Cross Building Packages

Stacey D. Son
Consultant/SRI International

BSDCan Developer’s Summit
15-May-2013
Classic Cross Building

Software supported cross building:

- TARGET=mips/TARGET_ARCH=mips64
- trhodes’ bsd.crossbuild.mk
- ./configure --host=mips64-freebsd

And when the above doesn’t work:

- distcc, NFS, and lots of embedded hardware or full system emulators

*See https://wiki.freebsd.org/CrossBuildingPorts for details.
Advantages
Software Supported

• Very fast results
• Works on lots of different host hardware
• Nice (when it is supported and it works).
Disadvantages

Software Supported

• Sources usually need to support cross building, dependencies for two architectures, etc.

• Build may differ from native target compile.

• No unit testing and regression testing during development and post build w/out moving binary to the target.

• No debugging w/out moving binary to the target.
System Mode Emulation

- System mode requires emulation of devices and hardware such as the MMU in addition to the CPU.
- Full target kernel is emulated as well.
- Because it has a lot of overhead may not be too practical for cross building/development.
User Mode Emulation

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

• Full System Emulation ("System Mode")
  • Has been used with distcc, NFS, etc. to offset the performance issues.

• User Only Emulation ("User Mode")
  • Used by some Linux embedded developers.
  • Some preliminary investigation by NetBSD developers.*

* "build.sh: Cross-building NetBSD", L. Mewburn and M. Green
Qemu User Mode

- No MMU emulation: Simply uses host mmap()‘s with offsets.
- Target kernel threads map one-to-one to host pthread threads.
- Target signals are (in some cases) multiplexed with the host signals.
- Handles endianness and 32-bit target to 64-bit host translation issues
Advantages

• No, or few, changes are needed ports to support cross building. Autoconfig scripts that do things like compile and run bits of test code work.

• Regression/unit tests can be run during cross development or post build checks.

• Can be used to reduce the development cycle time for embedded systems.
Disadvantages

- The emulator may have bugs and missing support which may influence the build results.
- Some system calls are problematic like sysctl(), ioctl(), signals, fork(), threads, _umtx_op(), etc.
- Support for things like new system calls need to also be added to the emulator. May get out of sync with kernel.
- While it is much faster than full system mode emulation there is still a lot of overhead.
- Some kernel support may need to be added to the host.
Initial State of Qemu User Mode on FreeBSD

- Qemu version 1.2.0
- Qemu bsd-user (User Mode for *BSD):
  - It would emulate a simple ‘Hello World!’ app for statically compiled ARM binary.
  - No signals, threads, user mutex, support for other arch’s, etc.
  - Explicit support for maybe 10 system calls.
Current Status of Qemu
BSD User

• Qemu 1.4.1
• Static and dynamic target binaries supported.
• System calls **not** supported: ktimer_*, cpuset_*, rctl_*, sctp_*, kld*, quota*, jail*, cap_*, jail*, _mac*, sendfile, ptrace, & utrace.
• MIPS64 and ARM has the needed machine dependent code and will run static/dynamic binaries. PPC has some of the machine dependent code and will run some very simple statically linked apps.
• Not all ioctl()’s, sysctl()’s, and sockopts are supported.
  * see [http://wiki.freebsd.org/QemuUserModeToDo](http://wiki.freebsd.org/QemuUserModeToDo) for details.
Cross Building FreeBSD Packages Using Qemu BSD User

- Cross build a ${ARCH} ‘root’ distribution for target. Install in ${DESTDIR}.
- Add devfs: `mount -t devfs devfs ${DESTDIR}/dev`
- Build statically linked version of qemu-${ARCH}. Install in ${DESTDIR}/usr/local/bin.
- chroot into ${DESTDIR}.
- `cd /usr/ports/${favorite_port} && make package`

*see https://wiki.freebsd.org/QemuUserModeHowTo for the details.
Results

Cross Building MIPS64 Packages

• Added simple script that simply tries to build all packages. If it fails then it goes on to next port.

• Over 9000 packages have been successfully cross built using an old, dual core AMD64 athlon for the emulator host.

• Perl 5.14 regression test results running under user mode emulation: "Failed 2 tests out of 1970, 99.90% okay." (The same two tests fail on target as well.)

*See package repo at http://www.cl.cam.ac.uk/research/security/ctsrd/mips64-packages/
Hybrid Cross Building Environment

Using native cross compiler in user mode emulation build environment:

5.15x Improvement over pure emulation
Kernel Support for Hybrid Environment

Miscellaneous Binary Image Activator:

- ‘imgact_binmisc’ kernel module and ‘binmiscctl’ command-line configuration tool.
- Invokes configured interpreter if given header magic (and optional mask) at file offset matches.
- Makes it possible to use lots of host native binaries in the cross build environment to increase performance.

* See http://people.freebsd.org/~sson/imgact_binmisc/ for source code and patches.
Future Work

• Fix some 32-bit targets on 64-bit hosts issues. Largely with `sysctl()`.

• Add PPC support.

• Review (and most likely rewrite) `_umtx_op()` syscall shim.

• Qemu code upstream.

• Build system integration.
Q & A

Links:

- https://wiki.freebsd.org/CrossBuildingPorts
- https://wiki.freebsd.org/QemuUserModeToDo
- https://wiki.freebsd.org/QemuUserModeHowTo
- http://www.cl.cam.ac.uk/research/security/ctsr.d/mips64-packages/
- http://people.freebsd.org/~sson/imgact_binmisc/
‘binmiscctl’ Examples

- LLVM bitcode JIT compiler/interpreter (‘lli’):
  ```
  # binmiscctl add llvmbc --interpreter “/usr/bin/lli --fake-arg0=” --magic “BC\xc0\xde” --size 4 --offset 0 --concat-old-arg0 --set-enabled
  ```

- Qemu user mode emulator (‘/usr/bin/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\x00[...]” --size 20
  ```
‘binmiscctl’ Examples

- Disable|enable|delete image activator:
  # binmiscctl disable|enable|delete llvmbc

- Lookup and list image activator:
  # binmiscctl lookup llvmbc

- List all image activators:
  # binmiscctl list-all