Lab 1: I'm Going to Need That T.P.S Report `1.0.3`

ECE 353: Systems Software
Jonathan Eyolfson
January 16, 2023 **Due: January 30, 2023 @ 11:59 PM ET**

In this lab you'll create a small utility called *Toronto ProcesseS* (`tps`) that prints out the currently running processes on the machine. Your implementation uses the same mechanisms as utilities like `ps` and `top` (you may want to use these to test your program). This is the Linux version of Windows *Task Manager* and macOS *Activity Monitor*. You'll be using Git to submit your work and save your progress.

Development environment. The labs in this course are specific to Linux, so you will have to setup a Linux development environment. The recommended way is to use a Dev Container inside of VSCode. First, install VSCode from `https://code.visualstudio.com`.

After you install VSCode, please visit `https://code.visualstudio.com/docs/devcontainers/containers` and install Docker as outlined in the requirements. We'll be using dev containers for this course. If you already have a Linux development environment, you should be able to use that, but the dev container is the consistent environment we'll use in this course, so be sure to test on it just in case.

After you've installed VSCode and Docker, we'll just need to clone your repository. Just make sure you install Git first.

Additional Windows setup. If you're using Windows, the following steps may help you install Docker:

1. Press the start button and seach for "Terminal"
2. Right click the first result and select "Run as administrator"
3. Type `wsl --set-default-version 2`
4. If you see a message about something missing, follow the link to `https://aka.ms/wsl2kernel`
   Install the first link and restart
5. Press the start button and search for "Microsoft Store"
6. Search for "Ubuntu 22.04", and click "Get"
7. Type "Open", let it install, and type a username and password you'll remember
   (it's just for a virtual machine, for reference I just used jon/jon)
8. Restart machine
9. Install Docker desktop
   See `https://docs.docker.com/desktop/install/windows-install/` for details

You'll need to have Docker desktop running in the background when working on the lab.

Other development environments. You may develop these labs without using the dev container as long as you're on a relatively up-to-date Linux distribution. For this lab, in addition to a C compiler, you'll need: `meson`, `ninja`, and `python3` (at least 3.8).

You may still use the dev container on any Linux distribution. The idea behind the dev container is to give everyone a consistent environment with the exact same software, regardless of operating system. Docker is built on top of Linux, and is very lightweight if you're using Linux already. On Windows and macOS, Docker creates a Linux virtual machine for you behind the scenes.

Git setup. First, make sure you login and can access `https://laforge.eecg.utoronto.ca/`. When you set your name and email it should match your profile on this GitLab instance. Next, open VSCode and open a terminal (Terminal → New Terminal). Run all these commands in whichever directory you'd like on your machine. Note: if you're unfamiliar with Git, please check out the Pro Git book. For any of the commands, run them in the terminal.

1. Run: `git config --global user.name "Your Full Name"`
2. Run: `git config --global user.email your@email.com`
3. Run: `ssh-keygen -o`
   (a) Press *Enter* for the default location
   (b) Press *Enter* for no passphrase
   (c) Press *Enter* again to confirm
4. Login to GitLab
5. Click your username in the top right
6. Click *Edit profile*
7. Click *SSH Keys* on the left
8. Add your SSH key
   (a) Run: `cat ~/.ssh/id_rsa.pub`
   (b) Copy and paste the contents into the text box
   (c) (Optional) Give the key a comment (it'll be its name)
   (d) Press *Add key*
9. Run: `git clone git@laforge.eecg.utoronto.ca:ece353/2023-winter/USERNAME/labs ece353-labs` (replace `USERNAME` with your username)
10. Run: `cd ece353-labs`
11. Run: `git remote add upstream git@laforge.eecg.utoronto.ca:ece353/2023-winter/student/labs`

Lab setup. Ensure you're in the repository (`cd ~/ece353-labs`) directory. Make sure you have the latest skeleton code from us by running: `git pull upstream main`. You can finally run: `cd tps` to begin the lab.

It's very important to open the `ece353-labs` directory in VSCode, not the `tps` directory. Click File → Open Folder, navigate to `ece353-labs` and hit "Select".

VSCode setup. Now that you're in VSCode go to Settings → Extensions and download Dev Containers. After installing click the green button in the bottom left and click *Reopen in Container*. The first time will take a bit to setup, but you should only have to do this once. Once in the dev container, it's recommended to install the `clangd` extension.

Your task. You should output the PID and name of every running process on your machine (or dev container if you're using the recommended setup). Your output should be exactly the same as if you ran `ps -eo pid:5,ucmd` (except for the PID and name of your process). You should start by printing a header, which will be `PID` right-justifeid with a width of 5 characters, a space, then `CMD`. To do this, your implementation needs to read the `/proc` directory and its contents. By default the directories should be in the order of ascending pid. Any directory within `/proc` that's a number (e.g. 1) represents a process with that pid. After that you should read the `/proc/<pid>/status` file to get it's name. The file starts with `Name:` followed by a tab character, then the name (anything that isn't a newline character), and finally followed by a newline character (not part of the name). For each process you should output the pid, right-justified with a width of 5 characters, a space, then the name.

You need to check for errors, and properly close all directories and file descriptors.

Building. First, make sure you're in the `tps` directory if you're not already. After, run the following commands:

```
meson setup build
meson compile -C build
```

Whenever you make changes, you can run the compile command again. You should only need to run setup once.

Testing. After building, you can run your program with `build/tps` and compare your output to running `ps -eo pid:5,ucmd`. You may also choose to run the test suite provided with the command:

```
meson test -C build
```

If you fail a test, you should be able to read the log file given at the end of the output for a reason why the test failed.

If you would like to run a test directly, run:

```
tests/ps_compare.py build/tps
```

After building your program. You should be able to see other tests in the `tests` directory.

Grading. Run the `./grade.py` script in the directory. This will rebuild your program, run the tests, and give you a grade out of 100 based on your test results. Note that these test cases may not be complete, more may be added before the due date, or there may be hidden test cases. These labs are new, so we may need to change.

Tips. You'll want to read the documentation on some C functions (some are light syscall wrappers). Some header files you'll need to use are provided for you in the skeleton code. You'll need to use the following functions:

```
opendir readdir closedir open read close perror exit
```

You should use all the functions above. You may be able to complete the lab without them, but this lab is short, and you'll be using these again for future labs. It's best to get some experience with them now for the shortest lab.

Submission. Simply push your code using `git push origin main` (or simply `git push`) to submit it. *You need to create your own commits to push, you can use as many as you'd like*. You'll need to use the `git add` and `git commit` commands. You may push as many commits as you want, your latest commit that modifies the lab files counts as your submission. For submission time we will *only* look at the timestamp on our server. We will never use your commit times (or file access times) as proof of submission, only when you push your code to the course Git server.