QEMU Emulator
User Documentation

Table of Contents

1 Introduction

1.1 Features

QEMU is a FAST! processor emulator using dynamic translation to achieve good emulation speed.

QEMU has two operating modes:

- Full system emulation. In this mode, QEMU emulates a full system (for example a PC), including one or several processors and various peripherals. It can be used to launch different Operating Systems without rebooting the PC or to debug system code.
- User mode emulation. In this mode, QEMU can launch processes compiled for one CPU on another CPU. It can be used to launch the Wine Windows API emulator (`http://www.winehq.org`) or to ease cross-compilation and cross-debugging.

QEMU can run without an host kernel driver and yet gives acceptable performance.

For system emulation, the following hardware targets are supported:

- PC (x86 or x86_64 processor)
- ISA PC (old style PC without PCI bus)
- PREP (PowerPC processor)
- G3 BW PowerMac (PowerPC processor)
- Mac99 PowerMac (PowerPC processor, in progress)
- Sun4m (32-bit Sparc processor)
- Sun4u (64-bit Sparc processor, in progress)
- Malta board (32-bit and 64-bit MIPS processors)
- ARM Integrator/CP (ARM)
- ARM Versatile baseboard (ARM)
- ARM RealView Emulation baseboard (ARM)
- Spitz, Akita, Borzoi and Terrier PDAs (PXA270 processor)
- Luminary Micro LM3S811EVB (ARM Cortex-M3)
- Luminary Micro LM3S6965EVB (ARM Cortex-M3)
- Freescale MCF5208EVB (ColdFire V2).
- Arnewsh MCF5206 evaluation board (ColdFire V2).
- Palm Tungsten|E PDA (OMAP310 processor)

For user emulation, x86, PowerPC, ARM, 32-bit MIPS, Sparc32/64 and ColdFire(m68k) CPUs are supported.

2 Installation

If you want to compile QEMU yourself, see Chapter 6 [compilation], page 39.

2.1 Linux

If a precompiled package is available for your distribution - you just have to install it. Otherwise, see Chapter 6 [compilation], page 39.

2.2 Windows

Download the experimental binary installer at `http://www.free.oszoo.org/download.html`.

2.3 Mac OS X

Download the experimental binary installer at `http://www.free.oszoo.org/download.html`.

3 QEMU PC System emulator

3.1 Introduction

The QEMU PC System emulator simulates the following peripherals:

- i440FX host PCI bridge and PIIX3 PCI to ISA bridge
- Cirrus CLGD 5446 PCI VGA card or dummy VGA card with Bochs VESA extensions (hardware level, including all non standard modes).
- PS/2 mouse and keyboard
- 2 PCI IDE interfaces with hard disk and CD-ROM support
- Floppy disk
- PCI/ISA PCI network adapters
- Serial ports
- Creative SoundBlaster 16 sound card
- ENSONIQ AudioPCI ES1370 sound card
- Adlib(OPL2) - Yamaha YM3812 compatible chip
- PCI UHCI USB controller and a virtual USB hub.

SMP is supported with up to 255 CPUs.

Note that adlib is only available when QEMU was configured with -enable-adlib

QEMU uses the PC BIOS from the Bochs project and the Plex86/Bochs LGPL VGA BIOS.

QEMU uses YM3812 emulation by Tatsuyuki Satoh.

3.2 Quick Start

Download and uncompress the linux image (‘`linux.img`’) and type:

```
qemu linux.img
```

Linux should boot and give you a prompt.

3.3 Invocation

`usage: qemu [options] [`*`disk_image`*`]`

disk_image is a raw hard disk image for IDE hard disk 0.

General options:

‘`-M` *`machine`*’
: Select the emulated *machine* (`-M ?` for list)

‘`-fda` *`file`*’
‘`-fdb` *`file`*’
: Use *file* as floppy disk 0/1 image (see Section 3.6 [disk_images], page 18). You can use the host floppy by using ‘`/dev/fd0`’ as filename (see Section 3.6.5 [host_drives], page 21).

‘`-hda` *file*’
‘`-hdb` *file*’
‘`-hdc` *file*’
‘`-hdd` *file*’
Use *file* as hard disk 0, 1, 2 or 3 image (see Section 3.6 [disk_images], page 18).

‘`-cdrom` *file*’
Use *file* as CD-ROM image (you cannot use ‘`-hdc`’ and ‘`-cdrom`’ at the same time). You can use the host CD-ROM by using ‘`/dev/cdrom`’ as filename (see Section 3.6.5 [host_drives], page 21).

‘`-drive` *option*`[,`*option*`[,`*option*`[,...]]]`’
Define a new drive. Valid options are:

`file=`*file*
This option defines which disk image (see Section 3.6 [disk_images], page 18) to use with this drive.

`if=`*interface*
This option defines on which type on interface the drive is connected. Available types are: ide, scsi, sd, mtd, floppy, pflash.

`bus=`*bus*`,unit=`*unit*
These options define where is connected the drive by defining the bus number and the unit id.

`index=`*index*
This option defines where is connected the drive by using an index in the list of available connectors of a given interface type.

`media=`*media*
This option defines the type of the media: disk or cdrom.

`cyls=`*c*`,heads=`*h*`,secs=`*s*`[,trans=`*t*`]`
These options have the same definition as they have in ‘`-hdachs`’.

`snapshot=`*snapshot*
snapshot is "on" or "off" and allows to enable snapshot for given drive (see ‘`-snapshot`’).

`cache=`*cache*
cache is "on" or "off" and allows to disable host cache to access data.

Instead of ‘`-cdrom`’ you can use:

```
qemu -drive file=file,index=2,media=cdrom
```

Instead of ‘`-hda`’, ‘`-hdb`’, ‘`-hdc`’, ‘`-hdd`’, you can use:

```
qemu -drive file=file,index=0,media=disk
qemu -drive file=file,index=1,media=disk
qemu -drive file=file,index=2,media=disk
qemu -drive file=file,index=3,media=disk
```

You can connect a CDROM to the slave of ide0:

```
qemu -drive file=file,if=ide,index=1,media=cdrom
```

If you don't specify the "file=" argument, you define an empty drive:

```
qemu -drive if=ide,index=1,media=cdrom
```

You can connect a SCSI disk with unit ID 6 on the bus #0:

```
qemu -drive file=file,if=scsi,bus=0,unit=6
```

Instead of '`-fda`', '`-fdb`', you can use:

```
qemu -drive file=file,index=0,if=floppy
qemu -drive file=file,index=1,if=floppy
```

By default, *interface* is "ide" and *index* is automatically incremented:

```
qemu -drive file=a -drive file=b"
```

is interpreted like:

```
qemu -hda a -hdb b
```

'`-boot [a|c|d|n]`'
: Boot on floppy (a), hard disk (c), CD-ROM (d), or Etherboot (n). Hard disk boot is the default.

'`-snapshot`'
: Write to temporary files instead of disk image files. In this case, the raw disk image you use is not written back. You can however force the write back by pressing ⟨C-a s⟩ (see Section 3.6 [disk_images], page 18).

'`-no-fd-bootchk`'
: Disable boot signature checking for floppy disks in Bochs BIOS. It may be needed to boot from old floppy disks.

'`-m` *megs*' Set virtual RAM size to *megs* megabytes. Default is 128 MiB.

'`-smp` *n*' Simulate an SMP system with *n* CPUs. On the PC target, up to 255 CPUs are supported. On Sparc32 target, Linux limits the number of usable CPUs to 4.

'`-audio-help`'
: Will show the audio subsystem help: list of drivers, tunable parameters.

'`-soundhw` *card1*`[,`*card2*`,...] or -soundhw all`'
: Enable audio and selected sound hardware. Use ? to print all available sound hardware.

```
qemu -soundhw sb16,adlib hda
qemu -soundhw es1370 hda
qemu -soundhw all hda
qemu -soundhw ?
```

'`-localtime`'
: Set the real time clock to local time (the default is to UTC time). This option is needed to have correct date in MS-DOS or Windows.

'`-startdate` *date*'
: Set the initial date of the real time clock. Valid format for *date* are: `now` or `2006-06-17T16:01:21` or `2006-06-17`. The default value is `now`.

‘`-pidfile` *file*’
: Store the QEMU process PID in *file*. It is useful if you launch QEMU from a script.

‘`-daemonize`’
: Daemonize the QEMU process after initialization. QEMU will not detach from standard IO until it is ready to receive connections on any of its devices. This option is a useful way for external programs to launch QEMU without having to cope with initialization race conditions.

‘`-win2k-hack`’
: Use it when installing Windows 2000 to avoid a disk full bug. After Windows 2000 is installed, you no longer need this option (this option slows down the IDE transfers).

‘`-option-rom` *file*’
: Load the contents of *file* as an option ROM. This option is useful to load things like EtherBoot.

‘`-name` *name*’
: Sets the *name* of the guest. This name will be display in the SDL window caption. The *name* will also be used for the VNC server.

Display options:

‘`-nographic`’
: Normally, QEMU uses SDL to display the VGA output. With this option, you can totally disable graphical output so that QEMU is a simple command line application. The emulated serial port is redirected on the console. Therefore, you can still use QEMU to debug a Linux kernel with a serial console.

‘`-no-frame`’
: Do not use decorations for SDL windows and start them using the whole available screen space. This makes the using QEMU in a dedicated desktop workspace more convenient.

‘`-full-screen`’
: Start in full screen.

