Tutorial: Getting Started With seL4 on RISC-V
With Renode’s Help
Jesse Millwood
Embedded Engineer
DornerWorks
jesse.millwood@dornerworks.com
Jesse Millwood

• Embedded Engineer
• DornerWorks
• Advanced Embedded Platform Group

• Objectives
  • seL4 Introduction
  • Using seL4 Test
  • Developing with CAmkES
  • Emulating with Renode

Secure Your System with seL4

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
Why Use seL4?

• It is a microkernel
• It is backed by formal methods
• It is Open Source
• It focuses on security
seL4: Microkernel

• Microkernel Architecture
  • Very small kernel
  • Pushes driver logic and OS logic to user space threads

• Main Responsibilities
  • Enforce flow control
  • Enforce separation and access control
    • Requires the address translation as specified by the Supervisor Privilege Mode
  • Scheduling
seL4: Proved Correct

- The core kernel has been expressed in formal specifications
  - Proves:
    - No Overflows
    - No Deadlocks
    - IPC Fastpath gives same results as regular IPC
    - Isolation
    - Capability Access Control
    - Information Flow

- Refinement proofs show that
  - The C code implements only what is specified in the higher level model
  - The binary only implements what is expressed in the C code

- Does not extend to User Space!
seL4: Open Source

- Originally developed by UNSW/NICTA
- Functional Correctness proof completed in 2009
- Open sourced code and proofs in 2014
  - Huge Undertaking
    - ~20 Person years to complete kernel development and reusable proof frameworks
    - ~ 11.5 Person years to design, implement, and verify the kernel
  - GPL’d kernel
  - Data61 userspace libraries are a mix of MIT, BSD, and GPL (no dynamic linking mechanism in user space)
- Development and Maintenance is taken care of by Data61 (Formed by merging of NICTA and CSIRO)
- Kernel and Libraries are there for you to alter to your needs!
  - Upstreaming increases quality
- Your advancement advances everyone!
  - https://dornerworks.com/tag/open
seL4: Secure

• Achieves secure access control by using capabilities for operations on all kernel objects / resources

• Kernel Objects:
  • Untyped Memory
  • VSpaces
  • Cspaces
  • IPC
  • Endpoints/Notification
  • Threads
Sounds Great!

• What can I use it for?
  • seL4 Desktop? Can I check my e-mails?
    • NO, not directly
    • On the horizon?
      • UX/RT https://gitlab.com/uxrt
      • Genode: https://genode.org/index
  • Highly secure static embedded systems?
    • YES
  • No POSIX
    • Possible to implement subsets
System Architecture and Tools

• Architectures
  • Bare seL4 Applications
  • Hypervisor
  • ADL Based

• Tools
  • CAmkES
  • Cogent
  • Rump Kernels
Let’s Build A System

• Data61’s seL4 Test
  • Great for testing seL4 on new platforms
  • Great for getting familiar with the surrounding tools and project layouts
seL4 Test: Platform

• Don’t Have Hardware?
  • Renode
    • Aims to be able to run the same binary as the hardware
    • Supported RISC-V CPUs with supervisor modes:
      • Microchip Polarfire SoC
      • SiFive fu540
seL4 Test: Get The Tools

- Host Build Setup
  - Requires many packages, differing across distributions and versions
  - [https://docs.sel4.systems/HostDependencies](https://docs.sel4.systems/HostDependencies)

- Install Google’s Repo tool
  - [https://gerrit.googlesource.com/git-repo/](https://gerrit.googlesource.com/git-repo/)

- Use Data61 Docker setup
  - Just need docker
    - [https://docs.docker.com/install/linux/docker-ce/ubuntu/](https://docs.docker.com/install/linux/docker-ce/ubuntu/)
  - Everything is packaged up for you
  - Just point it to your build directory
  - Can still use external tools (editors, etc..)
Renode: Install

• Install “mono-complete”
  • https://www.mono-project.com/download/stable/#download-lin

• Install Renode Package from Github Release Page
  • https://github.com/renode/renode/releases/tag/v1.8.2

• Install robot framework python packages (Python 2.7)
  ```
  python -m pip install robotframework==3.0.4
  netifaces
  requests
  psutil
  ```
seL4 Test: Get The Code

