Computer Engineering Department

Porting OpenBSD to RISC-V ISA

Project Advisor: Michael Larkin

Bamsch, Brian (MS Software Engineering)
He, Wenyan (MS Software Engineering)
Li, Mengshi (MS Computer Engineering)
Waghela, Shivam (MS Software Engineering)

Introduction

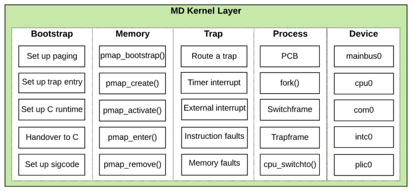
RISC-V is an open-source ISA developed by the University of California, Berkeley. The RISC-V ISA takes a modular approach to ISA design and encourages collaboration through practice of openness and transparency. The openness of the ISA plays to its strengths, particularly in the realm of security. Unlike other ISAs, open-source implementations of RISC-V benefit from community-driven security hardening efforts.

OpenBSD is well known for its strong security practices. However, the recent abundanance of obscure hardware-level security issues serves as a stark reminder that software systems are only as secure as their underlying hardware. We believe that the combination of security-focused supervisor software (OpenBSD) with an open and transparent hardware platform (RISC-V) will drive forward the development of noticably more secure platforms for general purpose computing. This project takes the first step towards a full port of the OpenBSD operating system to RISC-V, starting with the kernel.

Architecture

Kernel's Machine-Dependent Components

The project focuses on rewriting the architecture-specific routines and drivers of the machine-dependent layer of the OS kernel to support the RISC-V hardware architecture. The machine-dependent layer encapsulates the hardware details and provides services to the machine-independent layer of OS kernel, which can further provide Application Binary Interface to user programs. The kernel's machine-dependent layer is divided into five categories:

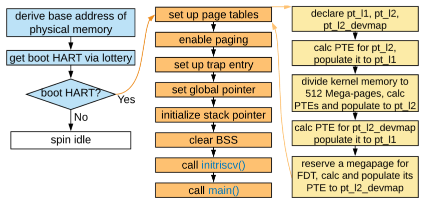


- Bootstrap: assembly initialization before jumping to C.
- PMAP: manages page table by interacting with MMU.
- Trap: handles interrupts and exceptions.
- Process: forks and schedules processes on HARTs.
- Device: probes and configures devices.

Implementation

Early Bootstrap Subsystem

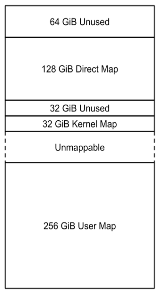
The early bootstrap subsystem written in assembly mainly sets up a preliminary page table, installs the trap handler, prepares the C runtime and jumps to C routine to continue bootstrap.


derive base address of physical memory
get boot HART via lottery
boot HART?
Yes
No
spin idle
set up page tables
enable paging
set up trap entry
set global pointer
initialize stack pointer
clear BSS
call initriscv()
call main()
declare pt_l1, pt_l2, pt_l2_devmap
calc PTE for pt_l2, populate it to pt_l1
divide kernel memory to 512 Mega-pages, calc PTEs and populate to pt_l2
calc PTE for pt_l2_devmap populate it to pt_l1
reserve a megapage for FDT, calc and populate its PTE to pt_l2_devmap

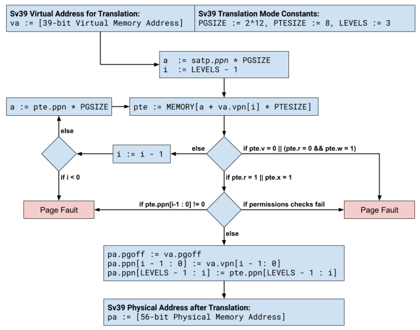
Memory Subsystem

OpenBSD uses the UVM Virtual Memory System to manage system memory resources. UVM operates on RISC-V machines in the *Sv39* virtual address translation mode.

Sv39 exposes a page-based virtual memory address scheme with 39 addressable bits. In Sv39 mode, page tables map virtual addresses to physical addresses. Virtual addresses are translated to physical addresses according to the state machine depicted below. The 512 GiB virtual address space is partitioned into regions as shown on the right. The upper 256 GiB is reserved for kernel use and the lower 256 GiB is reserved for user mode processes.



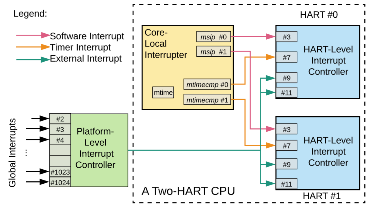
UVM manages the RISC-V MMU via a machine-dependent layer known as the Physical Map (PMAP). In this project, the Physical Map is adapted for the RISC-V MMU in *Sv39* mode.