‘`-vnc` *display*`[,`*option*`[,`*option*`[,...]]]`’
: Normally, QEMU uses SDL to display the VGA output. With this option, you can have QEMU listen on VNC display *display* and redirect the VGA display over the VNC session. It is very useful to enable the usb tablet device when using this option (option ‘`-usbdevice tablet`’). When using the VNC display, you must use the ‘`-k`’ parameter to set the keyboard layout if you are not using en-us. Valid syntax for the *display* is

  interface`:`*d*
  : TCP connections will only be allowed from *interface* on display *d*. By convention the TCP port is 5900+*d*. Optionally, *interface* can be omitted in which case the server will bind to all interfaces.

unix:*path*
: Connections will be allowed over UNIX domain sockets where *path* is the location of a unix socket to listen for connections on.

none
: VNC is initialized but not started. The monitor `change` command can be used to later start the VNC server.

Following the *display* value there may be one or more *option* flags separated by commas. Valid options are

password
: Require that password based authentication is used for client connections. The password must be set separately using the `change` command in the Section 3.5 [pcsys_monitor], page 14

tls
: Require that client use TLS when communicating with the VNC server. This uses anonymous TLS credentials so is susceptible to a man-in-the-middle attack. It is recommended that this option be combined with either the *x509* or *x509verify* options.

x509=*/path/to/certificate/dir*
: Valid if '`tls`' is specified. Require that x509 credentials are used for negotiating the TLS session. The server will send its x509 certificate to the client. It is recommended that a password be set on the VNC server to provide authentication of the client when this is used. The path following this option specifies where the x509 certificates are to be loaded from. See the Section 3.10 [vnc_security], page 25 section for details on generating certificates.

x509verify=*/path/to/certificate/dir*
: Valid if '`tls`' is specified. Require that x509 credentials are used for negotiating the TLS session. The server will send its x509 certificate to the client, and request that the client send its own x509 certificate. The server will validate the client's certificate against the CA certificate, and reject clients when validation fails. If the certificate authority is trusted, this is a sufficient authentication mechanism. You may still wish to set a password on the VNC server as a second authentication layer. The path following this option specifies where the x509 certificates are to be loaded from. See the Section 3.10 [vnc_security], page 25 section for details on generating certificates.

'`-k` *language*'
: Use keyboard layout *language* (for example `fr` for French). This option is only needed where it is not easy to get raw PC keycodes (e.g. on Macs, with some X11 servers or with a VNC display). You don't normally need to use it on PC/Linux or PC/Windows hosts.

  The available layouts are:

```
ar  de-ch  es  fo     fr-ca  hu  ja  mk     no  pt-br  sv
da  en-gb  et  fr     fr-ch  is  lt  nl     pl  ru     th
de  en-us  fi  fr-be  hr     it  lv  nl-be  pt  sl     tr
```

The default is `en-us`.

USB options:

'`-usb`' Enable the USB driver (will be the default soon)

'`-usbdevice` *devname*'
Add the USB device *devname*. See Section 3.9.1 [usb_devices], page 24.

Network options:

'`-net nic[,vlan=`*n*`][,macaddr=`*addr*`][,model=`*type*`]`'
Create a new Network Interface Card and connect it to VLAN *n* (*n* = 0 is the default). The NIC is an ne2k_pci by default on the PC target. Optionally, the MAC address can be changed. If no '`-net`' option is specified, a single NIC is created. Qemu can emulate several different models of network card. Valid values for *type* are `i82551`, `i82557b`, `i82559er`, `ne2k_pci`, `ne2k_isa`, `pcnet`, `rtl8139`, `smc91c111`, `lance` and `mcf_fec`. Not all devices are supported on all targets. Use -net nic,model=? for a list of available devices for your target.

'`-net user[,vlan=`*n*`][,hostname=`*name*`]`'
Use the user mode network stack which requires no administrator privilege to run. '`hostname=name`' can be used to specify the client hostname reported by the builtin DHCP server.

'`-net tap[,vlan=`*n*`][,fd=`*h*`][,ifname=`*name*`][,script=`*file*`]`'
Connect the host TAP network interface *name* to VLAN *n* and use the network script *file* to configure it. The default network script is '`/etc/qemu-ifup`'. Use '`script=no`' to disable script execution. If *name* is not provided, the OS automatically provides one. '`fd`'=*h* can be used to specify the handle of an already opened host TAP interface. Example:

```
qemu linux.img -net nic -net tap
```

More complicated example (two NICs, each one connected to a TAP device)

```
qemu linux.img -net nic,vlan=0 -net tap,vlan=0,ifname=tap0 \
               -net nic,vlan=1 -net tap,vlan=1,ifname=tap1
```

'`-net socket[,vlan=`*n*`][,fd=`*h*`][,listen=[`*host*`]:`*port*`][,connect=`*host*`:`*port*`]`'
Connect the VLAN *n* to a remote VLAN in another QEMU virtual machine using a TCP socket connection. If '`listen`' is specified, QEMU waits for incoming connections on *port* (*host* is optional). '`connect`' is used to connect to another QEMU instance using the '`listen`' option. '`fd`'=*h* specifies an already opened TCP socket.

Example:

```
# launch a first QEMU instance
qemu linux.img -net nic,macaddr=52:54:00:12:34:56 \
               -net socket,listen=:1234
# connect the VLAN 0 of this instance to the VLAN 0
```

```
# of the first instance
qemu linux.img -net nic,macaddr=52:54:00:12:34:57 \
               -net socket,connect=127.0.0.1:1234
```

'`-net socket[,vlan=n][,fd=h][,mcast=maddr:port]`'

Create a VLAN *n* shared with another QEMU virtual machines using a UDP multicast socket, effectively making a bus for every QEMU with same multicast address *maddr* and *port*. NOTES:

1. Several QEMU can be running on different hosts and share same bus (assuming correct multicast setup for these hosts).
2. mcast support is compatible with User Mode Linux (argument '`ethN=mcast`'), see `http://user-mode-linux.sf.net`.
3. Use '`fd=h`' to specify an already opened UDP multicast socket.

Example:

```
# launch one QEMU instance
qemu linux.img -net nic,macaddr=52:54:00:12:34:56 \
               -net socket,mcast=230.0.0.1:1234
# launch another QEMU instance on same "bus"
qemu linux.img -net nic,macaddr=52:54:00:12:34:57 \
               -net socket,mcast=230.0.0.1:1234
# launch yet another QEMU instance on same "bus"
qemu linux.img -net nic,macaddr=52:54:00:12:34:58 \
               -net socket,mcast=230.0.0.1:1234
```

Example (User Mode Linux compat.):

```
# launch QEMU instance (note mcast address selected
# is UML's default)
qemu linux.img -net nic,macaddr=52:54:00:12:34:56 \
               -net socket,mcast=239.192.168.1:1102
# launch UML
/path/to/linux ubd0=/path/to/root_fs eth0=mcast
```

'`-net none`'

Indicate that no network devices should be configured. It is used to override the default configuration ('`-net nic -net user`') which is activated if no '`-net`' options are provided.

'`-tftp dir`'

When using the user mode network stack, activate a built-in TFTP server. The files in *dir* will be exposed as the root of a TFTP server. The TFTP client on the guest must be configured in binary mode (use the command `bin` of the Unix TFTP client). The host IP address on the guest is as usual 10.0.2.2.

'`-bootp file`'

When using the user mode network stack, broadcast *file* as the BOOTP filename. In conjunction with '`-tftp`', this can be used to network boot a guest from a local directory.

Example (using pxelinux):

```
qemu -hda linux.img -boot n -tftp /path/to/tftp/files -bootp /pxelinux.0
```

‘`-smb` *dir*’
: When using the user mode network stack, activate a built-in SMB server so that Windows OSes can access to the host files in ‘*dir*’ transparently.

  In the guest Windows OS, the line:

  ```
  10.0.2.4 smbserver
  ```

  must be added in the file ‘`C:\WINDOWS\LMHOSTS`’ (for windows 9x/Me) or ‘`C:\WINNT\SYSTEM32\DRIVERS\ETC\LMHOSTS`’ (Windows NT/2000).

  Then ‘*dir*’ can be accessed in ‘`\\smbserver\qemu`’.

  Note that a SAMBA server must be installed on the host OS in ‘`/usr/sbin/smbd`’. QEMU was tested successfully with smbd version 2.2.7a from the Red Hat 9 and version 3.0.10-1.fc3 from Fedora Core 3.

‘`-redir [tcp|udp]:`*host-port*`:[`*guest-host*`]:`*guest-port*’
: When using the user mode network stack, redirect incoming TCP or UDP connections to the host port *host-port* to the guest *guest-host* on guest port *guest-port*. If *guest-host* is not specified, its value is 10.0.2.15 (default address given by the built-in DHCP server).

  For example, to redirect host X11 connection from screen 1 to guest screen 0, use the following:

  ```
  # on the host
  qemu -redir tcp:6001::6000 [...]
  # this host xterm should open in the guest X11 server
  xterm -display :1
  ```

  To redirect telnet connections from host port 5555 to telnet port on the guest, use the following:

  ```
  # on the host
  qemu -redir tcp:5555::23 [...]
  telnet localhost 5555
  ```

  Then when you use on the host `telnet localhost 5555`, you connect to the guest telnet server.

Linux boot specific: When using these options, you can use a given Linux kernel without installing it in the disk image. It can be useful for easier testing of various kernels.

‘`-kernel` *bzImage*’
: Use *bzImage* as kernel image.

‘`-append` *cmdline*’
: Use *cmdline* as kernel command line

‘`-initrd` *file*’
: Use *file* as initial ram disk.

