Embedded FreeBSD Development and Package Building Via QEMU

Sean Bruno, sbruno@FreeBSD.org

Stacey Son, sson@FreeBSD.org
Overview

- Significant Events in the History of Emulation
- A Very Brief Introduction to QEMU
- QEMU User-Mode Emulation
- Misc Binary Image Activator
- Cross Development using QEMU
- Poudriere Bulk Cross Building (Demo)
- Current State and Future
- Credits and Q&A
Significant Events in the History of Emulation

- **Theory:** Universal Turing Machine (1936)

- **Cross Development:** Gates/Allen's Altair 8800 Emulator (1975)

- **Transparent:** Apple's (or Transitive's) Rosetta (2006) and 68k emulator (1994)
Significant Events in the History of Emulation

- Theory: Universal Turing Machine (1936)

- Cross Development: Gates/Allen's Altair 8800 Emulator (1975)

- Transparent: Apple's (or Transitive's) Rosetta (2006) and 68k emulator (1994)
Significant Events in the History of Emulation

- **Theory:** Universal Turing Machine (1936)

- **Cross Development:** Gates/Allen's Altair 8800 Emulator (1975)

- **Transparent:** Apple's (or Transitive's) Rossetta (2006) and 68k emulator (1994)
Intro to QEMU

- QEMU = Quick EMUlator
- Fast, flexible, open source hardware emulator
- Has played a quiet but essential role in many other projects, including:
  - KVM
  - Xen
  - VirtualBox (forked version)
  - Android SDK (forked version)
    - In fact, a lot of embedded SDK's
QEMU's History

• Started by Fabrice Bellard in 2003
  – FFMPEG, TinyCC, TinyGL, JSLinux, etc.

\[\pi = \frac{1}{20} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{10n}} \left( -\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^5}{10n+3} - \frac{2^3}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9} \right)\]
QEMU's History

- Started by Fabrice Bellard in 2003
  - FFmpeg, TCC (and OTCC), JSLinux, etc.
- Initially a portable JIT translation engine for cross architecture emulation (aka. User Mode Emulation)
QEMU's History

- Started by Fabrice Bellard in 2003
  - FFMPEG, TCC (and OTCC), JSLinux, etc.
- Initially portable JIT translation engine for cross architecture emulation (aka. User Mode Emulation)
- Emulation of PC hardware added (aka. System Mode Emulation)
QEMU's History

- Started by Fabrice Bellard in 2003
  - FFmpeg, TCC (and OTCC), JSLinux, etc.
- Initially portable JIT translation engine for cross architecture emulation (aka. User Mode Emulation)
- Emulation of PC hardware added (aka. System Mode Emulation)
- Virtualization, Management API, Block Layer, etc.
QEMU's History
QEMU User Mode Emulation

- Only CPU is emulated. MMU, I/O, etc. are not.
- System calls are translated to host calls and/or emulated.
- Can use native host tools for cross development. Cross debugging and testing.
QEMU User Mode Emulation

- Only CPU is emulated. MMU, I/O, etc. are not.
- System calls are translated to host calls and/or emulated.
- Can use native host tools for cross development. Cross debugging and testing.

(More on this in a minute...)
QEMU User Mode Emulation

- Only CPU is emulated. MMU, I/O, etc. are not.
- System calls are translated to host calls and/or emulated.
- Can use native host tools for cross development. Cross debugging and testing.

(Remember these guys?)
System Call Argument Translation

Target(mips) ⇄ Host(amd64)

- **Endian:**
  - Byte Swap Arguments

Storage of the value D7C4₁₆

<table>
<thead>
<tr>
<th>Big Endian</th>
<th>Little Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motorola Processors: 68000, 68030, etc..</td>
<td></td>
</tr>
<tr>
<td>Intel Processors: 80386, Pentium, etc..</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>C4</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>D7</td>
<td></td>
</tr>
</tbody>
</table>
System Call Argument Translation

Target(mips) ↔ Host(amd64)

- Endian :
  - Byte Swap Args

- Word Size :
  - 32-bit to 64-bit conversion
System Call Argument Translation

Target(mips) ⇄ Host(amd64)

- **Endian** :
  - Byte Swap Args

- **Word Size** :
  - 32-bit to 64-bit conversion

- **ABI Differences** :
  - e.g. 64-bit arg passed in two evenly aligned 32-bit registers
  - Repackage 32-bit registers into a 64-bit argument
System Call Argument Translation

Target(mips) ↔ Host(amd64)

• Pointers:
  - Strings (No Problem)
  - Arrays (Byte Swap, 32to64 depending on element type)
  - Structures (Byte Swap, 32to64 depending on elements types, offsets)
  - Temporary buffer management and locking
Problem System Calls

- mmap() and friends
- Signals related calls
- fork(), threads and _umtx_op()
- ioctl() and sysctl()
- syscall() - ${ARCH} dependent syscalls.
- Other misc calls (most of which we simply don't support but don't need).
mmap()

- Target code and QEMU use the same address space.
- Target MAP_FIXED mappings that conflict with the QEMU host's mappings are mapped elsewhere but then fixed it in the emulation.
- QEMU keeps a table of all the host mappings.