• Use the Google Repo Tool
  • Point to an seL4 manifest repository

$ mkdir sel4test
$ cd sel4test
$ repo init -u https://github.com/dornerworks/riscvsummit-manifests -m sel4test.xml
$ repo sync
jessen@ubuntu:~$ ld projets/sel4test
  % repo init -u https://bitbucket.org/riscvsummit/sel4manifests
seL4 Test: Project Layout
seL4 Test: Enter Build Environment

• If using the Docker Environment, Enter it with make
  • Some developers make aliases for it

• My “start-container.sh” script will start the RISC-V specific container
  • By default it doesn’t pull in the RISC-V tools
seL4 Test: Configure The Build

• CMake/Ninja

• Project Root: init-build.sh
  • Pass CMake Variables to this script
    • What Cmake Variables Exist?
      • Look for these files: CMakeLists.txt, settings.cmake, easy-settings.cmake
      • Run “ccmake .” from the build directory

• Make a build directory in the Project Root
  • Run the CMake commands from here
seL4 Test: Build The Code!

• Simply run “ninja” in the build directory
  • Must have a successful CMake Configuration
seL4 Test: Scripting Build

- “run-build.sh” script
  - Provided by my tools in “tools/seL4Extra”
  - Creates a directory based on the platform and architecture
  - Changes directory into it
  - Calls “init-build.sh” with the following arguments:
    - -DPLATFORM=hifive
    - -DSIMULATION=FALSE
    - -DCROSS_COMPILER_PREFIX=riscv64-linux-gnu-
    - -DELfloaderImage=elf
    - -DLibSel4PlatSupportUseDebugPutChar=OFF
    - -DSel4testAllowSettingsOverride=True
    - -DCMAKE_BUILD_TYPE=Debug
    - -DRELEASE=False
    - -DLibSel4TestPrintXML=ON
    - -DRISCV64=TRUE
Renode: Configure Emulation

• Use a Renode Monitor script (*.resc)
  • Provide
    • The seL4 binary
    • The platforms DTB (Gets passed to the bootloader, BBL)
sel4test.resc

1: name sel4 Test
2: description Runs sel4 Test on Renode
3
4: $name?="sel4test"
5
6: using sysbus
7: mach create $name
8: $bin?=@./riscv-build-hifive-64/images/sel4test-driver-image-riscv-hifive
9: $dtb?=@./tools/sel4Extra/renode_support/hifive-renode.dtb
10: machine LoadPlatformDescription @./tools/sel4Extra/renode_support/sifive-fu540-dw.repl
11: showAnalyzer uart0
12
13: macro reset
14: ""
15:  sysbus LoadELF $bin
16:  sysbus LoadFdt $dtb 0x81000000 "earlyconsole mem=256M@0x80000000"
17:  e51 SetRegisterUnsafe 11 0x81000000
18:  u54_1 SetRegisterUnsafe 11 0x81000000
19: ""
20: runMacro $reset
21
seL4 Test: Run The Code

• Pass your Renode Monitor script to the Renode application

• Caveat: Kernel Modification Needed
  • Renode doesn’t handle “sfence” with arguments
  • The kernel provided in this manifest removes the arguments in order to run
seL4 Test: Output

```xml
<testcase classname="sel4test" name="TLS0001">
</testcase>
<testcase classname="sel4test" name="TLS0002">
</testcase>
<testcase classname="sel4test" name="TLS0006">
</testcase>
<testcase classname="sel4test" name="TRIVIAL0000">
</testcase>
<testcase classname="sel4test" name="TRIVIAL0001">
</testcase>
<testcase classname="sel4test" name="TRIVIAL0002">
</testcase>
<testcase classname="sel4test" name="VSPACE0000">
</testcase>
<testcase classname="sel4test" name="VSPACE0002">
</testcase>
<testcase classname="sel4test" name="VSPACE0003">
</testcase>
<testcase classname="sel4test" name="VSPACE0004">
</testcase>
<testcase classname="sel4test" name="VSPACE0005">
</testcase>
<testcase classname="sel4test" name="VSPACE0006">
</testcase>
<testcase classname="sel4test" name="Test all tests ran">
</testcase>
</testsuite>
All is well in the universe
```
Renode Features

• Declaratively Specify Boards, SoC, Connected Environments (RF, Ethernet, etc..)
  • Means no recompilation needed for adding devices or changing cores
  • Heterogeneous system support
• Scripting Interfaces (Python, Renode Monitor, Robot)
• Instruction Level Emulation Granularity
• Virtual Time
• Networking (Ethernet, Wireless, Wireshark)
• Peripherals and Cores in C#
  • Can include custom peripherals at run time
Renode: Robot Framework