Debug/Expert options:

‘`-serial` *dev*’
: Redirect the virtual serial port to host character device *dev*. The default device is `vc` in graphical mode and `stdio` in non graphical mode.

This option can be used several times to simulate up to 4 serials ports.

Use `-serial none` to disable all serial ports.

Available character devices are:

`vc[:WxH]` Virtual console. Optionally, a width and height can be given in pixel with

`vc:800x600`

It is also possible to specify width or height in characters:

`vc:80Cx24C`

`pty` [Linux only] Pseudo TTY (a new PTY is automatically allocated)

`none` No device is allocated.

`null` void device

`/dev/XXX` [Linux only] Use host tty, e.g. '`/dev/ttyS0`'. The host serial port parameters are set according to the emulated ones.

`/dev/parport`*N*
[Linux only, parallel port only] Use host parallel port *N*. Currently SPP and EPP parallel port features can be used.

`file:`*filename*
Write output to *filename*. No character can be read.

`stdio` [Unix only] standard input/output

`pipe:`*filename*
name pipe *filename*

`COM`*n* [Windows only] Use host serial port *n*

`udp:[`*remote_host*`]:`*remote_port*`[@[`*src_ip*`]:`*src_port*`]`
This implements UDP Net Console. When *remote_host* or *src_ip* are not specified they default to `0.0.0.0`. When not using a specified *src_port* a random port is automatically chosen.

If you just want a simple readonly console you can use `netcat` or `nc`, by starting qemu with: `-serial udp::4555` and nc as: `nc -u -l -p 4555`. Any time qemu writes something to that port it will appear in the netconsole session.

If you plan to send characters back via netconsole or you want to stop and start qemu a lot of times, you should have qemu use the same source port each time by using something like `-serial udp::4555@:4556` to qemu. Another approach is to use a patched version of netcat which can listen to a TCP port and send and receive characters via udp. If you have a patched version of netcat which activates telnet remote echo and single char transfer, then you can use the following options to step up a netcat redirector to allow telnet on port 5555 to access the qemu port.

`Qemu Options:`
-serial udp::4555@:4556

`netcat options:`
-u -P 4555 -L 0.0.0.0:4556 -t -p 5555 -I -T

`telnet options:`
localhost 5555

`tcp:[`*host*`]:`*port*`[,`*server*`][,nowait][,nodelay]`
The TCP Net Console has two modes of operation. It can send the serial I/O to a location or wait for a connection from a location. By default the TCP Net Console is sent to *host* at the *port*. If you use the *server* option QEMU will wait for a client socket application to connect to the port before continuing, unless the `nowait` option was specified. The `nodelay` option disables the Nagle buffering algorithm. If *host* is omitted, 0.0.0.0 is assumed. Only one TCP connection at a time is accepted. You can use `telnet` to connect to the corresponding character device.

`Example to send tcp console to 192.168.0.2 port 4444`
-serial tcp:192.168.0.2:4444

`Example to listen and wait on port 4444 for connection`
-serial tcp::4444,server

`Example to not wait and listen on ip 192.168.0.100 port 4444`
-serial tcp:192.168.0.100:4444,server,nowait

`telnet:`*host*`:`*port*`[,server][,nowait][,nodelay]`
The telnet protocol is used instead of raw tcp sockets. The options work the same as if you had specified `-serial tcp`. The difference is that the port acts like a telnet server or client using telnet option negotiation. This will also allow you to send the MAGIC_SYSRQ sequence if you use a telnet that supports sending the break sequence. Typically in unix telnet you do it with Control-] and then type "send break" followed by pressing the enter key.

`unix:`*path*`[,server][,nowait]`
A unix domain socket is used instead of a tcp socket. The option works the same as if you had specified `-serial tcp` except the unix domain socket *path* is used for connections.

`mon:`*dev_string*
This is a special option to allow the monitor to be multiplexed onto another serial port. The monitor is accessed with key sequence of ⟨Control-a⟩ and then pressing ⟨c⟩. See monitor access Section 3.4 [pcsys_keys], page 14 in the -nographic section for more keys. *dev_string* should be any one of the serial devices specified above. An example to multiplex the monitor onto a telnet server listening on port 4444 would be:

```
-serial mon:telnet::4444,server,nowait
```

'`-parallel dev`'
: Redirect the virtual parallel port to host device *dev* (same devices as the serial port). On Linux hosts, '`/dev/parportN`' can be used to use hardware devices connected on the corresponding host parallel port.

  This option can be used several times to simulate up to 3 parallel ports.

  Use `-parallel none` to disable all parallel ports.

'`-monitor dev`'
: Redirect the monitor to host device *dev* (same devices as the serial port). The default device is `vc` in graphical mode and `stdio` in non graphical mode.

'`-echr numeric_ascii_value`'
: Change the escape character used for switching to the monitor when using monitor and serial sharing. The default is `0x01` when using the `-nographic` option. `0x01` is equal to pressing `Control-a`. You can select a different character from the ascii control keys where 1 through 26 map to Control-a through Control-z. For instance you could use the either of the following to change the escape character to Control-t.

  ```
  -echr 0x14
  -echr 20
  ```

'`-s`'
: Wait gdb connection to port 1234 (see Section 3.11 [gdb_usage], page 28).

'`-p port`'
: Change gdb connection port. *port* can be either a decimal number to specify a TCP port, or a host device (same devices as the serial port).

'`-S`'
: Do not start CPU at startup (you must type 'c' in the monitor).

'`-d`'
: Output log in /tmp/qemu.log

'`-hdachs c,h,s,[,t]`'
: Force hard disk 0 physical geometry (1 <= c <= 16383, 1 <= h <= 16, 1 <= s <= 63) and optionally force the BIOS translation mode (t=none, lba or auto). Usually QEMU can guess all those parameters. This option is useful for old MS-DOS disk images.

'`-L path`'
: Set the directory for the BIOS, VGA BIOS and keymaps.

'`-std-vga`'
: Simulate a standard VGA card with Bochs VBE extensions (default is Cirrus Logic GD5446 PCI VGA). If your guest OS supports the VESA 2.0 VBE extensions (e.g. Windows XP) and if you want to use high resolution modes (>= 1280x1024x16) then you should use this option.

'`-no-acpi`'
: Disable ACPI (Advanced Configuration and Power Interface) support. Use it if your guest OS complains about ACPI problems (PC target machine only).

'`-no-reboot`'
: Exit instead of rebooting.

‘`-loadvm file`’
: Start right away with a saved state (`loadvm` in monitor)

‘`-semihosting`’
: Enable semihosting syscall emulation (ARM and M68K target machines only).

  On ARM this implements the "Angel" interface. On M68K this implements the "ColdFire GDB" interface used by libgloss.

  Note that this allows guest direct access to the host filesystem, so should only be used with trusted guest OS.

3.4 Keys

During the graphical emulation, you can use the following keys:

⟨Ctrl-Alt-f⟩ Toggle full screen

⟨Ctrl-Alt-n⟩ Switch to virtual console ’n’. Standard console mappings are:

1	Target system display
2	Monitor
3	Serial port

⟨Ctrl-Alt⟩ Toggle mouse and keyboard grab.

In the virtual consoles, you can use ⟨Ctrl-Up⟩, ⟨Ctrl-Down⟩, ⟨Ctrl-PageUp⟩ and ⟨Ctrl-PageDown⟩ to move in the back log.

During emulation, if you are using the ‘`-nographic`’ option, use ⟨Ctrl-a h⟩ to get terminal commands:

⟨Ctrl-a h⟩ Print this help

⟨Ctrl-a x⟩ Exit emulator

⟨Ctrl-a s⟩ Save disk data back to file (if -snapshot)

⟨Ctrl-a t⟩ toggle console timestamps

⟨Ctrl-a b⟩ Send break (magic sysrq in Linux)

⟨Ctrl-a c⟩ Switch between console and monitor

⟨Ctrl-a Ctrl-a⟩
: Send Ctrl-a

3.5 QEMU Monitor

The QEMU monitor is used to give complex commands to the QEMU emulator. You can use it to:

- Remove or insert removable media images (such as CD-ROM or floppies).
- Freeze/unfreeze the Virtual Machine (VM) and save or restore its state from a disk file.
- Inspect the VM state without an external debugger.

3.5.1 Commands

The following commands are available:

‘`help or ? [cmd]`’
Show the help for all commands or just for command *cmd*.

‘`commit`’ Commit changes to the disk images (if -snapshot is used).

‘`info subcommand`’
Show various information about the system state.

‘`info network`’
show the various VLANs and the associated devices

‘`info block`’
show the block devices

‘`info registers`’
show the cpu registers

‘`info history`’
show the command line history

‘`info pci`’ show emulated PCI device

‘`info usb`’ show USB devices plugged on the virtual USB hub

‘`info usbhost`’
show all USB host devices

‘`info capture`’
show information about active capturing

‘`info snapshots`’
show list of VM snapshots

‘`info mice`’
show which guest mouse is receiving events

‘`q or quit`’
Quit the emulator.

‘`eject [-f] device`’
Eject a removable medium (use -f to force it).

‘`change device setting`’
Change the configuration of a device.

‘`change diskdevice filename`’
Change the medium for a removable disk device to point to *filename*. eg

```
(qemu) change cdrom /path/to/some.iso
```

‘`change vnc display,options`’
Change the configuration of the VNC server. The valid syntax for *display* and *options* are described at Section 3.3 [sec_invocation], page 3. eg

```
(qemu) change vnc localhost:1
```

'`change vnc password`'
: Change the password associated with the VNC server. The monitor will prompt for the new password to be entered. VNC passwords are only significant upto 8 letters. eg.

```
(qemu) change vnc password
Password: ********
```

'`screendump` *`filename`*'
: Save screen into PPM image *filename*.

