

Porting  NetBSD® to the

 RISC-V

“Of course it runs NetBSD!”

Zachary McGrew, Philip A. Nelson

Today's Talk

- Introduction
- Simulators, Toolchains, and Build Scripts, Oh My!
- Moving Targets
- Virtual Memory
- Pmap Common
- Current Status
- Future Work

Introduction - NetBSD

- Free
- Fast
- Secure
- Highly portable
Unix-like Open
Source operating
system



Introduction - NetBSD

- The NetBSD project was started in 1993
 - Happy 25th birthday!
- Zach McGrew is a new NetBSD developer
 - I worked on the RISC-V port for grad school
- Phil Nelson is NetBSD developer #6
 - He did the PC532 port
 - Currently updating the Wi-Fi stack
- Runs on **57** different platforms
 - And **16** different types of CPUs
- “Of course it runs NetBSD!”



(armv6hf)

(amd64)

Other toaster (A.K.A. IronForge)
(armv7hf)

So why not one more?

Toaster
(arm32)

```

Memory Test:
ST RAM ----- 4096 KB
TT RAM ----- 65536 KB
Memory Test Complete.

NetBSD/atari secondary bootloader ($Revision: 1.17 $)

|
NetBSD/Atari tertiary bootloader ($Revision: 1.8 $)

```

NetBSD

```

>>> Toaster ready to accept toast.
Press "Cancel" button to abort anything.
Total toasted: 159. Burnlevel: 257
Current CPU board temp is 42.35C/108.23F

```

ok
(64)

cast

Introduction - RISC-V

- Originally developed at U.C. Berkeley in 2010 as a teaching example
- A free and open source ISA
- Royalty free too!
- Currently controlled by the RISC-V foundation

- Who are the foundation members?



Google

QUALCOMM

Western Digital

SEAGATE

SAMSUNG

HUAWEI

oculus



NVIDIA

TESLA

HITACHI
Inspire the Next

IBM

NXP

MARVELL

Micron

Hewlett Packard
Enterprise

Alibaba Group
阿里巴巴集团

(a few) RISC-V Foundation Members

Introduction - RISC-V



- Multiple specifications
 - RISC-V ISA
 - RV32I
 - RV32E
 - RV64I
 - RV128I (*Seriously*)
 - Many extensions to the specification
 - Privileged ISA

- **NetBSD** port targets RV64GSU (*RV64IMAFDSU*)
 - Plans for eventual RV32GSU support

RISC-V Extensions

- **A** - Atomic Instructions
- **B** - Extended Bit Manipulation
- **C** - Compressed Instructions
- **D** - Double-Precision Floating-Point
- **E** - Base Integer Instruction Set (embedded), 32-bit, 16 registers
- **F** - Single-Precision Floating-Point
- **G** - Shorthand for IMAFD extensions
- **I** - Base Integer Instruction Set, [32-bit, 64-bit, or 128-bit], 32 registers
- **J** - Dynamically Translated Languages
- **L** - Decimal Floating-Point
- **M** - Integer Multiplication and Division
- **N** - User-Level Interrupts
- **P** - Packed-SIMD Instructions
- **Q** - Quad-Precision Floating-Point
- **S** - Supervisor mode
- **T** - Transactional Memory
- **U** - User mode
- **V** - Vector Operations
- **Yxxx/Zxxx** - Nonstandard vendor extensions



Simulators,
Toolchains,
and Build Scripts,
Oh My!

Simulators

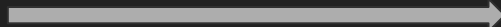
- Spike
 - “RISC-V ISA Simulator, implements a functional model of one or more RISC-V processors.”
 - Feed it a RISC-V binary and run code on your standard AMD64 desktop/laptop
 - Spike doesn't have a BIOS/UEFI services
 - Needs a boot loader to actually load the binary (BBL)
 - BBL provides BIOS/UEFI services via SBI
 - BBL probes the hardware and generates the FDT
 - Still a problem...
 - Need to build the RISC-V binaries
 - BBL
 - **Net**BSD Kernel

Toolchains

- BBL needs needs a C library to do some of its work
 - riscv64-unknown-elf-
 - Build Binutils (gas, ld, and friends)
 - Build GCC
 - Build Newlib (Small C library)
 - Build GCC *again*

