QEMU as a Test Target

QEMU is a portable open source emulator for x86, MIPS, ARM, PPC, and Sparc. For the purposes of QNX, this makes a handy tool for quick tests and debugging. Some supported features are snapshots, basic networking, and gdb integration. Eventually, it should be possible to debug the kernel much like you can do with VMware, but for all architectures supported by QNX except for SuperHitachi. The port is still a work in progress, and experimental, but most major problems have been worked out.

Advantages over a virtual machine like VMWare is that QEMU has a Neutrino port, and can emulate many targets. There are advantages to using Neutrino as a development platform rather than cross-development, and QEMU is currently the only emulator that runs on Neutrino. QEMU can be used when a test machine is not available or practical, or for running dangerous tests that may lock the system or alter important files. The snapshot feature is extremely useful for this. A system can be restored to a previous state quickly, or saved after boot for a fast startup.

QEMU now has it's own project page, and so the guides have been moved there.#

See the QEMU Community Project Wiki#

Kernel debugging with QEMU#

Debugging the kernel is very easy with QEMU. First see Debugging The Kernel for a general guide. This method has been tested using Linux and Neutrino hosts, but should also work with Windows (perhaps with minor modifications).

The following instructions are for x86 targets. For a working ARM target, see the Beagle Board page.

Step 1 - the buildfile#

Use a minimal buildfile, or the existing qnxbasedma.build, and put something like this as the boot section:

```
[virtual=x86,bios +compress] boot = {
   # Reserve 64k of video memory to handle multiple video cards

   startup-bios -s64k -Dconsole -K 8250.3f8.9600 -v
   gdb_kdebug -K

   # PATH is the *safe* path for executables (confstr(_CS_PATH...))
   # LD_LIBRARY_PATH is the *safe* path for libraries (confstr(_CS_LIBPATH))
   #    i.e. This is the path searched for libs in setuid/setgid executables.
[+keeplinked]
   PATH=/proc/boot:/bin:/usr/bin:/opt/bin LD_LIBRARY_PATH=/proc/boot:/lib:/usr/lib:/lib/dll:/opt/lib procnto-instr_g -P
}
```

The -v option for startup bios and the -K option for gdb_kdebug are optional.

Step 2 - getting the image on a virtual drive#

We can then run the usual mkifs, and either overwrite the /.boot file if it is a full CD installation, or use dinit for small raw disk images. See Neutrino as a Guest in QEMU for help.

Step 3 - Running QEMU#

On invocation of QEMU, we can redirect the virtual serial port to a tcp socket like this:

```
qemu kernel-debug.img -serial tcp:localhost:5678,server,nowait,nodelay
```

The "server" option tells QEMU to wait for a connection before starting, except when "nowait" is specified. The "nodelay" option disables output buffering. All three of these options are required for gdb to communicate properly with qemu.

For a second serial device, simply add another "-serial" using a separate port, like so:

```
qemu kernel-debug.img -serial tcp:localhost:5678,server,nowait,nodelay -serial tcp:localhost:5679,server,nowait,nodelay
```

Step 4 - Running gdb#

We can now run gdb on procnto-instr_g.sym that was generated from mkifs:

```
ntox86-gdb procnto-instr_g.sym
(gdb) target remote localhost:5678
```

That's all. Happy debugging!#