'`mouse_move` *`dx dy [dz]`*'
: Move the active mouse to the specified coordinates *dx* *dy* with optional scroll axis *dz*.

'`mouse_button` *`val`*'
: Change the active mouse button state *val* (1=L, 2=M, 4=R).

'`mouse_set` *`index`*'
: Set which mouse device receives events at given *index*, index can be obtained with

```
info mice
```

'`wavcapture` *`filename`* [*`frequency`* [*`bits`* [*`channels`*]]]'
: Capture audio into *filename*. Using sample rate *frequency* bits per sample *bits* and number of channels *channels*.

  Defaults:

  - Sample rate = 44100 Hz - CD quality
  - Bits = 16
  - Number of channels = 2 - Stereo

'`stopcapture` *`index`*'
: Stop capture with a given *index*, index can be obtained with

```
info capture
```

'`log` *`item1`*`[,...]`'
: Activate logging of the specified items to '`/tmp/qemu.log`'.

'`savevm [`*`tag`*`|`*`id`*`]`'
: Create a snapshot of the whole virtual machine. If *tag* is provided, it is used as human readable identifier. If there is already a snapshot with the same tag or ID, it is replaced. More info at Section 3.6.3 [vm_snapshots], page 18.

'`loadvm` *`tag`*`|`*`id`*'
: Set the whole virtual machine to the snapshot identified by the tag *tag* or the unique snapshot ID *id*.

'`delvm` *`tag`*`|`*`id`*'
: Delete the snapshot identified by *tag* or *id*.

'`stop`' Stop emulation.

‘`c or cont`’
Resume emulation.

‘`gdbserver [port]`’
Start gdbserver session (default *port*=1234)

‘`x/fmt addr`’
Virtual memory dump starting at *addr*.

‘`xp /fmt addr`’
Physical memory dump starting at *addr*.

fmt is a format which tells the command how to format the data. Its syntax is: ‘`/{count}{format}{size}`’

count is the number of items to be dumped.

format can be x (hex), d (signed decimal), u (unsigned decimal), o (octal), c (char) or i (asm instruction).

size can be b (8 bits), h (16 bits), w (32 bits) or g (64 bits). On x86, `h` or `w` can be specified with the `i` format to respectively select 16 or 32 bit code instruction size.

Examples:

- Dump 10 instructions at the current instruction pointer:

```
(qemu) x/10i $eip
0x90107063:  ret
0x90107064:  sti
0x90107065:  lea    0x0(%esi,1),%esi
0x90107069:  lea    0x0(%edi,1),%edi
0x90107070:  ret
0x90107071:  jmp    0x90107080
0x90107073:  nop
0x90107074:  nop
0x90107075:  nop
0x90107076:  nop
```

- Dump 80 16 bit values at the start of the video memory.

```
(qemu) xp/80hx 0xb8000
0x000b8000: 0x0b50 0x0b6c 0x0b65 0x0b78 0x0b38 0x0b36 0x0b2f 0x0b42
0x000b8010: 0x0b6f 0x0b63 0x0b68 0x0b73 0x0b20 0x0b56 0x0b47 0x0b41
0x000b8020: 0x0b42 0x0b69 0x0b6f 0x0b73 0x0b20 0x0b63 0x0b75 0x0b72
0x000b8030: 0x0b72 0x0b65 0x0b6e 0x0b74 0x0b2d 0x0b63 0x0b76 0x0b73
0x000b8040: 0x0b20 0x0b30 0x0b35 0x0b20 0x0b4e 0x0b6f 0x0b76 0x0b20
0x000b8050: 0x0b32 0x0b30 0x0b30 0x0b33 0x0720 0x0720 0x0720 0x0720
0x000b8060: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
0x000b8070: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
0x000b8080: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
0x000b8090: 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720 0x0720
```

‘`p or print/fmt expr`’
Print expression value. Only the *format* part of *fmt* is used.

‘`sendkey keys`’
Send *keys* to the emulator. Use - to press several keys simultaneously. Example:

`sendkey ctrl-alt-f1`

This command is useful to send keys that your graphical user interface intercepts at low level, such as `ctrl-alt-f1` in X Window.

'`system_reset`'
Reset the system.

'`usb_add` *devname*'
Add the USB device *devname*. For details of available devices see Section 3.9.1 [usb_devices], page 24

'`usb_del` *devname*'
Remove the USB device *devname* from the QEMU virtual USB hub. *devname* has the syntax `bus.addr`. Use the monitor command `info usb` to see the devices you can remove.

3.5.2 Integer expressions

The monitor understands integers expressions for every integer argument. You can use register names to get the value of specifics CPU registers by prefixing them with *$*.

3.6 Disk Images

Since version 0.6.1, QEMU supports many disk image formats, including growable disk images (their size increase as non empty sectors are written), compressed and encrypted disk images. Version 0.8.3 added the new qcow2 disk image format which is essential to support VM snapshots.

3.6.1 Quick start for disk image creation

You can create a disk image with the command:

`qemu-img create myimage.img mysize`

where *myimage.img* is the disk image filename and *mysize* is its size in kilobytes. You can add an `M` suffix to give the size in megabytes and a `G` suffix for gigabytes.

See Section 3.6.4 [qemu_img_invocation], page 19 for more information.

3.6.2 Snapshot mode

If you use the option '`-snapshot`', all disk images are considered as read only. When sectors in written, they are written in a temporary file created in '`/tmp`'. You can however force the write back to the raw disk images by using the `commit` monitor command (or ⟨C-a s⟩ in the serial console).

3.6.3 VM snapshots

VM snapshots are snapshots of the complete virtual machine including CPU state, RAM, device state and the content of all the writable disks. In order to use VM snapshots, you must have at least one non removable and writable block device using the `qcow2` disk image format. Normally this device is the first virtual hard drive.

Use the monitor command `savevm` to create a new VM snapshot or replace an existing one. A human readable name can be assigned to each snapshot in addition to its numerical ID.

Use `loadvm` to restore a VM snapshot and `delvm` to remove a VM snapshot. `info snapshots` lists the available snapshots with their associated information:

```
(qemu) info snapshots
Snapshot devices: hda
Snapshot list (from hda):
ID        TAG                 VM SIZE                DATE       VM CLOCK
1         start                   41M 2006-08-06 12:38:02   00:00:14.954
2                                 40M 2006-08-06 12:43:29   00:00:18.633
3         msys                    40M 2006-08-06 12:44:04   00:00:23.514
```

A VM snapshot is made of a VM state info (its size is shown in `info snapshots`) and a snapshot of every writable disk image. The VM state info is stored in the first `qcow2` non removable and writable block device. The disk image snapshots are stored in every disk image. The size of a snapshot in a disk image is difficult to evaluate and is not shown by `info snapshots` because the associated disk sectors are shared among all the snapshots to save disk space (otherwise each snapshot would need a full copy of all the disk images).

When using the (unrelated) `-snapshot` option (Section 3.6.2 [disk_images_snapshot_mode], page 18), you can always make VM snapshots, but they are deleted as soon as you exit QEMU.

VM snapshots currently have the following known limitations:

- They cannot cope with removable devices if they are removed or inserted after a snapshot is done.
- A few device drivers still have incomplete snapshot support so their state is not saved or restored properly (in particular USB).

3.6.4 `qemu-img` Invocation

```
usage: qemu-img command [command options]
```

The following commands are supported:

'`create [-e] [-6] [-b` *base_image*`] [-f` *fmt*`]` *filename* `[`*size*`]`'
'`commit [-f` *fmt*`]` *filename*'
'`convert [-c] [-e] [-6] [-f` *fmt*`]` *filename* `[-O` *output_fmt*`]` *output_filename*'
'`info [-f` *fmt*`]` *filename*'

Command parameters:

filename is a disk image filename

base_image
: is the read-only disk image which is used as base for a copy on write image; the copy on write image only stores the modified data

fmt is the disk image format. It is guessed automatically in most cases. The following formats are supported:

`raw`
: Raw disk image format (default). This format has the advantage of being simple and easily exportable to all other emulators. If your file system supports *holes* (for example in ext2 or ext3 on Linux

or NTFS on Windows), then only the written sectors will reserve space. Use `qemu-img info` to know the real size used by the image or `ls -ls` on Unix/Linux.

`qcow2` QEMU image format, the most versatile format. Use it to have smaller images (useful if your filesystem does not supports holes, for example on Windows), optional AES encryption, zlib based compression and support of multiple VM snapshots.

`qcow` Old QEMU image format. Left for compatibility.

`cow` User Mode Linux Copy On Write image format. Used to be the only growable image format in QEMU. It is supported only for compatibility with previous versions. It does not work on win32.

`vmdk` VMware 3 and 4 compatible image format.

`cloop` Linux Compressed Loop image, useful only to reuse directly compressed CD-ROM images present for example in the Knoppix CD-ROMs.

size is the disk image size in kilobytes. Optional suffixes `M` (megabyte) and `G` (gigabyte) are supported

output_filename
is the destination disk image filename

output_fmt
is the destination format

-c indicates that target image must be compressed (qcow format only)

-e indicates that the target image must be encrypted (qcow format only)

-6 indicates that the target image must use compatibility level 6 (vmdk format only)

Command description:

‘`create [-6] [-e] [-b base_image] [-f fmt] filename [size]`’
Create the new disk image *filename* of size *size* and format *fmt*.