- **Net**BSD kernel uses it own “mini” C library
 - riscv64--netbsd-
 - Build Binutils (gas, ld, and friends)
 - Build GCC

Buildscripts

- Building toolchains is complicated, and takes a lot of work
 - You need to pass the correct flags to the configure script
 - Enjoy a sample of my nightmare 
- Got a working compiler?
 - Great
 - Now try and compile the kernel
 - Use the [NetBSD](#) build script...

```
68 binutils-newlib() {
69     cd "${TOP}/riscv-gnu-toolchain/riscv-binutils-gdb"
70     rm -rf build-binutils
71     mkdir build-binutils
72     cd build-binutils
73     #--disable-gdb
74     ../configure \
75     --target=riscv64-unknown-elf \
76     --with-arch=rv64imafd \
77     --with-abi=lp64d \
78     --prefix=${RISCV} \
79     --with-guile=no \
80     --disable-sim \
81     --enable-tls \
82     --disable-intl \
83     --disable-werror &&
84     gmake ${MAKES} && gmake ${MAKES} install
85 }
86
87 gcc-noheaders() {
88     cd "${TOP}/riscv-gnu-toolchain/riscv-gcc" &&
89     rm -rf build-gcc &&
90     mkdir build-gcc &&
91     cd build-gcc &&
92     ../configure \
93     --target=riscv64-netbsd \
94     --with-arch=rv64imafd \
95     --with-abi=lp64d \
96     --prefix=${RISCV} \
97     --without-headers \
98     --disable-multilib \
99     --disable-werror \
100    --disable-shared \
101    --enable-static \
102    --disable-threads \
103    --enable-tls \
104    --enable-languages=c,c++ \
105    --disable-libatomic \
106    --disable-libmudflap \
107    --disable-libssp \
108    --disable-libquadmath \
109    --disable-libgomp \
110    --disable-nls &&
111    echo "Preparing to call gmake..." &&
112    gmake ${MAKES} all-gcc &&
113    echo "Preparing to install gcc in $RISCV" &&
114    gmake ${MAKES} install-gcc
115 }
116
117 gcc-stagel() {
118     cd "${TOP}/riscv-gnu-toolchain/riscv-gcc"
119     rm -rf build-gcc
120     mkdir build-gcc
121     cd build-gcc
122     ../configure \
123     --target=riscv64-unknown-elf \
124     --with-arch=rv64imafd \
125     --with-abi=lp64d \
126     --prefix=${RISCV} \
127     --without-headers \
```

Buildscripts - build.sh

- **Net**BSD uses a build script to wrap its makefiles
 1. Build Make without a makefile
 2. Do a bunch of automagic detecting of things that Make can't detect
 3. Invoke the newly built Make telling it about all the things you've learned
 4. Make builds all the tools it needs to build **Net**BSD
 - 4.1. Uh... We've got our own external toolchain already *(fsekl!)*
 - 4.2. Modify the makefiles to accept an external toolchain
 - 4.3. Rejoice!
 5. Make builds the kernel
 - 5.1. Discover ~50 compilation errors in code that's not even in your part of the source tree because you're using a newer, "smarter" version of GCC than the rest of the **Net**BSD developers
 - 5.2. Turn off `-werror`
 - 5.3. Rejoice!

Oh My!

- Got a built kernel?
 - *Can't load it into Spike directly*
- Wrap kernel in BBL
- Each new kernel build requires a build of BBL as well
- Extend build script to automate this process
- Load BBL that holds the kernel into Spike
- Crash & Burn.



Moving Targets

- RISC-V port was started by Matt Thomas in 2015
- Specs weren't as concrete as they are today
 - Specs have changed
 - A lot
 - In small ways
 - But enough to break ~~things~~ everything

- SFENCE.VM **Vs.** SFENCE.VMA
 - One letter difference (*Goodbye SFENCE.VM*)
 - Assembler will still generate opcode for you
 - Spike will get angry and crash though...

Moving Targets

- Privileged Spec changed the most
 - Page Table Entries (PTEs) completely redone
 - New way of marking transient entries in page tables
 - Interrupts handled completely different
 - Control and Status registers moved bits all over the place
 - Privileges themselves are mapped differently
 - (Currently) no hypervisor mode - May come back?
 - Supervisor can't read user's memory by default
 - Meltdown isn't (currently) a problem on RISC-V! w00t!

