Week 6: Problems

Now for the fun stuff! For these problems, we've provided the image `pementorship/w6_problems`. We recommend running it as:

```
sudo docker run -it --cap-add=SYS_PTRACE pementorship/w6_problems
```

Mostly so that you can use `strace`. If a question requires a different invocation, we'll let you know in the question.

Most of the images going forwards will not run with `tmux`, but you might find it useful or necessary to use it to run multiple things at once on your container. You can start it by running `tmux` from your shell and working from there. If you need a reminder for how to use it, take a look at this short tutorial.

We've also provided multiple c files to modify. If you want to build then, just run `make` from the home directory. If you modify them, you can rerun `make` and it'll rebuild the files you modified. If you run `make clean`, it'll delete all the binaries, so you can rerun `make` to get everything to rebuild.

Question 1

We've provided a program called `my_daemon` (the source file is `my_daemon.c`. A daemon is a kind of program that runs in the background. One technique for creating daemons (not mentioned in the article) is running the process from a shell and finding a way of making sure the shell doesn't kill the process when it dies. How would you do this? Modify `my_daemon.c` to become a daemon process after the parent process dies. You can test your solution by running `my_daemon` from one of your tmux panes, closing it, and on a second pane, running `tail -f /tmp/results1.txt`. If your solution works, you should see your daemon process still writing to the file after the shell it was running under is gone.

HINT: Take a look at ex1. There's some useful discussion about the relevant signal(s) there.

Question 2

We've provided a small python program called `signal_barrage.py`. It will send the signal `SIGUSR1` (a signal for miscellaneous, user-defined purposes) 1,000 times at a process with a given PID. We'll be using this on a program you create. For this question:

1. Implement a program that, every time it receives a `SIGUSR1`, will increment a counter and tell you how many times it's received the signal. You can either implement this in the `signal_reader.c` file, or write a python file that does the same.
    1. Run your program and point `./signal_barrage.py` to it (use the `--help` to see how it works, but only pass in the PID parameter). Take a look at the output of your program. How many times did it receive the signal? It should be lower than 1,000. Can you figure out why?
2. Modify your program so that, every time it receives a signal, it opens "/tmp/results2.txt", writes a new line with the current timestamp, and closes the file. What happens when you do this?