QEMU

From OSDev Wiki

Contents

Features

- two operating modes: full system emulation (which interests us) and Linux user process emulation (which interests other people ;) and is a NxM platform emulator (multiple host, multiple targets).
- it is faster than Bochs because it uses 'just in time' code compilation technique (allowing reuse of previous interpretation)
- lacks technical documentation so far (imho), which probably makes it less suitable for "baby steps".
- provides native GDB support and you can attach it to GDB/DDD by adding the "-s -S" switches to the command line and from the GDB window start the debugging session with "target remote :1234" if QEMU is waiting on local port 1234.
- support VBE 2.0. This can be checked if you use the GRUB command line and type vbeprobe. The test returns:

Supported VBE modes

0x101	Packed pixel	640x480x8

0x110	Direct Color	640x480x15
0x111	Direct Color	640x480x16
0x112	Direct Color	640x480x24
0x103	Packed pixel	800x600x8
0x113	Direct Color	800x600x15
0x114	Direct Color	800x600x16
0x115	Direct Color	800x600x24
0x105	Packed pixel	1024x768x8
0x116	Direct Color	1024x768x15
0x117	Direct Color	1024x768x16
0x118	Direct Color	1024x768x24
0x107	Packed pixel	1024x768x8
0x119	Direct Color	1024x768x15
0x11A	Direct Color	1024x768x16

Supported Architectures

- X86
- x86_64
- ARM
- Sparc
- PowerPC
- MIPS

SPARC64, PowerPC64, m68k and SH-4 are in development.

Supported Devices

- built in NE2000 support
- PCI SVGA card (Cirrus Logic 5446)
- PCI support (With BIOS32).

Usage

QEMU is easy to use, it does not have a configuration script like Bochs. To use QEMU with your OS,

```
qemu -L .\ -fda my_disk_image.img -m 32
```

Or, if you use UNIX,

```
qemu -fda my_disk_image.img -m 32
```

The -L tells QEMU where to find its BIOS images, which is not necessary in a standard unix installation. The -m tells how many megabytes of memory to use; the default is 128

You can use -fda/-fdb for disk image files, and -hda/-hdb/-hdc/-hdd for hard disks. To change boot devices, use -boot {a/b/c/d}. a/b tell it to boot floppy a or b. c for hard disk and d for CDROM.

Alternatively you can point -hdc or use -cdrom to an ISO image file (2048 bytes per sector ISO format).

Whilst inside the emulator you can use CTRL-ALT-{1,2,3} to swap in/out of the emulation screen, the QEMU console and a serial console. The system console lets you change disk images and other things and do memory dumps etc.

The QEMU Console

It is possible to communicate with the QEMU console like one communicates with the Bochs console. However, this does not occur through port 0xE9 (which may be achieved by patching the source code however), but by using a serial port. ReactOS uses this to dump warning messages in the QEMU console. See the ReactOS wiki (http://www.reactos.org/wiki/QEMU#Redirect_to_the_console) for details.

The QEMU monitor

When you hit CTRL-ALT-2 you are placed in the QEMU monitor which is a command-line for querying information about the system while it is running. It does not quite act as a debugger, but, used in combination with GDB-stub, you can get pretty much all the functionality you will need. In Unix hosts, you can even redirect this monitor interface to the standard output using the

```
-monitor stdio
```

command-line option. Some useful commands:

xp
: eXamine Physical memory. Much like GDB's `x` command, but with no address

translation.

cpu n
: switch to CPU n. Note that GDB's threads are numbered from 1, but QEMU's CPUs are numbered from 0.

info registers
: dump register state

info tlb
: Show virtual memory translation state.

info mem
: Show the page table mappings in a compact form.

help
: List all commands -- keep in mind that there may be more commands available than those mentioned in the QEMU documentation (http://www.qemu.org/qemu-doc.html#SEC12) .

GDB-stub

When you supply the `-s` command line option, QEMU will listen on port localhost:1234 for a connection by GDB. If you also supply the `-S` command line option, then QEMU will start as if you set a breakpoint at time zero, and you will need to use the GDB command "continue" to actually begin the simulation.

For convenience, I put a file called `.gdbinit` in the current working directory which automatically runs certain commands when you start GDB without the `-n` option. For example:

```
file <my-kernel-binary>
target remote localhost:1234
```

will load into GDB your kernel and then connect to QEMU. Be sure to compile your kernel with the GCC option `-g` for debugging symbols. Now you may debug your kernel as a C program. If you have an SMP kernel, check out the `info threads` and `thread` commands. It is also possible to use the QEMU monitor and its commands using the `monitor` command in GDB. For a list of available commands and their description, check out `monitor help`.

Getting detailed logs

Most of the QEMU source code has commented lines of the form:

```
// #define DEBUG_*
```

If you are willing to edit and recompile QEMU, then you can get a good deal of debugging info output to stdout by uncommenting those lines at the top of the files that implement the pieces of the simulation you need more info about.

See Also

External Links

- QEMU homepage (http://www.qemu.org/)
- QEMU forum (http://qemu-forum.ipi.fi/)
- Q - QEMU on Mac OS X (http://www.kju-app.org/kju/)