Moving Targets

- What does it all mean?
 - locore.S is pretty much garbage at this point
 - Needs a rewrite
 - Start with bootstrapping virtual memory

Virtual Memory

- RISC-V has three different sizes of virtual memory
 - Sv32 - 4 GB max memory
 - Sv39 - 512 GB max memory
 - Sv48 - 256 TB max memory
- **NetBSD** port uses Sv39
 - Shouldn't be hard to move to Sv48 in the future, just add extra page of lookups
- Sv39 is three layers
 - 512 entries of 1 GB
 - 512 entries of 2 MB
 - 512 entries of 4 K

Virtual Memory

- **Net**BSD takes advantage of the hardware support for big pages
 - Kernel is initially mapped on 2 MB pages
 - Expands on standard 4 K pages when it requests more memory
- Create L1 page table
 - Put entry in for start of kernel address
 - Currently `0xffffffff000000` — Makes the math easy
- Create L2 page table
 - Put entries for all 2 MB pages up until just past the end of the kernel
- No L3 pages created by default

Virtual Memory

- Got a page table? Great!
- Load it into the Supervisor Address Translation and Protection (satp) register
- Crash & Burn.



Virtual Memory

- Anytime the satp register is changed it causes a fault
- Clever fix (*Thanks FreeBSD!*)
 - Set fault vector to my current address in highmem + 4
 - Set satp
 - Fault to highmem and keep running
 - Set fault vector to real fault handler
- Alternate fix
 - Set bit in mtvec to not fault when satp register changes
 - Requires modifying BBL
 - Repercussions?
- Now running where we belong in virtual memory! 😊
 - Spend an hour writing a console device driver and keep going

Pmap

- Physical mapping of memory in processes
 - pmap(9)
 - “machine-dependent portion of the virtual memory system.”
 - Very complex code
 - FreeBSD RISC-V devs said it was the hardest part of the port for them
 - Can't mess it up, or you're in for a world of hurt when debugging
-
- Every machine does the more or less the same task, why not abstract it further?

Pmap Common

- Started in ~2011 by Matt Thomas — Same guy who started RISC-V port
- Handle all the normal work (*machine independent*) that all platforms do anyway
- Call helper functions to do small amounts of *machine dependent* work
- Awesome idea for **portability!**
- In use by MIPS and (some) PowerPC ports
- Work started to convert other ports to it as well

Pmap Common

- Just one small problem...
 - Only works on platforms without hardware page tables
 - RISC-V has a hardware page table
 - MIPS and PowerPC just have TLBs
 - RISC-V doesn't allow direct writes to the TLB
 - TLB writes are managed by the MMU
- Crash & Burn.



Pmap Common

- This project extended pmap common to support hardware page tables
 - New set of helper functions
 - Create/update hardware specific page tables
 - Do manual lookups when needed
 - Extract hardware bits for permissions

- Some things can't exist in pmap common
 - `pmap_bootstrap()`
 - Hyper specific to the platform it's running on

Current Status

😊 Virtual memory

😊 Console driver

😊 Pmap

😊 kthread_create()

😊 fork1()

😊 cpu_switchto()