If *base_image* is specified, then the image will record only the differences from *base_image*. No size needs to be specified in this case. *base_image* will never be modified unless you use the `commit` monitor command.

‘`commit [-f fmt] filename`’
Commit the changes recorded in *filename* in its base image.

‘`convert [-c] [-e] [-f fmt] filename [-O output_fmt] output_filename`’
Convert the disk image *filename* to disk image *output_filename* using format *output_fmt*. It can be optionally encrypted (`-e` option) or compressed (`-c` option).

Only the format `qcow` supports encryption or compression. The compression is read-only. It means that if a compressed sector is rewritten, then it is rewritten as uncompressed data.

Encryption uses the AES format which is very secure (128 bit keys). Use a long password (16 characters) to get maximum protection.

Image conversion is also useful to get smaller image when using a growable format such as `qcow` or `cow`: the empty sectors are detected and suppressed from the destination image.

'`info [-f fmt] filename`'
Give information about the disk image *filename*. Use it in particular to know the size reserved on disk which can be different from the displayed size. If VM snapshots are stored in the disk image, they are displayed too.

3.6.5 Using host drives

In addition to disk image files, QEMU can directly access host devices. We describe here the usage for QEMU version >= 0.8.3.

3.6.5.1 Linux

On Linux, you can directly use the host device filename instead of a disk image filename provided you have enough privileges to access it. For example, use '`/dev/cdrom`' to access to the CDROM or '`/dev/fd0`' for the floppy.

`CD` You can specify a CDROM device even if no CDROM is loaded. QEMU has specific code to detect CDROM insertion or removal. CDROM ejection by the guest OS is supported. Currently only data CDs are supported.

`Floppy` You can specify a floppy device even if no floppy is loaded. Floppy removal is currently not detected accurately (if you change floppy without doing floppy access while the floppy is not loaded, the guest OS will think that the same floppy is loaded).

`Hard disks`
Hard disks can be used. Normally you must specify the whole disk ('`/dev/hdb`' instead of '`/dev/hdb1`') so that the guest OS can see it as a partitioned disk. WARNING: unless you know what you do, it is better to only make READ-ONLY accesses to the hard disk otherwise you may corrupt your host data (use the '`-snapshot`' command line option or modify the device permissions accordingly).

3.6.5.2 Windows

`CD` The preferred syntax is the drive letter (e.g. '`d:`'). The alternate syntax '`\\.\d:`' is supported. '`/dev/cdrom`' is supported as an alias to the first CDROM drive.

Currently there is no specific code to handle removable media, so it is better to use the `change` or `eject` monitor commands to change or eject media.

`Hard disks`
Hard disks can be used with the syntax: '`\\.\PhysicalDrive`*N*' where *N* is the drive number (0 is the first hard disk).

WARNING: unless you know what you do, it is better to only make READ-ONLY accesses to the hard disk otherwise you may corrupt your host data

(use the '`-snapshot`' command line so that the modifications are written in a temporary file).

3.6.5.3 Mac OS X

'`/dev/cdrom`' is an alias to the first CDROM.

Currently there is no specific code to handle removable media, so it is better to use the `change` or `eject` monitor commands to change or eject media.

3.6.6 Virtual FAT disk images

QEMU can automatically create a virtual FAT disk image from a directory tree. In order to use it, just type:

```
qemu linux.img -hdb fat:/my_directory
```

Then you access access to all the files in the '`/my_directory`' directory without having to copy them in a disk image or to export them via SAMBA or NFS. The default access is *read-only*.

Floppies can be emulated with the `:floppy:` option:

```
qemu linux.img -fda fat:floppy:/my_directory
```

A read/write support is available for testing (beta stage) with the `:rw:` option:

```
qemu linux.img -fda fat:floppy:rw:/my_directory
```

What you should *never* do:

- use non-ASCII filenames ;
- use "-snapshot" together with ":rw:" ;
- expect it to work when loadvm'ing ;
- write to the FAT directory on the host system while accessing it with the guest system.

3.7 Network emulation

QEMU can simulate several network cards (PCI or ISA cards on the PC target) and can connect them to an arbitrary number of Virtual Local Area Networks (VLANs). Host TAP devices can be connected to any QEMU VLAN. VLAN can be connected between separate instances of QEMU to simulate large networks. For simpler usage, a non privileged user mode network stack can replace the TAP device to have a basic network connection.

3.7.1 VLANs

QEMU simulates several VLANs. A VLAN can be symbolised as a virtual connection between several network devices. These devices can be for example QEMU virtual Ethernet cards or virtual Host ethernet devices (TAP devices).

3.7.2 Using TAP network interfaces

This is the standard way to connect QEMU to a real network. QEMU adds a virtual network device on your host (called `tapN`), and you can then configure it as if it was a real ethernet card.

3.7.2.1 Linux host

As an example, you can download the '`linux-test-xxx.tar.gz`' archive and copy the script '`qemu-ifup`' in '`/etc`' and configure properly `sudo` so that the command `ifconfig` contained in '`qemu-ifup`' can be executed as root. You must verify that your host kernel supports the TAP network interfaces: the device '`/dev/net/tun`' must be present.

See Section 3.3 [sec_invocation], page 3 to have examples of command lines using the TAP network interfaces.

3.7.2.2 Windows host

There is a virtual ethernet driver for Windows 2000/XP systems, called TAP-Win32. But it is not included in standard QEMU for Windows, so you will need to get it separately. It is part of OpenVPN package, so download OpenVPN from : `http://openvpn.net/`.

3.7.3 Using the user mode network stack

By using the option '`-net user`' (default configuration if no '`-net`' option is specified), QEMU uses a completely user mode network stack (you don't need root privilege to use the virtual network). The virtual network configuration is the following:

```
QEMU VLAN      <------>  Firewall/DHCP server <-----> Internet
                   |          (10.0.2.2)
                   |
                   ---->  DNS server (10.0.2.3)
                   |
                   ---->  SMB server (10.0.2.4)
```

The QEMU VM behaves as if it was behind a firewall which blocks all incoming connections. You can use a DHCP client to automatically configure the network in the QEMU VM. The DHCP server assign addresses to the hosts starting from 10.0.2.15.

In order to check that the user mode network is working, you can ping the address 10.0.2.2 and verify that you got an address in the range 10.0.2.x from the QEMU virtual DHCP server.

Note that `ping` is not supported reliably to the internet as it would require root privileges. It means you can only ping the local router (10.0.2.2).

When using the built-in TFTP server, the router is also the TFTP server.

When using the '`-redir`' option, TCP or UDP connections can be redirected from the host to the guest. It allows for example to redirect X11, telnet or SSH connections.

3.7.4 Connecting VLANs between QEMU instances

Using the '`-net socket`' option, it is possible to make VLANs that span several QEMU instances. See Section 3.3 [sec_invocation], page 3 to have a basic example.

3.8 Direct Linux Boot

This section explains how to launch a Linux kernel inside QEMU without having to make a full bootable image. It is very useful for fast Linux kernel testing.

The syntax is:

```
qemu -kernel arch/i386/boot/bzImage -hda root-2.4.20.img -append "root=/dev/hda"
```

Use '`-kernel`' to provide the Linux kernel image and '`-append`' to give the kernel command line arguments. The '`-initrd`' option can be used to provide an INITRD image.

When using the direct Linux boot, a disk image for the first hard disk '`hda`' is required because its boot sector is used to launch the Linux kernel.

If you do not need graphical output, you can disable it and redirect the virtual serial port and the QEMU monitor to the console with the '`-nographic`' option. The typical command line is:

```
qemu -kernel arch/i386/boot/bzImage -hda root-2.4.20.img \
     -append "root=/dev/hda console=ttyS0" -nographic
```

Use ⟨Ctrl-a c⟩ to switch between the serial console and the monitor (see Section 3.4 [pc-sys_keys], page 14).

3.9 USB emulation

QEMU emulates a PCI UHCI USB controller. You can virtually plug virtual USB devices or real host USB devices (experimental, works only on Linux hosts). Qemu will automatically create and connect virtual USB hubs as necessary to connect multiple USB devices.

3.9.1 Connecting USB devices

USB devices can be connected with the '`-usbdevice`' commandline option or the `usb_add` monitor command. Available devices are:

`mouse` Virtual Mouse. This will override the PS/2 mouse emulation when activated.

`tablet` Pointer device that uses absolute coordinates (like a touchscreen). This means qemu is able to report the mouse position without having to grab the mouse. Also overrides the PS/2 mouse emulation when activated.

`disk:`*file*
: Mass storage device based on *file* (see Section 3.6 [disk_images], page 18)

`host:`*bus.addr*
: Pass through the host device identified by *bus.addr* (Linux only)

`host:`*vendor_id:product_id*
: Pass through the host device identified by *vendor_id:product_id* (Linux only)

`wacom-tablet`
: Virtual Wacom PenPartner tablet. This device is similar to the `tablet` above but it can be used with the tslib library because in addition to touch coordinates it reports touch pressure.

`keyboard` Standard USB keyboard. Will override the PS/2 keyboard (if present).

3.9.2 Using host USB devices on a Linux host

WARNING: this is an experimental feature. QEMU will slow down when using it. USB devices requiring real time streaming (i.e. USB Video Cameras) are not supported yet.

1. If you use an early Linux 2.4 kernel, verify that no Linux driver is actually using the USB device. A simple way to do that is simply to disable the corresponding kernel module by renaming it from '`mydriver.o`' to '`mydriver.o.disabled`'.

