Week 7: Problems

Once more, we're back to some interesting problems. This week, following our usual pattern, use the `pementorship/w7_problems`. We again advise running it with:

```
$ sudo docker run -it --cap-add=SYS_PTRACE pementorship/w7_problems
```

We also recommend using `tmux` again, but the image won't launch it by default.

Problem 1

This week, we looked at the `fork()`, `exec()` and `wait()` system calls to start new processes.

1. With this knowledge, implement a simple program that, using the arguments passed in from the terminal, executes a program and waits for said program to finish, and then exits. In essence, you're writing a wrapper for a program. If your program is called `minishell`, you can imagine running:

   ```
   $ ./minishell ls -l
   ```

   Will be equivalent to running

   ```
   $ ls -l
   ```

   If you're not sure how to create a new C program for this, take a look at the `Makefile` file in the home directory of your container.

2. Modify this `minishell` to always redirect standard output to the file `/tmp/output.txt`

   HINT: Look up the `pipe2()` syscall, either online or with `man 2 pipe2`. Take a look at the documentation of `fork()` as well to see what happens to files you open before a `fork()`.

Problem 2

On this problem, we're taking a look a little at what bash does when it executes programs. Startup the container, and open two tmux panes. Attach an `strace` instance on one pane to the bash process running on the other one (you can print the PID of a bash process by running `echo $$`).

1. Run the command `ls -l *.c` and watch the output of strace. Can you identify where bash does the `fork()` (or more accurately, `clone()`), and `wait()`?
2. You'll notice that you won't see any `exec()` syscall. Why's that?
3. Rerun the `ls -l *.c` command, but this time, run `strace` with the `-f` flag. This will also follow any children. Can you find the `exec()` syscall now? Are the arguments what you expected?
4. Now type in the command `echo "Hello world"`. What do you notice that's strange about the system calls here?
5. Let's take a look at how bash searches for binaries it doesn't know the location of already (bash has a cache). With strace still attached with the `-f` flag, run a command that doesn't exist, like `potato`. What happens? How did bash try to find it? What do the syscalls that it uses do?
6. Different shells could implement this search differently. Run `zsh` on one of your panes, and attach `strace` to it (zsh is another shell, like bash, and you can also run `echo $$` to get the PID of zsh). Again, run `potato` and see what happens. How does it search for the binary? What are the advantages and disadvantages of using `zsh`'s technique vs `bash`'s?

Problem 3

Last week we took a look at how to create a daemon process by ignoring `SIGHUP` inside of a program called `my_daemon` we could modify. However, you often don't have access to the source of the program you want to "daemonize" (or re-compiling it is tedious or unnecessary). In this problem, write a C program that, given a program name and arguments, will run it in a way that will ignore `SIGHUP`. We've provided `my_daemon` again for you to test your change. Feel free to use the `daemonize.c` file to get started.

HINT: You might want to take a look at `execpe()` if you want to avoid implementing searching over the `$PATH`. You might also want to take a look at what happens to signal handlers and signal settings (like setting a signal to get ignored with `SIG_IGN`) after any kind of `exec()`.