GH repo: https://github.com/evgerritz/cpsc429_labs/tree/2_p4
Steps to run:
On Zoo:
Boot server vm (./vms/boot); this script automatically forwards the VNC and server ports

On Host:
Use VNC to connect to server
Enter movenet_server/
cargo run

On Target:
Connect camera to VM (Checkbox in Devices menu for VirtualBox)
Enter rust_movenet/
Start yuv422 webcam with ./fake_webcam
./tunnel (to connect to the Zoo)
./test_module.sh (gets latest version of rust_camera.ko and inserts into kernel)
./run (starts rust_movenet as root)

Part 4:

- Acknowledgments
    - Resources provided in assignment
    - http://fivelinesofcode.blogspot.com/2014/03/how-to-translate-virtual-to-physical.html
    - https://elixir.bootlin.com/linux/latest/source/arch/csky/include/asm/page.h#L38
    - https://www.unix.com/man-page/suse/9/filp_open
- Main files for this assignment
    - lab2/part4/rust_kmod/custom_modules/rust_camera.rs
    - lab2/part4/rust_movenet/src/main.rs
- Challenges
    - Getting the pfns from pagemap. I didn't know we had to execute the program with root privileges so I spent a while debugging these pfns to no avail.
    - Getting the vfs_ioctl calls to work. These were hard to debug and the lack of the usual Rust memory checking forced me to think through when each object was valid and how the kernel module and userspace program's executions overlapped.

    - Using pointers in Rust was always tricky. Additionally, often there was a memory issue, it caused a segfault in the kernel, and I had to reboot the virtual machine.
    - Setting up the TCP stream. I had to read through a lot of the bindings code as well as the relevant Linux man pages to understand how to use the networking code. I also had to change the visibility of the TcpStream struct's sock member in order to instantiate it directly.
- Overview
    - In userspace, we call mmap to get the userspace virtual addresses for the video buffer, use /proc/self/pagemap to get the corresponding page frame numbers, and then send them as well as our userspace buffer's address to the kernel module.
    - The kernel module then spawns a thread to queue and dequeue buffers for v4l, sending the buffers (accessed through their kernel address) to the server. The server computes the output of the network, sends it back to the kernel module, and the result is stored in a global buffer protected by a mutex.
    - The userspace program periodically obtains the contents of the kernel module's output buffer and displays the output for the user.