2. Verify that '`/proc/bus/usb`' is working (most Linux distributions should enable it by default). You should see something like that:

   ```
   ls /proc/bus/usb
   001  devices  drivers
   ```

3. Since only root can access to the USB devices directly, you can either launch QEMU as root or change the permissions of the USB devices you want to use. For testing, the following suffices:

   ```
   chown -R myuid /proc/bus/usb
   ```

4. Launch QEMU and do in the monitor:

   ```
   info usbhost
     Device 1.2, speed 480 Mb/s
       Class 00: USB device 1234:5678, USB DISK
   ```

   You should see the list of the devices you can use (Never try to use hubs, it won't work).

5. Add the device in QEMU by using:

   ```
   usb_add host:1234:5678
   ```

   Normally the guest OS should report that a new USB device is plugged. You can use the option '`-usbdevice`' to do the same.

6. Now you can try to use the host USB device in QEMU.

When relaunching QEMU, you may have to unplug and plug again the USB device to make it work again (this is a bug).

3.10 VNC security

The VNC server capability provides access to the graphical console of the guest VM across the network. This has a number of security considerations depending on the deployment scenarios.

3.10.1 Without passwords

The simplest VNC server setup does not include any form of authentication. For this setup it is recommended to restrict it to listen on a UNIX domain socket only. For example

```
qemu [...OPTIONS...] -vnc unix:/home/joebloggs/.qemu-myvm-vnc
```

This ensures that only users on local box with read/write access to that path can access the VNC server. To securely access the VNC server from a remote machine, a combination of netcat+ssh can be used to provide a secure tunnel.

3.10.2 With passwords

The VNC protocol has limited support for password based authentication. Since the protocol limits passwords to 8 characters it should not be considered to provide high security. The password can be fairly easily brute-forced by a client making repeat connections. For this reason, a VNC server using password authentication should be restricted to only listen on the loopback interface or UNIX domain sockets. Password ayuthentication is requested with the `password` option, and then once QEMU is running the password is set with the monitor. Until the monitor is used to set the password all clients will be rejected.

```
qemu [...OPTIONS...] -vnc :1,password -monitor stdio
(qemu) change vnc password
Password: ********
(qemu)
```

3.10.3 With x509 certificates

The QEMU VNC server also implements the VeNCrypt extension allowing use of TLS for encryption of the session, and x509 certificates for authentication. The use of x509 certificates is strongly recommended, because TLS on its own is susceptible to man-in-the-middle attacks. Basic x509 certificate support provides a secure session, but no authentication. This allows any client to connect, and provides an encrypted session.

```
qemu [...OPTIONS...] -vnc :1,tls,x509=/etc/pki/qemu -monitor stdio
```

In the above example `/etc/pki/qemu` should contain at least three files, `ca-cert.pem`, `server-cert.pem` and `server-key.pem`. Unprivileged users will want to use a private directory, for example `$HOME/.pki/qemu`. NB the `server-key.pem` file should be protected with file mode 0600 to only be readable by the user owning it.

3.10.4 With x509 certificates and client verification

Certificates can also provide a means to authenticate the client connecting. The server will request that the client provide a certificate, which it will then validate against the CA certificate. This is a good choice if deploying in an environment with a private internal certificate authority.

```
qemu [...OPTIONS...] -vnc :1,tls,x509verify=/etc/pki/qemu -monitor stdio
```

3.10.5 With x509 certificates, client verification and passwords

Finally, the previous method can be combined with VNC password authentication to provide two layers of authentication for clients.

```
qemu [...OPTIONS...] -vnc :1,password,tls,x509verify=/etc/pki/qemu -monitor stdio
(qemu) change vnc password
Password: ********
(qemu)
```

3.10.6 Generating certificates for VNC

The GNU TLS packages provides a command called `certtool` which can be used to generate certificates and keys in PEM format. At a minimum it is neccessary to setup a certificate authority, and issue certificates to each server. If using certificates for authentication, then each client will also need to be issued a certificate. The recommendation is for the server to keep its certificates in either `/etc/pki/qemu` or for unprivileged users in `$HOME/.pki/qemu`.

3.10.6.1 Setup the Certificate Authority

This step only needs to be performed once per organization / organizational unit. First the CA needs a private key. This key must be kept VERY secret and secure. If this key is compromised the entire trust chain of the certificates issued with it is lost.

```
# certtool --generate-privkey > ca-key.pem
```

A CA needs to have a public certificate. For simplicity it can be a self-signed certificate, or one issue by a commercial certificate issuing authority. To generate a self-signed certificate requires one core piece of information, the name of the organization.

```
# cat > ca.info <<EOF
cn = Name of your organization
ca
cert_signing_key
EOF
# certtool --generate-self-signed \
           --load-privkey ca-key.pem
           --template ca.info \
           --outfile ca-cert.pem
```

The `ca-cert.pem` file should be copied to all servers and clients wishing to utilize TLS support in the VNC server. The `ca-key.pem` must not be disclosed/copied at all.

3.10.6.2 Issuing server certificates

Each server (or host) needs to be issued with a key and certificate. When connecting the certificate is sent to the client which validates it against the CA certificate. The core piece of information for a server certificate is the hostname. This should be the fully qualified hostname that the client will connect with, since the client will typically also verify the hostname in the certificate. On the host holding the secure CA private key:

```
# cat > server.info <<EOF
organization = Name  of your organization
cn = server.foo.example.com
tls_www_server
encryption_key
signing_key
EOF
# certtool --generate-privkey > server-key.pem
# certtool --generate-certificate \
           --load-ca-certificate ca-cert.pem \
           --load-ca-privkey ca-key.pem \
           --load-privkey server server-key.pem \
           --template server.info \
           --outfile server-cert.pem
```

The `server-key.pem` and `server-cert.pem` files should now be securely copied to the server for which they were generated. The `server-key.pem` is security sensitive and should be kept protected with file mode 0600 to prevent disclosure.

3.10.6.3 Issuing client certificates

If the QEMU VNC server is to use the `x509verify` option to validate client certificates as its authentication mechanism, each client also needs to be issued a certificate. The client certificate contains enough metadata to uniquely identify the client, typically organization, state, city, building, etc. On the host holding the secure CA private key:

```
# cat > client.info <<EOF
```

```
country = GB
state = London
locality = London
organiazation = Name of your organization
cn = client.foo.example.com
tls_www_client
encryption_key
signing_key
EOF
# certtool --generate-privkey > client-key.pem
# certtool --generate-certificate \
           --load-ca-certificate ca-cert.pem \
           --load-ca-privkey ca-key.pem \
           --load-privkey client-key.pem \
           --template client.info \
           --outfile client-cert.pem
```

The `client-key.pem` and `client-cert.pem` files should now be securely copied to the client for which they were generated.

3.11 GDB usage

QEMU has a primitive support to work with gdb, so that you can do 'Ctrl-C' while the virtual machine is running and inspect its state.

In order to use gdb, launch qemu with the '-s' option. It will wait for a gdb connection:

```
> qemu -s -kernel arch/i386/boot/bzImage -hda root-2.4.20.img \
       -append "root=/dev/hda"
Connected to host network interface: tun0
Waiting gdb connection on port 1234
```

Then launch gdb on the 'vmlinux' executable:

```
> gdb vmlinux
```

In gdb, connect to QEMU:

```
(gdb) target remote localhost:1234
```

Then you can use gdb normally. For example, type 'c' to launch the kernel:

```
(gdb) c
```

Here are some useful tips in order to use gdb on system code:

1. Use `info reg` to display all the CPU registers.
2. Use `x/10i $eip` to display the code at the PC position.
3. Use `set architecture i8086` to dump 16 bit code. Then use `x/10i $cs*16+$eip` to dump the code at the PC position.

3.12 Target OS specific information

3.12.1 Linux

To have access to SVGA graphic modes under X11, use the `vesa` or the `cirrus` X11 driver. For optimal performances, use 16 bit color depth in the guest and the host OS.

When using a 2.6 guest Linux kernel, you should add the option `clock=pit` on the kernel command line because the 2.6 Linux kernels make very strict real time clock checks by default that QEMU cannot simulate exactly.

When using a 2.6 guest Linux kernel, verify that the 4G/4G patch is not activated because QEMU is slower with this patch. The QEMU Accelerator Module is also much slower in this case. Earlier Fedora Core 3 Linux kernel (< 2.6.9-1.724_FC3) were known to incorporate this patch by default. Newer kernels don't have it.

3.12.2 Windows

If you have a slow host, using Windows 95 is better as it gives the best speed. Windows 2000 is also a good choice.

3.12.2.1 SVGA graphic modes support

QEMU emulates a Cirrus Logic GD5446 Video card. All Windows versions starting from Windows 95 should recognize and use this graphic card. For optimal performances, use 16 bit color depth in the guest and the host OS.

If you are using Windows XP as guest OS and if you want to use high resolution modes which the Cirrus Logic BIOS does not support (i.e. >= 1280x1024x16), then you should use the VESA VBE virtual graphic card (option '`-std-vga`').

3.12.2.2 CPU usage reduction

Windows 9x does not correctly use the CPU HLT instruction. The result is that it takes host CPU cycles even when idle. You can install the utility from `http://www.user.cityline.ru/~maxamn/amnhltm.zip` to solve this problem. Note that no such tool is needed for NT, 2000 or XP.

3.12.2.3 Windows 2000 disk full problem

Windows 2000 has a bug which gives a disk full problem during its installation. When installing it, use the '`-win2k-hack`' QEMU option to enable a specific workaround. After Windows 2000 is installed, you no longer need this option (this option slows down the IDE transfers).

