Week 3: Problems

Setup

For most of these questions, we'll be using the image `pementorship/w3_problems` unless the question says otherwise. As one last reminder, you can start this up in docker using:

```
$ docker run  --cap-add=SYS_PTRACE -it pementorship/w3_problems
```

NOTE: DON'T SKIP AHEAD. Read one question at a time, as the next question and section might have answers to the previous one. You'll find it more interesting if you read one question at a time!

Question 1

For this problem, we'll need to start the container in a slightly different way. Instead, you need to run:

```
$ docker run -it -p 8080:8080 pementorship/w3_problems
```

Once you start it, run `./start.sh`, which will start up a simple flask web server (flask is a very simple web server library/framework for python). You can see the source code in `server.py`.

The way we started docker made it so that the network port 8080 on your computer forwards to port 8080 on the container.

1. Go to http://localhost:8080/ on the computer you started the container in. What do you see? What if you go to other URLs on that site like http://localhost:8080/random?
2. What's the PID (process ID) of the flask process?

   HINT: Run `top` to look at a list of running processes in a nice interface. You can quit this by pressing `q`.
3. Attach strace to the process by running `sudo strace -p $PID` (where `$PID` is the PID of the flask process). Don't open the site on your browser. What's flask doing when no one's sending any HTTP requests?
4. In the last question, you should have seen a lot of `poll()` syscalls. What does that syscall do? Why would it be useful for a webserver to call that intermittently?

   HINT: Use `man 2 poll` and search the internet for how TCP sockets are established.
5. Next up, let's see what happens when you request the site. Open a browser again to http://localhost:8080/. What happens in your `strace` invocation when you load the site? What syscalls do you see?
   1. What happened to `poll()` when you sent the request?
   2. Interestingly, you won't be able to find a syscall sending a message back over the network. Can you figure out why? Are there any syscalls there that could be making it so that you can't see the message being sent back?

As mentioned in the last question, you won't be able to see the syscall that sends back the HTTP response to your browser. Let's take a quick trip through what a standard socket connection looks like in Linux. There's basically 4 syscalls: `poll()`, `accept()`, `recv()`, and `send()`. Once you've set your socket connection up on a given port (here using the TCP protocol), you have your server continuously running `poll()`, which will wait on the socket for a message saying there's a new incoming connection. Once one's received, you run `accept()` on that incoming connection to tell the OS that you're ready to start the socket connection. Finally, we can use `recv()` and `send()` on that connection to send data back and forth.

To handle every connection separately, flask starts up a new child process that runs the `recv()` and `send()` syscalls for a given connection. This is the `clone()` syscall you're seeing. Now that we know that

6. Rerun `strace`, but add the `-f` flag at the end. This will also trace child processes. Can you identify the syscalls that are being used to send the data back and forth between the client and the server?
7. In the previous command, you'll see the `sendto()` syscall called multiple times. What is each call used for?