- Acceptance Based Testing
- Test inputs and outputs to your emulated system
- Renode integration provides direct interaction with the emulated platform
seL4 Test: Robot Framework

• “run-robot.sh” will run the following script:

```bash
*** Keywords ***
Prepare Machine
  Execute Command using sysbus
  Execute Command mach create "sel4test"
  Execute Command \$bin?@${CURDIR}/.../.../riscv-build-hifive-64/images/sel4test-driver-image-riscv-hifive
  Execute Command \$dtb?@${CURDIR}/hifive-renode.dtb
  Execute Command machine LoadPlatformDescription @${CURDIR}/sifive-fu540-dw.repl
  Execute Command showAnalyzer uart0
  Execute Command sysbus LoadELF $bin
  Execute Command sysbus LoadDtb $dtb 0x81000000 "earlyconsole mem=256M@0x80000000"
  Execute Command e51 SetRegisterUnsafe 11 0x81000000
  Execute Command u54_1 SetRegisterUnsafe 11 0x81000000
```
*** Test Cases ***

**Should Boot seL4**
[Documentation] Boots seL4 on SiFive Freedom U540 platform.
[Tags] sel4 uart

**Prepare Machine**
Create Terminal Tester ${UART}$
Start Emulation

**Wait For Line On Uart**
Wait For Line On Uart bbl loader timeout=120
Wait For Line On Uart ELF-loader started on (HART 1) (NODES 1) timeout=120

**Provides** booted-sel4

**Should Test There Are Tests**
[Documentation] Check for the first test from sel4test
[Tags] sel4 test uart

**Requires** booted-sel4
Wait For Line On Uart <testcase classname="sel4test" name="Test that there are tests"> timeout=9999

**Provides** there-are-tests
seL4 Test: Robot Framework

```bash
[jesse@dwbuntu ~/sel4-projects/sel4test]
% ./run-robot.sh
Preparing suites
Starting suites
Running tools/sel4Extra/renode_support/sel4test.robot
===============================================
sel4test
===============================================
Should Boot seL4 :: Boots seL4 on SiFive Freedom U540 platform. | PASS |
Should Test There Are Tests :: Check for the first test from sel4test | PASS |

sel4test
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
===============================================
Output: /home/jesse/sel4-projects/sel4test/sel4test.xml
Cleaning up suites
Aggregating all robot results
Output: /home/jesse/sel4-projects/sel4test/robot_output.xml
Log:    /home/jesse/sel4-projects/sel4test/log.html
Report: /home/jesse/sel4-projects/sel4test/report.html
Tests finished successfully ;)
```
SUITE Test Suite

Full Name: Test Suite
HotSpot Action: -
Source: /home/jesse/sel4-projects/sel4test/tools/sel4Extra/remote_support/sel4test.robot
Status: 2 critical test, 2 passed, 0 failed
2 test total, 2 passed, 0 failed

+ SETUP remote-keywords.Setup

+ TEARDOWN remote-keywords.TearDown

+ TEST Should Boot sel4

+ TEST Should Test There Are Tests

Full Name: Test Suite:Should Test There Are Tests
Documentation: Check for the first test from sel4test
Tags: sel4 test uart
Status: PASS (critical)

+ SETUP Remote:Reset Emulation

+ KEYWORD Remote:Requires booted-sel4

+ KEYWORD Remote:Wait For Line On Uart <testcase classname="sel4test" name="Test that there are tests">, timeout=9999

+ KEYWORD Remote:Provides there-are-tests
Building More Complex Systems

• As systems get more complicated, there are way more resources to keep track of
  • This makes it difficult to make sure the system is configured in the correct, secure way

• CAmkES to the rescue
CAmkES: Get The Code

• Similar command as before

```
$ mkdir camkes
$ cd camkes
$ repo init -u https://github.com/dornerworks/riscvsummit-manifests -m camkes.xml
$ repo sync
```