3.12.2.4 Windows 2000 shutdown

Windows 2000 cannot automatically shutdown in QEMU although Windows 98 can. It comes from the fact that Windows 2000 does not automatically use the APM driver provided by the BIOS.

In order to correct that, do the following (thanks to Struan Bartlett): go to the Control Panel => Add/Remove Hardware & Next => Add/Troubleshoot a device => Add a new device & Next => No, select the hardware from a list & Next => NT Apm/Legacy Support & Next => Next (again) a few times. Now the driver is installed and Windows 2000 now correctly instructs QEMU to shutdown at the appropriate moment.

3.12.2.5 Share a directory between Unix and Windows

See Section 3.3 [sec_invocation], page 3 about the help of the option '`-smb`'.

3.12.2.6 Windows XP security problem

Some releases of Windows XP install correctly but give a security error when booting:

```
A problem is preventing Windows from accurately checking the
license for this computer. Error code: 0x800703e6.
```

The workaround is to install a service pack for XP after a boot in safe mode. Then reboot, and the problem should go away. Since there is no network while in safe mode, its recommended to download the full installation of SP1 or SP2 and transfer that via an ISO or using the vvfat block device ("-hdb fat:directory_which_holds_the_SP").

3.12.3 MS-DOS and FreeDOS

3.12.3.1 CPU usage reduction

DOS does not correctly use the CPU HLT instruction. The result is that it takes host CPU cycles even when idle. You can install the utility from `http://www.vmware.com/software/dosidle210.zip` to solve this problem.

4 QEMU System emulator for non PC targets

QEMU is a generic emulator and it emulates many non PC machines. Most of the options are similar to the PC emulator. The differences are mentioned in the following sections.

4.1 QEMU PowerPC System emulator

Use the executable '`qemu-system-ppc`' to simulate a complete PREP or PowerMac PowerPC system.

QEMU emulates the following PowerMac peripherals:

- UniNorth PCI Bridge
- PCI VGA compatible card with VESA Bochs Extensions
- 2 PMAC IDE interfaces with hard disk and CD-ROM support
- NE2000 PCI adapters
- Non Volatile RAM
- VIA-CUDA with ADB keyboard and mouse.

QEMU emulates the following PREP peripherals:

- PCI Bridge
- PCI VGA compatible card with VESA Bochs Extensions
- 2 IDE interfaces with hard disk and CD-ROM support
- Floppy disk
- NE2000 network adapters
- Serial port
- PREP Non Volatile RAM
- PC compatible keyboard and mouse.

QEMU uses the Open Hack'Ware Open Firmware Compatible BIOS available at `http://perso.magic.fr/l_indien/OpenHackWare/index.htm`.

The following options are specific to the PowerPC emulation:

'`-g WxH[xDEPTH]`'
: Set the initial VGA graphic mode. The default is 800x600x15.

More information is available at `http://perso.magic.fr/l_indien/qemu-ppc/`.

4.2 Sparc32 System emulator

Use the executable '`qemu-system-sparc`' to simulate a SPARCstation 5, SPARCstation 10, or SPARCserver 600MP (sun4m architecture). The emulation is somewhat complete. SMP up to 16 CPUs is supported, but Linux limits the number of usable CPUs to 4.

QEMU emulates the following sun4m peripherals:

- IOMMU
- TCX Frame buffer
- Lance (Am7990) Ethernet

- Non Volatile RAM M48T08
- Slave I/O: timers, interrupt controllers, Zilog serial ports, keyboard and power/reset logic
- ESP SCSI controller with hard disk and CD-ROM support
- Floppy drive (not on SS-600MP)
- CS4231 sound device (only on SS-5, not working yet)

The number of peripherals is fixed in the architecture. Maximum memory size depends on the machine type, for SS-5 it is 256MB and for SS-10 and SS-600MP 2047MB.

Since version 0.8.2, QEMU uses OpenBIOS `http://www.openbios.org/`. OpenBIOS is a free (GPL v2) portable firmware implementation. The goal is to implement a 100% IEEE 1275-1994 (referred to as Open Firmware) compliant firmware.

A sample Linux 2.6 series kernel and ram disk image are available on the QEMU web site. Please note that currently NetBSD, OpenBSD or Solaris kernels don't work.

The following options are specific to the Sparc32 emulation:

'`-g WxHx[xDEPTH]`'
: Set the initial TCX graphic mode. The default is 1024x768x8, currently the only other possible mode is 1024x768x24.

'`-prom-env string`'
: Set OpenBIOS variables in NVRAM, for example:

```
qemu-system-sparc -prom-env 'auto-boot?=false' \
 -prom-env 'boot-device=sd(0,2,0):d' -prom-env 'boot-args=linux single'
```

'`-M [SS-5|SS-10|SS-600MP]`'
: Set the emulated machine type. Default is SS-5.

4.3 Sparc64 System emulator

Use the executable '`qemu-system-sparc64`' to simulate a Sun4u machine. The emulator is not usable for anything yet.

QEMU emulates the following sun4u peripherals:

- UltraSparc IIi APB PCI Bridge
- PCI VGA compatible card with VESA Bochs Extensions
- Non Volatile RAM M48T59
- PC-compatible serial ports

4.4 MIPS System emulator

Four executables cover simulation of 32 and 64-bit MIPS systems in both endian options, '`qemu-system-mips`', '`qemu-system-mipsel`' '`qemu-system-mips64`' and '`qemu-system-mips64el`'. Four different machine types are emulated:

- A generic ISA PC-like machine `"mips"`
- The MIPS Malta prototype board `"malta"`
- An ACER Pica `"pica61"`. This machine needs the 64-bit emulator.

– MIPS emulator pseudo board "mipssim"

The generic emulation is supported by Debian 'Etch' and is able to install Debian into a virtual disk image. The following devices are emulated:

– A range of MIPS CPUs, default is the 24Kf
– PC style serial port
– PC style IDE disk
– NE2000 network card

The Malta emulation supports the following devices:

– Core board with MIPS 24Kf CPU and Galileo system controller
– PIIX4 PCI/USB/SMbus controller
– The Multi-I/O chip's serial device
– PCnet32 PCI network card
– Malta FPGA serial device
– Cirrus VGA graphics card

The ACER Pica emulation supports:

– MIPS R4000 CPU
– PC-style IRQ and DMA controllers
– PC Keyboard
– IDE controller

The mipssim pseudo board emulation provides an environment similiar to what the proprietary MIPS emulator uses for running Linux. It supports:

– A range of MIPS CPUs, default is the 24Kf
– PC style serial port
– MIPSnet network emulation

4.5 ARM System emulator

Use the executable '`qemu-system-arm`' to simulate a ARM machine. The ARM Integrator/CP board is emulated with the following devices:

– ARM926E, ARM1026E, ARM946E, ARM1136 or Cortex-A8 CPU
– Two PL011 UARTs
– SMC 91c111 Ethernet adapter
– PL110 LCD controller
– PL050 KMI with PS/2 keyboard and mouse.
– PL181 MultiMedia Card Interface with SD card.

The ARM Versatile baseboard is emulated with the following devices:

– ARM926E, ARM1136 or Cortex-A8 CPU
– PL190 Vectored Interrupt Controller
– Four PL011 UARTs

- SMC 91c111 Ethernet adapter
- PL110 LCD controller
- PL050 KMI with PS/2 keyboard and mouse.
- PCI host bridge. Note the emulated PCI bridge only provides access to PCI memory space. It does not provide access to PCI IO space. This means some devices (eg. ne2k_pci NIC) are not usable, and others (eg. rtl8139 NIC) are only usable when the guest drivers use the memory mapped control registers.
- PCI OHCI USB controller.
- LSI53C895A PCI SCSI Host Bus Adapter with hard disk and CD-ROM devices.
- PL181 MultiMedia Card Interface with SD card.

The ARM RealView Emulation baseboard is emulated with the following devices:

- ARM926E, ARM1136, ARM11MPCORE(x4) or Cortex-A8 CPU
- ARM AMBA Generic/Distributed Interrupt Controller
- Four PL011 UARTs
- SMC 91c111 Ethernet adapter
- PL110 LCD controller
- PL050 KMI with PS/2 keyboard and mouse
- PCI host bridge
- PCI OHCI USB controller
- LSI53C895A PCI SCSI Host Bus Adapter with hard disk and CD-ROM devices
- PL181 MultiMedia Card Interface with SD card.

The XScale-based clamshell PDA models ("Spitz", "Akita", "Borzoi" and "Terrier") emulation includes the following peripherals:

- Intel PXA270 System-on-chip (ARM V5TE core)
- NAND Flash memory
- IBM/Hitachi DSCM microdrive in a PXA PCMCIA slot - not in "Akita"
- On-chip OHCI USB controller
- On-chip LCD controller
- On-chip Real Time Clock
- TI ADS7846 touchscreen controller on SSP bus
- Maxim MAX1111 analog-digital converter on I^2C bus
- GPIO-connected keyboard controller and LEDs
- Secure Digital card connected to PXA MMC/SD host
- Three on-chip UARTs
- WM8750 audio CODEC on I^2C and I^2S busses

The Palm Tungsten|E PDA (codename "Cheetah") emulation includes the following elements:

- Texas Instruments OMAP310 System-on-chip (ARM 925T core)
- ROM and RAM memories (ROM firmware image can be loaded with -option-rom)

- On-chip LCD controller
- On-chip Real Time Clock
- TI TSC2102i touchscreen controller / analog-digital converter / Audio CODEC, connected through MicroWire and I²S busses
- GPIO-connected matrix keypad
- Secure Digital card connected to OMAP MMC/SD host
- Three on-chip UARTs

The Luminary Micro Stellaris LM3S811EVB emulation includes the following devices:

- Cortex-M3 CPU core.
- 64k Flash and 8k SRAM.
- Timers, UARTs, ADC and I²C interface.
- OSRAM Pictiva 96x16 OLED with SSD0303 controller on I²C bus.