Sv39 Virtual Address for Translation:
Sv39 Translation Mode Constants:
else
if i < 0
else
if pte.v = 0 || (pte.r = 0 && pte.w = 1)
if pte.r = 1 || pte.x = 1
if pte.ppn[i-1 : 0] != 0
if permissions checks fail
Page Fault
Page Fault
else
Sv39 Physical Address after Translation:

```
va := [39-bit Virtual Memory Address]
PGSIZE := 2^12, PTESIZE := 8, LEVELS := 3
a  := satp.ppn * PGSIZE
i  := LEVELS - 1
a := pte.ppn * PGSIZE
pte := MEMORY[a + va.vpn[i] * PTESIZE]
i := i - 1
pa.pgoff := va.pgoff
pa.ppn[i - 1 : 0] := va.vpn[i - 1: 0]
pa.ppn[LEVELS - 1 : i] := pte.ppn[LEVELS - 1 : i]
pa := [56-bit Physical Memory Address]
```

Trap Subsystem

A trap halts the CPU execution and transfers control to a handler for either an interrupt or an exception. Interrupts (software, timer and external interrupts) are managed by a two-level nested interrupt controller architecture as shown below.

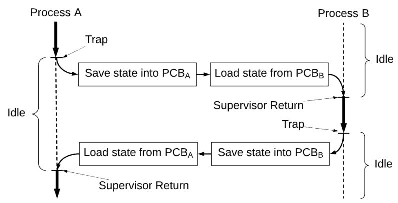


Exceptions are synchronous to HART's execution. All supported exceptions can be divided into two categories: instruction faults and memory faults. The OpenBSD kernel utilizes environment call from user-mode to implement system calls. Memory faults are normally handled by a call to *data_abort()*.

Code	Description
0	Instruction address misaligned
1	Instruction access fault
2	Illegal instruction
3	Breakpoint
4	Load address misaligned
5	Load access fault
6	Store / AMO address misaligned
7	Store / AMO access fault
8	Environment call from U-mode
9	Environment call from S-mode
10-11	*Reserved*
12	Instruction page fault
13	Load page fault
14	*Reserved for future standard use*
15	Store / AMO page fault
$\geq$16	*Reserved*

Process Subsystem

The process subsystem manages the lifecycle of processes running on the system. A new process is spawned from its parent via a system call to *fork()*. All processes are organized in a tree hierarchy.

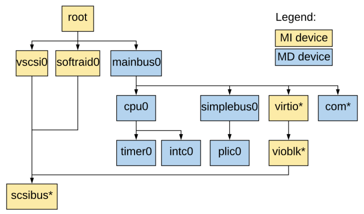


CPU scheduler is responsible for switching multiple processes that time-slice a few HARTs. A context switch is performed to save the execution state of the current-running process and restore the execution state of the upcoming process.

Device Subsystem

The device information is presented to kernel via FDT. All devices are probed and configured via the *autoconf* framework. The resultant device hierarchy is shown below.

This project needs to provide the device definition, configuration, and driver files for all MD devices. The *mainbus0* and *simplebus0* are virtual devices introduced for easy device probing and attaching. *timer0* provides periodical interrupts. *Intc0 and plic0* are two nested interrupt controllers. *com0* is the UART device for console output.



Summary/Conclusions

To port OpenBSD to RISC-V ISA, we have implemented assembly initialization for kernel early bootstrap, adapted PMAP to RISC-V MMU in *Sv39* mode, implemented routines for process creation and context switch, developed trap handlers for interrupts and exceptions, and developed drivers to probe and attach MD devices.

With supports of RISC-V from the above five subsystems, the OpenBSD kernel can boot up on a single HART and jump to a dummy "Hello-World" userland process.

Key References

[1] A. Waterman and K. Asanović, "The RISC-V instruction set manual, volume I: Unprivileged ISA," tech. rep., RISC-V Foundation, Dec. 2019.

[2] A. Waterman and K. Asanović, "The RISC-V instruction set manual, volume II: Privileged architecture," tech. rep., RISC-V Foundation, June 2019.

[3] H. H. Porter III, "RISC-V: An overview of the instruction set architecture," Jan. 2018.

[4] C. D. Cranor and G. M. Parulkar, "The UVM virtual memory system," in *Proceedings of the USENIX Annual Technical Conference*, pp. 117—130, June 1999.

[5] SiFive, "SiFive interrupt cookbook, version 1.2." SiFive, Inc., Feb. 2020.

Acknowledgements

We would like to thank our advisor, Michael Larkin. His expertise with OpenBSD has been essential to the team's success in both the implementation and bring-up of the OpenBSD operating system on RISC-V.