply a reasonably credible technical source (book, upvoted post on a technical forum, respectable online tutorial) to support your definition.

Each glossary entry must contain the term, a one to three sentence self-contained and commonly accepted definition **stated in your own words**, and a citation (hyperlink if possible, or name+author of the book, formal reference style is not required).

For the Glossary assignment, you are allowed to use and copy from any available source, provided that you explain the terms in your own words. Do not include **anything** that you don't understand, do more search and try to make sense of the terms.

Submission

Place all the code you wrote for this assignment in a single source file `lab3.c`, adding comments clearly indicating the question that each function addresses. Include your text answers in the comments as well.

Put the glossary in a word document `glossary.docx` in a table format (term, explanation, citation).

Submit both files to Canvas before the deadline.