Adding Networking Support to CertiKOS

Kevin Abbott
Advisor: Zhong Shao

Overview

The FLINT research group at Yale is currently working on CertiKOS, the Certified Kit Operating System, which is a small operating system kernel that was built in order to give provable guarantees on extensibility, security, and resilience. The kernel was designed by breaking up a complex kernel system into many abstraction layers that can each be certified independently with formal method proofs. In its current state, CertiKOS provides memory and process management and is able to support multiple programs running independently and securely. Since CertiKOS requires a provably correct specification, it is inherently limited in its size and scope of features. As a result, the goal of CertiKOS is to use it as a hypervisor and have traditional operating systems running on top of it. In this setup, users get the benefit of both provable security and the full feature set of commercial operating systems.

One necessary feature currently missing from CertiKOS, though, is a full network stack. Once it is running a network stack, it will be able to support a much broader range of applications and can function as a full-fledged hypervisor. The goal of this project is to add networking support to the CertiKOS kernel.

Network Stack

A kernel network consists of a number of layers. At the bottom of the stack is the physical Network Interface Controller (NIC) which provides the ability to communicate over a network using a specific physical layer and data link layer. In order to communicate with the NIC, a software driver lives in kernel space. The purpose of this driver is to first initialize
communication with the NIC. It does this by initializing the card, typically over PCI, and then setting up the memory-mapped IO (MMIO) region to read/write the NIC’s registers, setting up the inbound and outbound buffers in main memory to allow the NIC to read/write packets via direct memory access (DMA), and setting up the interrupt handlers to know when new packets arrive. After the driver has initialized communication, it then acts as the middleman between the upper levels of the network stack and the NIC by reading/writing packets to the DMA region and multiplexing/de-multiplexing packets to the correct TCP/IP stack. The TCP/IP stack, which lives in either user or kernel space, is responsible for getting data packets from user applications, typically via the sockets API, and adding the correct TCP/UDP and IP headers before sending it to the data link layer. It is also responsible for getting incoming packets from the data link layer and de-multiplexing the packet data to the correct application.

Goals of the Project

In order to add networking support to CertiKOS both a NIC driver and TCP/IP stack have to be added. The goal of this project is to write the driver and add the TCP/IP stack. The goal is not, however, to create provably correct versions of these, because first a working version and formal specification of the network stack must be made.

Since the NIC driver is highly dependent on the specific hardware, this project aims to write an e1000 driver to support the set of Intel e1000 network adapters. The reason for choosing this is because these Intel NICs are very prevalent and it will be possible to test the driver within a QEMU virtual machine, since it provides a virtualized Intel NIC and debugging support. There is a lot of work to be done within the driver, both to add the MMIO and DMA interaction for the network adapter within a CertiKOS context and to add all the features a network driver needs to support, especially to be able to use it within a hypervisor. As a result, the majority of the project
time will be spent within this layer. In order to provide a functional network stack, though, a TCP/IP stack needs to be added; therefore, an existing stack will likely be ported, such as the lightweight lwIP stack.

**Deliverables**

1. **Write NIC Driver:**

   The primary goal of this project is to write the driver to interface with the physical NIC, specifically an Intel e1000 network adapter

2. **Port TCP/IP Stack**

   Once the NIC driver has been written, a lightweight TCP/IP stack can be ported and the system will be able to support networking applications.

3. **Write Formal Specification of Network Stack**

   One of the main challenges of creating a provably correct kernel is splitting it up into small enough abstraction layers. A major goal of this project is determining a formal specification of the network stack that outlines these layers of abstraction.

4. **Create the Framework to be Used in CPSC 422**

   A version of CertiKOS is used for the assignments in CPSC 422: Operating Systems. One lasting goal of this project is to create the framework for a networking assignment.