• Start from a known working system
• Note about docker environment (need camkes target)
CAmKES: Project Layout

easy-settings.cmake -> projects/camkes/easy-settings.cmake
griddle -> tools/seL4/cmake-tool/griddle
init-build.sh -> tools/seL4/cmake-tool/init-build.sh
README.md -> projects/camkes-tool/docs/index.md
run-build.sh -> tools/seL4Extra/run-build.sh
run-renode.sh -> tools/seL4Extra/run-renode.sh
run-robot.sh -> tools/seL4Extra/run-robot.sh
start-container.sh -> tools/seL4Extra/start-container.sh

kernel$

projects$
  cakeml_libs$
  camkes$
  camkes-tool$
  capdl$
  global-components$
  musllibcs$
  picotcp$
  projects_libs$
  seL4_libs$
  seL4_projects_libs$
  sel4_runtime$
  util_libs$

tools$
  cogent$
  DockerFiles$
  pruner$
  riscv-pk$
  rumprun$
  seL4$
  seL4Extra$
CAmkES: Main Application Directory

```
projects/camkes/apps/serialcom$
  ├── CMakeLists.txt$
  │    └── components$
  │         └── SerialApplication$
  │                 └── src$
  │                         └── application.c$
  └── README$
      └── serialcom.camkes$
```
CAmkES: Serial Application

• Goal of this application:
  • Print and read from the serial device
  • Split up functionality among separated components

• Sounds Trivial
  • Consider taking advantage of the separation enforced by seL4 and the connections available by CAmkES
  • This won’t be in one address space
CAmKES: Serial Application

• Every CAmKES Application has a Root CAmKES file
  • This defines the architecture of the whole system
    • Instantiates Components and Connections
    • Defines connections
      • RPC interfaces
      • Shared Memory
    • Configures Components
assembly {
  composition {
    component SerialApplication serial_app;
    component SerialServer serial;
    component TimeServer time_server;

    connection sel4RPCCall serial_time_server(from time_server_putchar, to serial.processed_putchar);
    connection sel4RPCCall serial_client(from serial_app_putchar, to serial.processed_putchar);

    connection sel4SerialServer serial_input(from serial_app_getchar, to serial_getchar);
    connection sel4TimeServer serial_server_timer(from serial.timeout, to time_server.the_timer);

    connection sel4GlobalAsynchCallback global_callback(from serial_app_self, to serial_app_wait);
  }

  configuration {
    serial_app_putchar_attributes = 0;
    serial_app_getchar_global_endpoint = "serial_app";
    serial_app_getchar_badge = "1";
    serial_app_getchar_attributes = "0";
    serial_app_getchar shm mem_size = 0x1000;

    serial.timeout_attributes = 1;
    time_server_putchar_attributes = 0;

    time_server.timers_per_client = 1;

    serial.wait_global_endpoint = "serial";
    serial.self_global_endpoint = "serial";

    serial_app.wait_global_endpoint = "serial_app";
    serial_app_self_global_endpoint = "serial_app";
  }
}
CAmkES: Serial Application

```
component SerialApplication {
  control;
  attribute int simple = true;
  uses PutChar putchar;
  uses GetChar getchar;
  emits Callback self;
  consumes Callback wait;
}
```
CAmkES: Serial Server (Snippet)

```cpp
component SerialServer {
    control;
    has mutex serial;
    provides PutChar processed_putchar;
    provides PutChar raw_putchar;
    provides GetChar getchar;
    uses Timer timeout;

    /* Virtqueue connection to facilitate read and write */
    maybe uses VirtQueueDev read;
    maybe uses VirtQueueDev write;
    //emits Callback self;
    emits Callback self_read;
    emits Callback self_write;

    //maybe consumes Callback serial_wait;
    maybe consumes Callback serial_write_wait;
    maybe consumes Callback serial_read_wait;
}
```
CAmkES: RPC Call

• Must have an interface defined (procedure keyword)
  • At least one component “uses” the interface
  • One component “provides” the interface
CAmkES: RPC Calls (Procedure)

```c
procedure PutChar {
    void putchar(in int c);
};
```
CAmkES: RPC Calls (Provider)

Procedure Interface
Instance Name

RPC Function Name

```
void processed_putchar_putchar(int c)
{
   ... sel4_Word n = processed_putchar_get_sender_id();
   ... internal_putchar((int)n, c);
   ... if (c == '\n') {
      ... internal_putchar(n, '\r');
   ... }
}
```
CAmkES: RPC Calls (User)

```c
void print_string(char* string, uint32_t len)
{
    for (uint32_t i=0; i<len; i++)
    {
        putchar_putchar(string[i]);
    }
}
```

- Interface
- Instance Name
- RPC Function Name

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
CAmKES: Why?

