Porting mCertiKOS to RISC-V
A Senior Project Proposal

Adam Cimpeanu
CPSC 490: Special Projects
Advisor: Professor Zhong Shao
Spring 2016

1. Introduction

1.1 CertiKOS and mCertiKOS

The FLINT group at Yale, which focuses on almost all aspects of building certified systems software, is currently conducting research on an in-house, lightweight operating system kernel called mCertiKOS. Its parent project, CertiKOS (Certified Kit Operating System) was originally designed as a platform for secure cloud computing [1]. Since the completion of this larger verified kernel, the group has streamlined its kernel and verified a smaller version both to decrease development time upon successive iterations and to allow it to run on smaller hardware platforms including embedded systems. Projects like Embassies at Microsoft Research have demonstrated that a number of interesting research questions surface when working with minimal execution models [2].

The smaller version of CertiKOS is called mCertiKOS and currently exists as a set of verified kernels derived from the same base uniprocessor version. Included in this set are the mCertiKOS-hyp kernel, which has been extended to run as a fully-functional hypervisor, mCertiKOS-rz, which supports “ring 0” processes, and mCertiKOS-emb, which has been stripped of virtualization, virtual memory, and user-space interrupt handling for the sole purpose of being used in embedded systems [3].

In addition to these functional variants to mCertiKOS, attempts have been made by the group to rewrite the kernel using other instruction set architectures. The kernel was originally written using a combination of C and x86 assembly, but working versions have been written in AMD and ARM. Extending mCertiKOS to other instruction sets has allowed the kernel to run on other hardware platforms (the lab has experimented with Landshark UGV and the American-Built Car platforms) and the work is also a rich source of research, as different instruction sets make different assumptions about the underlying hardware and each version must be verified separately.

1.2 RISK-V

In 2010, researchers at the University of California Berkeley began the project of building an open-source instruction set architecture. Their design was based on the concept of reduced instruction set computing (RISC) in which a small number of optimized instructions are used instead of a larger, more complex set. Often, what gets sacrificed in specialization and versatility within an instruction set architecture is made up for by dramatic increases in the speed and efficiency of a microprocessor running
that set. While other RISC instruction sets exist (most notably MIPS from Stanford), and other attempts have been made at building open-source instruction sets, what sets RISC-V apart is that it is both open-source and it was designed to keep up with the demands of more serious computing environments, such those seen in cloud and mobile platforms.

With its increasing hardware and software support, RISC-V has begun to fulfill some of the promises of RISC-based designs. The RISC-V Rocket core microprocessor is twice as energy efficient as the Cortex-A5, the most similar ARM implementation [4]. The potential promise of the platform has inspired the R&D division of the Indian government to invest $45 million to design a 64-bit processor based on RISC-V [5]. Many of the potential chipsets to come out of similar work at IIT Madras will be targeted towards IoT applications.

2. Discussion

The largest potential market for secure kernels lies in embedded systems. The FLINT group took a major step towards running on this platform when they rewrote their kernel using ARM and AMD. Rewriting the kernel in a RISC instruction set would seem to be the next logical step, as these reduced sets are ideal for embedded systems, and writing it in RISC-V would open many exciting new doors for the group. If verified, mCertiKOS on RISC-V would be the first verified kernel on this instruction set. Because of the continuing work on RISC-V at UC Berkeley, its BSD Open Source License, and the increased attention it is getting both from research labs in India and major companies like Google, HP, and Oracle, there is a very good chance that this instruction set will take a more central role in the research community in the years to come [6]. This makes it even more of a priority for the FLINT group to investigate ways of integrating RISC-V with their work.

3. Project Design

The goal of this semester project is to write a simplified version of the base uniprocessor model of mCertiKOS that can run on the RISC-V instruction set. The degree of the simplification will be detailed below. As the standard version of mCertiKOS is written in C and x86 assembly, the main work of the project will involve rewriting the portions written in assembly while maintaining the C code as much as possible. Because of the implicit assumptions of different instruction sets, certain lower-level features, such as paging, will likely need to be rewritten. Also, parts of the C code that interface with assembly, such as the context switching layers, will also need to be treated differently.

While the goal will be to eventually run the kernel on a RISC-V chip, development will happen on the QEMU machine emulator.

Porting the entire base uniprocessor model of mCertiKOS with all its functionality to RISC-V is outside the scope of a single semester project. Instead, this project will take the approach of building the kernel in layers, much as one would write a kernel in a class on OS design. Note that these layers also serve as milestones for the project.

3.1 Baremetal Kernel

By this, it is meant that the kernel that can print something on the screen. A boot loader will be needed to start kernel code. The kernel will be as simple as possible, written in assembly and using the easiest way to
send outputs to the screen.

3.2 Transition to C

This layer will have the physical memory layout figured out so that from now on the kernel can be written in C and compiled to fit the desired memory layout. Outputting code in the first milestone must also be revised to make for more sophisticated debugging functions (usually printf), following the complete procedure specified in the hardware manual.

3.3 Trap and Simple System Calls

Trapping is introduced, requiring the kernel to save and restore a process context. On top of this, the first system calls are written to provide the interface for user processes to request kernel services.

3.4 Process Management and Scheduling

With context switches working, process management can be implemented so that this becomes a multiprocessing kernel. User processes typically run in the lowest privilege mode, and they can only request kernel services through system calls.

3.5 Stretch Goals

At this point, the kernel is fairly limited in what it can do, and there are a number of independent ways of adding to its functionality. In no particular order, this list includes adding timer interrupts and preemptive multiprocessing, virtual memory management, floating point support, and inter process communication.

4. Deliverables

The final deliverables for this project will be both the code of the miniature kernel, hopefully written up to layer 3.4, and a final report on the exact ways in which mCertiKOS needed to be modified in order accommodate the new instruction set architecture.

References