CS 111: Operating System Principles
Lab 0

A Kernel Seedling 3.0.0

Can Aygun, Alexandre Tiard, Vishnu Vardhan Bachupally
Derivative document by: Jonathan Eyolfson
March 31, 2022
Due: April 8, 2022 @ 11:59 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. You'll be using Git to submit your work and save your progress. 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. For example, you can install VSCode with: `sudo pacman -S code`. You may replace `code` with `emacs` or `vim` as well. You should only run your code on the virtual machine. Note:If you are using M1 Macbook follow the m1 virtual machine setup guide (alternative link)

1. Download and install VirtualBox 6.1.26: `https://www.virtualbox.org/wiki/Downloads`
2. Download our virtual machine: `https://sefer.cs.ucla.edu/media/cs111/vm.ova`, (alternative link)
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. Use graceful shutdown/reboot options instead. You can also use 'save-state' option.

Git setup. Run all these commands in your home directory (or anywhere really) on your virtual machine. First, open a terminal by going to *Activities* and selecting the *Terminal* icon on the left. 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 the course website
   (a) (Optional) Go to *Activities* and click the *Firefox* icon on the left
5. Click your username in the top right
6. Click *New SSH Key (Text Input)*
7. 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 submit
8. Run: `cd ~`
9. Run: `git clone git@sefer.cs.ucla.edu:spring22/USERNAME/cs111`
   (replace `USERNAME` with your username)
10. Run: `cd cs111`

11. Run: `git checkout main` , if the main branch does not exist, Run: `git checkout -b main`
12. Run: `git remote add upstream git@sefer.cs.ucla.edu:spring22/cs111`
    (do not change this)

Lab Setup. Ensure you're in the repository (`cd ~/cs111`) directory. Make sure you have the latest skeleton code from us by running: `git pull upstream main`. You can finally run: `cd lab0` to begin the lab.

Your task. You're going to create a `/proc/count` file that shows the current number of running processes (or tasks) running. 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 your submission you'll modify `proc_count.c`, and only this file, for the coding part. 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 (or tasks) running on the machine. Your final task is to fill in your documentation in the `README.md` for `lab0`.

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 (or tasks) 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 a new 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`

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 late days we will 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.

We've created a new system that double checks you have the latest upstream code and have submitted something. Please check `https://sefer.cs.ucla.edu/cs111/grades/` to see your status. You're expected to properly merge in upstream code without rebasing. Note that the website only updates your lab modification status if you've merged the latest code.

Late Days. You have 4 late days for the entire quarter. There will be no penalty for using late days. If you use 5 late days, your lowest lab grade will be halved. Any additional late days will cause you to receive a 0 on your latest lab and you will recoup the late days. We will try to give you the option that results in the highest grade.