## Lab 5: Virtual Memory Simulator `1.0.1`

ECE 353: Systems Software
Jonathan Eyolfson
March 13, 2023 **Due: March 27, 2023 @ 11:59 PM ET**

In this lab you'll be managing page tables and simulating what happens when a process forks. You'll be given code that simulates the MMU, and code that can modify page table entries (PTEs) without low-level C code. You'll implement two fork strategies: one that creates a new page table and copies all page tables, and another that creates a new page table and uses a copy-on-write optimization. We'll be using the Sv39 multi-level page table design as discussed in the lectures.

**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`.

This will create a merge, which you should be able to do cleanly. If you don't know how to do this read Pro Git. **Be sure to read chapter 3.1 and 3.2 fully.** This is how software developers coordinate and work together in large projects. For this course, you should always merge to maintain proper history between yourself and the provided repository. **You should never rebase in this course, and in general you should never rebase unless you have a good reason to.** It will be important to keep your code up-to-date during this lab as the test cases may change with your help.

You can finally run: `cd vms` to begin the lab.

**Building.** First, make sure you're in the `vms` 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.

**Starting the lab.** After you build you'll have a `build/vms` executable. The source code for this executable is in `src/main.c` and is meant for you to test your code and experiment easily. It sets up page tables such that virtual address `0xABC123` is valid. You should read `src/main.c` and understand the library functions it uses from `include/vms.h`.

**Files to modify.** You should only be modifying `src/vms.c`.

**Your task.** You'll have to implement two library functions in `vms.h`: first `vms_fork_copy()` and then `vms_fork_copy_on_write()`. Both functions should return a new L2 page table that's set up as if the currently running process calls `fork` (you should call `vms_get_root_page_table()`).

`vms_fork_copy_on_write()` is independent of the first, and will be trickier to get correct. You'll have to take advantage of handling page faults through implementing `page_fault_handler` in `src/vms.c`. In this function you may modify the faulting PTE and make it valid. After the MMU generates a page fault, it will re-try the operation once. If the MMU generates a page fault again, your process will exit with the `EFAULT` error code. You'll also be able to use a bit in the PTE for whatever purpose you would like. You can access it through `vms_pte_custom` (it also has clear and set like the other bits). Finally, you'll want to keep track of the number of references to some pages. You may take advantage of the defined values in `pages.h` and that the system will not create more pages (assume it represents physical memory which does not change).

**Testing.** You cannot execute your library directly, however you can run the test programs manually. Please find the files in `tests/*.c`. You should be able to read and understand what they're doing with your library. Find the executables in `build/tests/*`.

You may also choose to run the test suite provided with the command:

```
meson test --print-errorlogs -C build
```

**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.

**Errors.** You need to check for and properly handle errors. For fatal errors, you should exit with the `errno` of the first fatal error. However, your implementation should not generate errors

**Tips.** Carefully review the multi-level page table lecture, and draw a diagram of how the page tables look for the example given to you in `src/main.c`. You should be able to make any virtual address valid to read and write to.

Aside from what's given in the description of the task, you should have everything except `memcpy`. That should be the only library function you need. Otherwise, please read `vms.h` and understand `src/main.c`.

**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. 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.