🚫 Root file system

Current Status

“Of course it runs NetBSD!”

```
1 2 3 4 zmcgrew@nothingman [ Mem: 14% | CPU: 12% | Thermal: 45°C | 38 | 100% | Thu Oct 25, 21:33 ]
zmcgrew@wanderingstar zmcgrew@nothingman zmcgrew@mrmagpie
memory@80000000 {
  device_type = "memory";
  reg = <0x00000000 0x80000000 0x00000000 0x80000000>;
}
soc {
  #address-cells = <0x00000002>;
  #size-cells = <0x00000002>;
  compatible = "ucbbar,spike-bare-soc", "simple-bus";
  ranges;
}
htif {
  compatible = "ucb,htif0";
}
}
entry point set
0x0000000080200000
entering supervisor mode
preparing to return to entry (0x0000000080200000)
[ 1.0000000] pmap_steal_memory: need 8359 pages
[ 1.0000000] pmap_steal_memory: seg 0: 0x80800 0x80800 0xc0000 0xc0000
[ 1.0000000] pmap_steal_memory: seg 0: 8359 pages stolen (0x3d759 left)
[ 1.0000000] Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005,
[ 1.0000000] 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017,
[ 1.0000000] 2018 The NetBSD Foundation, Inc. All rights reserved.
[ 1.0000000] Copyright (c) 1982, 1986, 1989, 1991, 1993
[ 1.0000000] The Regents of the University of California. All rights reserved.

[ 1.0000000] NetBSD 8.99.25 (GENERIC) #451: Thu Oct 25 21:22:26 PDT 2018
[ 1.0000000] zmcgrew@nothingman:/home/zmcgrew/netbsd/rv64/obj/sys/arch/riscv/compile/GENERIC
[ 1.0000000] total memory = 1024 MB
[ 1.0000000] avail memory = 983 MB
[ 1.0000000] mainbus0 (root)cpu0 at mainbus0 instance 0cpu0: WARNING: power management not supported
[ 1.0000000] mainbus0: WARNING: power management not supported
[ 1.0000030] root on md0a dumps on md0b
[ 1.0000030] Supported file systems: mfs lfs ext2fs ffs nfs umap procfs overlay null kernfs fdsc union tmpfs smbfs puffs ptys ntfs msdos cd966
0 coda
[ 1.0000030] no file system for md0 (dev 0x12e00)
[ 1.0000030] cannot mount root, error = 79
[ 1.0000030] root device (default md0a): █
[0] 0:spike* 1:bash 2:bash 3:bash 4:bash 5:less 6:bash "nothingman" 21:30 25-Oct-18
```

Current Status

- I joined the **Net**BSD Project and got my commit bit last month
- Maya got userland built, so I'll be at the hacker lounge tonight trying to get it running the init process.
- Need some free time to sit down and work with Maxime Villard (port maintainer) and get it merged into the CVS tree
 - Side note: I graduated and got a job with weekends off, so this can actually happen =)

Future Work

- Support 32-bit and 64-bit NetBSD tree
 - Other developers already play with FreeBSD code
 - Enables to stop slacking and keep working
 - Getting rid of the external toolchains
 - Ability to build a root file system
 - Hey who likes userland tools?

Future Work

- Other simulators
 - QEMU seems to be popular among RISC-V community

- Booting on physical hardware
 - SiFive HiFive Unleashed board is sitting on my workbench
 - Get access to devices — Like networking!

Future Work

- DDB
 - Stack traces on RISC-V are weird
 - RISC-V has a return address register (ra)
 - This means the return address doesn't always get pushed onto the stack
 - Provides a speedup by not doing this, but makes it harder to debug
 - Started work on this, but haven't figured out a way to extract the needed information
 - Current debugging method: `printf()`;

- FDT support
 - Would be nice to know exactly what the hardware is
 - Also memory ranges
 - System currently makes assumption about RAM


Future Work

- SMP
 - Machines have all these “extra” cores now...
 - Might as well use them?
 - Currently all cores besides the first just sit in an infinite loop waiting on interrupts
 - Requires IPI work and probably more locking

- RV32 compatibility
 - NetBSD has extensive compatibility for other platforms
 - AMD64 can run i386 binaries
 - Aarch64 can run arm32 binaries
 - Would be neat if RV64 could run RV32 binaries

Future Work

- Reclaim wasted memory after kernel and up to 2 MB boundary
 - Non-debug kernel is slightly over 8 MB, which means 10 MB gets mapped
 - Rest of that space can't be reclaimed
 - What's the fix?
 - Map 8 MB on 2 MB pages, then the rest on 4 K pages

- Teach malloc() about big pages
 - Could help speed up memory requests for memory hungry programs
 - Looking at you, Firefox. 
 - Future research project?

Future Work

- Move to Sv48 Virtual Memory
 - Expands address space, allowing address space randomisation even more space to play with
 - Shouldn't be too much work to extend Sv39
 - Kernel option to pick?

Thanks

- Phil Nelson - Research advisor and all around awesome dude
- Aran Clauson - For inspiration and listening to my dumb ideas
- Nick Hudson - Helped with PMAP Common stuff when I was really lost
- Matt Thomas - Starting PMAP Common and RISC-V port



ANY QUESTIONS?