Target: `mmap(MAP_FIXED)`

<table>
<thead>
<tr>
<th>Host Mapping</th>
<th>QEMU Offsets</th>
<th>Target Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- `mmap()` function
- Target code and QEMU use the same address space.
- Target MAP_FIXED mappings conflict with QEMU host's mappings.
- QEMU maps these mappings elsewhere and then fixes them in the emulation.
- QEMU keeps a table of all the host mappings.
Signal Handling

- Target signals are mostly muxed with host signals.
- Target signals are queued and then dispatched out the main loop.
- Therefore, the emulation of the basic block has to finish before target gets the signal.
Threads and _umtx_x_op()

- Threads are mapped to pthreads one-to-one.
- The undocumented _umtx_x_op() system call supports many operations or commands that embedded flags into the same field as counters/semaphores.

  e.g. UMTX_OP_SEM2_WAIT, the high order bit of semaphore is a 'has waiters' flag. The kernel ends up checking or flipping the wrong bit when the host and target are different endian. Currently, we do user level emulation of these => Slow/Ugly

Solution? (Maybe add other endian versions of these calls.)
ioctl() Thunking

• ioctl() and sysctl() are used and abused for passing large amounts of data in and out of the kernel.

• Thunking – A generic way using macros to convert data flowing in and out with the ioctl() system call to save LOC. e.g...

  \[\text{IOCTL(TIOCFLUSH, IOC\_W, MK\_PTR(TYPE\_INT))}\]
  \[\text{IOCTL(TIOCGWINSZ, IOC\_R, MK\_PTR(MK\_STRUCT(STRUCT\_winsize)))}\]

• Thunking should also be used for sysctl() but it's not (yet).

• Many ioctl()'s and sysctl()'s are not supported.
Sysarch() and Others

- sysarch() is emulated. Mainly for thread local storage, etc.
- Other system calls that are missing:
  - Jail related system calls.
  - Mandatory Access Control or mac(3) calls.
  - kld(4) related calls.
  - Capsicum(4) related calls.
  - Exotic networking: e.g. sctp(4) and some socket options.
  - sendfile(2), ptrace(2), and utrace(2).
  - Some misc others.
Adding a New Arch to QEMU BSD User-Mode (1/2)

- https://github.com/seanbruno/qemu-bsd-user/ (bsd-user branch)
- Arch dependent code: bsd-user/${arch}
  _cpu_init() - CPU startup initialization
  _cpu_loop() - CPU exception decoding/dispatching
  _cpu_{get, set}_tls() - Get/Set TLS in CPU state
  _cpu_fork() - CPU state initialization for child after fork()
  {get, set}_mcontext() - Get/Set machine context/ucontext
  _thread_init() - First thread initialization after loading image
  _thread_set_upcall() - New thread CPU state initialization
set_sigtramp_args() - Set up the signal trampoline arguments in the QEMU CPU state

get_ucontext_sigreturn() - Get the user context for sigreturn()

setup_sigtramp() - Customize/Copy the signal trampoline code into the target memory space.

_arch_sysarch() - sysarch() syscall emulation

get_sp_from_cpustate() - Get the stack pointer

set_second_rval() - Set the second return value
'imgact_binmisc.ko' is a kernel image activator that will invoke an user-level emulator or interpreter based the binary header of the file.

binmiscctl(8) is a command-line utility that is used to load the kernel module (if not already loaded) and configure the interpreter/emulator path for a set of magic bytes and mask.

Part of FreeBSD since 10.1.
imgact_binmisc Kernel Module

x86 Host

ARM Binary

a.out --arg

/usr/bin/qemu-arm a.out --arg
Binmiscctl(8) Examples

- LLVM bitcode interpreter ('lli'):
  ```
  # binmiscctl add llvmbc --interpreter "/usr/bin/lli --fake-arg0=#a" --magic "BC\xc0\xde" --size 4 --offset 0 --set-enabled
  ```

- QEMU MIPS64 emulator ('qemu-mips64'):
  ```
  # binmiscctl add mips64elf --interpreter "/usr/bin/qemu-mips64" --magic "\x7f\x45\x4c\x46\x02\x02\x01\x00[...]" --mask "\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\x00[...]" --size 20
  ```

- See binmiscctl(8) for additional examples.
Cross Debugging, using QEMU's gdb server:

```
% qemu-arm -g 4567 a.out
```

- Using cross gdb in second terminal:
  
  ```
  % cross-gdb a.out
  (gdb) target remote 127.1:4567
  ```

- Using lldb* in second terminal:
  
  ```
  % lldb a.out
  (lldb) gdb-remote 4567
  ```

QEMU currently doesn't create target cores.