• But I don’t trust generated code!
• There is too much indirection!
• Capability Calculation
  • Serial application generated roughly 1000 capabilities
    • Look for the *.cdl file in your build directory
CAmkES: Configure The Build

- Provided “run-build.sh” calls init-build.sh with the following CMake settings:
  - -DPLATFORM=hifive
  - -DSIMULATION=FALSE
  - -DElfloaderImage=elf
  - -DLibSel4PlatSupportUseDebugPutChar=OFF
  - -DCMAKE_BUILD_TYPE=Debug
  - -DRELEASE=False
  - -DCAMKES_APP=serialcom
  - -DCAmkESDTS=ON
  - -DRISCV64=TRUE

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
CAmkES: Run The Build

• Same process as before
• Run “run-build.sh” script or “init-build.sh” with your flags
Step 6/7: VOLUME /home/$(UNAME)
    ---> Using cache
    ---> 97f4e25ddbb0
Step 7/7: VOLUME /isabelle
    ---> Using cache
    ---> f9dc9fb90881
Successfully built f9dc9fb90881
Successfully tagged user_img--riscv:latest
docker run \
    -it \
    --hostname in-container \ 
    --rm \ 
    -u 1000:1000 \ 
    -v /home/jesse/DornerWorksProjects/RISC-V_Summit_2019/Code/Repo-camkes:/host \ 
    -v jesse-home:/home/jesse \ 
    -v /etc/localhost:/etc/localhost:ro \ 
    user_img--riscv bash

Hello, welcome to the sel4/CAmKEs/L4v docker build environment
CAmKES: Emulate It

:name seL4 Test
:description Runs seL4 Test on Renode
$name?="sel4test"

using sysbus
mach create $name
$bin?=@./riscv-build-hifive-64/images/capdl-loader-image-riscv-hifive
$dbt?=@./tools/seL4Extra/renode_support/hifive-renode.dbt
machine LoadPlatformDescription @./tools/seL4Extra/renode_support/sifive-fu540-dw.repl
showAnalyzer uart0

macro reset

""
    sysbus LoadELF $bin
    sysbus LoadFdt $dbt 0x81000000 "earlyconsole mem=256M@0x80000000"
    e51 SetRegisterUnsafe 11 0x81000000
    u54_1 SetRegisterUnsafe 11 0x81000000
""
runMacro $reset
Using seL4 in your projects

• Start from a manifest to get the build process correct
• Set up a testing environment
  • There are a lot of moving parts, it’s important to know when things go wrong
• Explore Userspace architecture systems:
  • Cogent
  • Rump Kernels
  • CAmkES
• Join the community
  • Mailing List: https://sel4.systems/lists/listinfo/
  • Github: https://github.com/seL4 and https://github.com/SEL4PROJ
  • Discourse: https://sel4.discourse.group/
• Checkout other Data61 Content
  • Tutorials: https://docs.sel4.systems/Tutorials/
  • FAQ: https://docs.sel4.systems/FrequentlyAskedQuestions.html
• DornerWorks can help with system design and board support
  • https://dornerworks.com/contact-us

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
Questions?
Extra Slides
seL4 Test: Root Thread

• Once seL4 has booted, it hands off the following to the “Root Thread”
  • All of the remaining “Un-typed” memory
  • Boot Info Structure

• Remember, the proofs don’t apply to code here
  • It is up to the programmer to make sure you don’t have overflows or any of the other runtime bugs not present in the kernel
  • There are tools to help here, such as Data61’s Cogent

• This thread has the ability to “Re-Type” this memory into different kernel objects

• It can create and delegate capabilities to the rest of the system

• In the seL4 test project, the root thread is responsible for:
  • Creating Virtual memory spaces
  • Creating Memory allocators
  • Creating Capability Spaces
  • Initialized needed hardware such as timers and interrupts
  • Creates a larger stack
  • Creates threads and hands out capabilities to tests
seL4 Test: Boot Process

- Berkley Boot Loader (riscv-pk project)
- Elfloader
- seL4 kernel
- Userspace Root Thread
seL4 Test: ELF Loader

- Build process archives the userspace image and the kernel in the same CPIO Archive
- ELF Loader extracts the images and places them at the proper spots