CS 111: Operating System Principles
Lab 0

# A Kernel Seedling 3.0.0

Rustem Can Aygun, Yadi Cao, Victor Zhang
Derivative document by: Jonathan Eyolfson
April 5, 2023
Due: April 12, 2023 @ 11:55 PM PT

In this lab, you'll setup a virtual machine and write your (probably) first kernel module. We'll use VirtualBox as our hypervisor since it supports many different host operating systems, and is friendly to learn. Finally, you'll write a kernel module that adds a file to `/proc/` to expose internal kernel information.

**Virtual machine setup.** After the setup you'll have a fully functioning Linux virtual machine. You're free to edit your files with whatever you're comfortable with, e.g., `emacs`, `vim`. You should only run your code on the virtual machine. Note: If you're using a M1/M2 Macbook, Virtualbox has only recently provided a beta version for that platform, you can try it or (the lengthier) M1 Virtual Machine Setup Guide.

1. Download and install VirtualBox 7: https://www.virtualbox.org/wiki/Downloads
2. Download our virtual machine: https://ucla.box.com/s/8fx06w3rwn8au0h04rhc3opho0zux343.
   If your computer is an old model, use the lightweight version of the vm https://ucla.box.com/s/arvh779n3ggchsekj5nxclyna0iwk2ce
   For the lightweight vm, first login through the terminal, then run `startx` command to start the desktop environment if you are not comfortable with the command line interface.
3. Import the virtual machine
   (a) *File* → *Import Appliance*
   (b) Choose `vm.ova` from your local file system
   (c) *Next* → *Import*
4. Select *CS 111* from the left panel and click *Start* at the top of the right panel
5. Use `cs111` for both the username and password
6. (Optional) Go to *View* → *Virtual Screen 1* and resize to any resolution you'd like
7. Note: Using the power off option might cause errors in your git repo (in case you want to use git for backup purposes). Use graceful shutdown/reboot options instead. You can also use 'save-state' option.

**Your task.** You're going to create a `/proc/count` file that shows the current number of running processes. The process table runs within kernel mode, so to access it you'll need to write a kernel module that runs in kernel mode. For the coding part, you'll modify `proc_count.c`, and only this file. In the `lab0` directory we should be able to run the following commands:

```
make
sudo insmod proc_count.ko
cat /proc/count
```

The last command should report a single integer representing the number of processes running on the machine. Your final task is to fill in your documentation in the `README.md` (also, please replace the number on the first line with your UID).

**Tips.** The kernel code is well commented, you can use https://elixir.bootlin.com/ for looking up functions and macros (symbols). There's already a skeleton that uses: `MODULE_AUTHOR`, `MODULE_DESCRIPTION`, `MODULE_LICENSE`, `module_init`, `module_exit`, and `pr_info`. You'll probably want to use the following to complete this lab:

```
proc_create_single
proc_remove
for_each_process
```

```
seq_printf
```

You can divide this task into small subtasks:

1. Properly create and remove `/proc/count` when your module loads and unloads, respectively.
2. Make `/proc/count` return some string when you `cat /proc/count`
3. Make `/proc/count` return a integer with the number of running processes when you `cat /proc/count`

**Commands.** You'll have to use the following commands for this lab:

Build your module with `make`
Insert your module into the kernel with `sudo insmod proc_count.ko`
Read any information messages printed in the kernel with `sudo dmesg -l info`
Remove your module from the kernel (so you can insert an updated one) with `sudo rmmod proc_count`
Sanity check your module information with `modinfo proc_count.ko`

**Testing.** There are a set of basic test cases given to you. For this lab the provided test cases are likely the ones we'll use for grading. In the future we'll withhold more advanced tests which we'll use for grading. Part of programming is coming up with tests yourself. To run the provided test cases please run the following command in your lab directory:

```
python -m unittest
```

**Grading.** The breakdown is as follows:

75% code implementation in `proc_count.c`
25% documentation in `README.md`
3% for *extra credit*, your notes from the Week 1 discussion session (we will discussion this lab then)

**Submission.**

- All lab submissions will take place on BruinLearn. You will find submission links for all labs under the Assignments page.
- Your submission should be a single tarball that includes two files: `proc_count.c` and `README.md`. The tarball should be named `YOURUID-lab0-submission.tar`, where `YOURUID` is your 9 digit UID without any separators. You can create this tarball by running the command `make tar` inside your `lab0` directory (note: running `make tar` will overwrite the previously created tarball). Any submission that does not follow the submission guideline will receive -15 pts. (Note: if you make multiple submissions, Bruinlearn will automatically append the filename with `-SOMENUMBER`, you don't need to worry about this.)
- If you attended Week 1 discussion session, submit a copy of your notes for a small extra credit. You can submit your notes as image, pdf, or plain text file.
- Your submission will be graded solely based on your last submission to BruinLearn, without any exceptions. Please double check your submission to make sure you submit the correct version of your implementation.