- It only dumps the core image of the emulator.
Binary Packages for my RPi?

- Goal: **Binary FreeBSD Packages for Tier 2 Architectures**
- Number of Raspberry Pi's sold (as of 2/15)... > 5 Million!
- OK, my Raspberry Pi is running FreeBSD. Now what?

"FreeBSD - Helping kids get a better OS!"
Cross Building Packages for Tier 2 Arch's

Solutions:

- Ideally, cross building should be easy (e.g. 'make crossbuild')
  - Autotools, cmake, /usr/share/mk/*, etc. are somewhat friendly for this.
  - Others not so friendly.*

- Hardware (or full emulation), distcc, and NFS

- QEMU user-mode

* See Baptiste's EuroBSD 2014 Talk for Details:
http://www.slideshare.net/eurobsdcon/baptiste-daroussin-crosscompiling-ports
Building Packages with Large Amounts of Hardware

- Stacks of Embedded System Boards, distcc, NFS
  - Limited Memory
  - Switch Ports/Console and Power Management ($$$$)
  - Not Rack Friendly

- Target $$server $$$olutions
  - e.g. Calxeda/SLS ECX-1000 ($20K USD)
Cross Building with QEMU User-Mode

- Create a jail image (w/ 'qemu-static-user' port):
  
  ```bash
  # poudriere jail -c -j 11armv632 -m svn -v head -a arm armv6 -x
  -or-
  # poudriere jail -c -j 11mips32 -m svn -v head -a mips mips -x
  -or-
  # poudriere jail -c -j 11mips64 -m svn -v head -a mips mips64 -x
  -and add something to build-
  # poudriere ports -c -m svn
  ```

- Mount devfs and nullfs for ports:
  
  ```bash
  # mount -t devfs devfs <path_to_jail>/dev
  # mount -t nullfs /usr/local/poudriere/ports/default
  <path_to_jail>/usr/ports
  ```

- Chroot and Enjoy:
  
  ```bash
  # chroot /usr/local/poudriere/jails/11armv632
  # uname -p
  armv6
  ```
Using a Cross Build Toolchain with QEMU

- Make a cross build toolchain (i.e. 'make xdev') and install into jail. With imgact_binmisc it just works.

The 'cd /usr/ports/editors/vim-lite && make' Benchmark:

- Replacing things like /bin/sh with host native versions further benefits performance.
Poudriere Bulk

Using the tools you already know
Userland Components

- Poudriere is the easiest way to get started
- Knows how to understand binmiscctl(8)
- Knows to copy QEMU into jails
- Creates clean backup, in case of accident
- Use ZFS, save yourself some pain
Current State of QEMU Cross Building

- The ports cluster is building packages for arm, mips, and mips64. Nearly 50,000 packages!
  - Over 20,000 for arm, 15,000 for mips and 12,000 for mips64. (All coming to a pkg.FreeBSD.org near you.)
- Aarch64/ARM64 support is mostly there
  - Have cross built a handful of packages (e.g. vim-lite)
  - Missing some threading/\_umtx\_op() stuff, etc.
- QEMU- Sparc64 and PPC will run simple static binaries.
Future

- Cross build (most) ports without QEMU. Only use QEMU with that doesn't work (as 'plan b')
- Build more arm, mips, and mips64 packages
  - Toolchain, bug fixes, etc.
- Start building Aarch64/arm64 packages
- Better cross debugger support and add target core file generation
- Support for PPC
Credits

- **Stacey Son** – binmiscctl(8)/imgact_binmisc(4) and QEMU user-mode for FreeBSD.
- **Juergen Lock** – QEMU ports maintainer and patch contributor.
- **Ed Maste** – QEMU patch contributor and cat herder.
- **Peter Wemm** – Sigtramp patch.
- **Alexander Kabaev** – QEMU patch contributor.
- **Adrian Chadd** – For ignoring Sean's pleading for help with kern_imgact.c.
- **Baptiste Daroussin** – Poudriere and inflicting Sean with a ports commit bit.
- **Bryan Drewery** – Poudriere and support.
- Dimitry Andric – Clang Help and Updates
- Andrew Turner – Arm GCC and Ports Patches
- Mikael Urankar – Mysql Patches
- Warner Losh – Created the native-xtools target
- Ian Lapore – ARMv6 Assembly Help
- Brook Davis – Inspiration and initial guidance
- Sean Bruno – The master electrician that wired all this together and got it working
- U.S. Taxpayers – For funding some of this work*

* Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237.
QEMU BSD User-Mode Src: https://github.com/seanbruno/qemu-bsd-user/tree/bsd-user

QEMU User-Mode HowTo: https://wiki.freebsd.org/QemuUserModeHowTo

Sean's Blog: http://blog.ignoranthack.me

Email: {sbruno, sson}@FreeBSD.org