The Luminary Micro Stellaris LM3S6965EVB emulation includes the following devices:

- Cortex-M3 CPU core.
- 256k Flash and 64k SRAM.
- Timers, UARTs, ADC, I²C and SSI interfaces.
- OSRAM Pictiva 128x64 OLED with SSD0323 controller connected via SSI.

A Linux 2.6 test image is available on the QEMU web site. More information is available in the QEMU mailing-list archive.

4.6 ColdFire System emulator

Use the executable '`qemu-system-m68k`' to simulate a ColdFire machine. The emulator is able to boot a uClinux kernel.

The M5208EVB emulation includes the following devices:

- MCF5208 ColdFire V2 Microprocessor (ISA A+ with EMAC).
- Three Two on-chip UARTs.
- Fast Ethernet Controller (FEC)

The AN5206 emulation includes the following devices:

- MCF5206 ColdFire V2 Microprocessor.
- Two on-chip UARTs.

5 QEMU User space emulator

5.1 Supported Operating Systems

The following OS are supported in user space emulation:

- Linux (referred as qemu-linux-user)
- Mac OS X/Darwin (referred as qemu-darwin-user)

5.2 Linux User space emulator

5.2.1 Quick Start

In order to launch a Linux process, QEMU needs the process executable itself and all the target (x86) dynamic libraries used by it.

- On x86, you can just try to launch any process by using the native libraries:

  `qemu-i386 -L / /bin/ls`

  `-L /` tells that the x86 dynamic linker must be searched with a '`/`' prefix.

- Since QEMU is also a linux process, you can launch qemu with qemu (NOTE: you can only do that if you compiled QEMU from the sources):

  `qemu-i386 -L / qemu-i386 -L / /bin/ls`

- On non x86 CPUs, you need first to download at least an x86 glibc ('`qemu-runtime-i386-XXX-.tar.gz`' on the QEMU web page). Ensure that `LD_LIBRARY_PATH` is not set:

  `unset LD_LIBRARY_PATH`

  Then you can launch the precompiled '`ls`' x86 executable:

  `qemu-i386 tests/i386/ls`

  You can look at '`qemu-binfmt-conf.sh`' so that QEMU is automatically launched by the Linux kernel when you try to launch x86 executables. It requires the `binfmt_misc` module in the Linux kernel.

- The x86 version of QEMU is also included. You can try weird things such as:

```
qemu-i386 /usr/local/qemu-i386/bin/qemu-i386 \
          /usr/local/qemu-i386/bin/ls-i386
```

5.2.2 Wine launch

- Ensure that you have a working QEMU with the x86 glibc distribution (see previous section). In order to verify it, you must be able to do:

  `qemu-i386 /usr/local/qemu-i386/bin/ls-i386`

- Download the binary x86 Wine install ('`qemu-XXX-i386-wine.tar.gz`' on the QEMU web page).

- Configure Wine on your account. Look at the provided script '`/usr/local/qemu-i386/bin/wine-conf.sh`'. Your previous `${HOME}/.wine` directory is saved to `${HOME}/.wine.org`.

- Then you can try the example '`putty.exe`':

```
qemu-i386 /usr/local/qemu-i386/wine/bin/wine \
          /usr/local/qemu-i386/wine/c/Program\ Files/putty.exe
```

5.2.3 Command line options

`usage: qemu-i386 [-h] [-d] [-L path] [-s size] program [arguments...]`

'`-h`' Print the help

'`-L path`' Set the x86 elf interpreter prefix (default=/usr/local/qemu-i386)

'`-s size`' Set the x86 stack size in bytes (default=524288)

Debug options:

'`-d`' Activate log (logfile=/tmp/qemu.log)

'`-p pagesize`'
Act as if the host page size was 'pagesize' bytes

Environment variables:

`QEMU_STRACE`
Print system calls and arguments similar to the 'strace' program (NOTE: the actual 'strace' program will not work because the user space emulator hasn't implemented ptrace). At the moment this is incomplete. All system calls that don't have a specific argument format are printed with information for six arguments. Many flag-style arguments don't have decoders and will show up as numbers.

5.2.4 Other binaries

`qemu-arm` is also capable of running ARM "Angel" semihosted ELF binaries (as implemented by the arm-elf and arm-eabi Newlib/GDB configurations), and arm-uclinux bFLT format binaries.

`qemu-m68k` is capable of running semihosted binaries using the BDM (m5xxx-ram-hosted.ld) or m68k-sim (sim.ld) syscall interfaces, and coldfire uClinux bFLT format binaries.

The binary format is detected automatically.

`qemu-sparc32plus` can execute Sparc32 and SPARC32PLUS binaries (Sparc64 CPU, 32 bit ABI).

`qemu-sparc64` can execute some Sparc64 (Sparc64 CPU, 64 bit ABI) and SPARC32PLUS binaries (Sparc64 CPU, 32 bit ABI).

5.3 Mac OS X/Darwin User space emulator

5.3.1 Mac OS X/Darwin Status

- target x86 on x86: Most apps (Cocoa and Carbon too) works. [1]
- target PowerPC on x86: Not working as the ppc commpage can't be mapped (yet!)
- target PowerPC on PowerPC: Most apps (Cocoa and Carbon too) works. [1]
- target x86 on PowerPC: most utilities work. Cocoa and Carbon apps are not yet supported.

[1] If you're host commpage can be executed by qemu.

5.3.2 Quick Start

In order to launch a Mac OS X/Darwin process, QEMU needs the process executable itself and all the target dynamic libraries used by it. If you don't have the FAT libraries (you're running Mac OS X/ppc) you'll need to obtain it from a Mac OS X CD or compile them by hand.

- On x86, you can just try to launch any process by using the native libraries:

  `qemu-i386 /bin/ls`

  or to run the ppc version of the executable:

  `qemu-ppc /bin/ls`

- On ppc, you'll have to tell qemu where your x86 libraries (and dynamic linker) are installed:

  `qemu-i386 -L /opt/x86_root/ /bin/ls`

  `-L /opt/x86_root/` tells that the dynamic linker (dyld) path is in '`/opt/x86_root/usr/bin/dyld`'.

5.3.3 Command line options

```
usage: qemu-i386 [-h] [-d] [-L path] [-s size] program [arguments...]
```

'`-h`' Print the help

'`-L path`' Set the library root path (default=/)

'`-s size`' Set the stack size in bytes (default=524288)

Debug options:

'`-d`' Activate log (logfile=/tmp/qemu.log)

'`-p pagesize`'
Act as if the host page size was 'pagesize' bytes

6 Compilation from the sources

6.1 Linux/Unix

6.1.1 Compilation

First you must decompress the sources:

```
cd /tmp
tar zxvf qemu-x.y.z.tar.gz
cd qemu-x.y.z
```

Then you configure QEMU and build it (usually no options are needed):

```
./configure
make
```

Then type as root user:

```
make install
```

to install QEMU in '`/usr/local`'.

6.1.2 GCC version

In order to compile QEMU successfully, it is very important that you have the right tools. The most important one is gcc. On most hosts and in particular on x86 ones, *gcc 4.x is not supported*. If your Linux distribution includes a gcc 4.x compiler, you can usually install an older version (it is invoked by `gcc32` or `gcc34`). The QEMU configure script automatically probes for these older versions so that usually you don't have to do anything.

6.2 Windows

- Install the current versions of MSYS and MinGW from `http://www.mingw.org/`. You can find detailed installation instructions in the download section and the FAQ.
- Download the MinGW development library of SDL 1.2.x ('`SDL-devel-1.2.x-mingw32.tar.gz`') from `http://www.libsdl.org`. Unpack it in a temporary place, and unpack the archive '`i386-mingw32msvc.tar.gz`' in the MinGW tool directory. Edit the '`sdl-config`' script so that it gives the correct SDL directory when invoked.
- Extract the current version of QEMU.
- Start the MSYS shell (file '`msys.bat`').
- Change to the QEMU directory. Launch '`./configure`' and '`make`'. If you have problems using SDL, verify that '`sdl-config`' can be launched from the MSYS command line.
- You can install QEMU in '`Program Files/Qemu`' by typing '`make install`'. Don't forget to copy '`SDL.dll`' in '`Program Files/Qemu`'.

6.3 Cross compilation for Windows with Linux

- Install the MinGW cross compilation tools available at `http://www.mingw.org/`.

- Install the Win32 version of SDL (`http://www.libsdl.org`) by unpacking '`i386-mingw32msvc.tar.gz`'. Set up the PATH environment variable so that '`i386-mingw32msvc-sdl-config`' can be launched by the QEMU configuration script.
- Configure QEMU for Windows cross compilation:

  `./configure --enable-mingw32`

  If necessary, you can change the cross-prefix according to the prefix chosen for the MinGW tools with –cross-prefix. You can also use –prefix to set the Win32 install path.
- You can install QEMU in the installation directory by typing '`make install`'. Don't forget to copy '`SDL.dll`' in the installation directory.

Note: Currently, Wine does not seem able to launch QEMU for Win32.

6.4 Mac OS X

The Mac OS X patches are not fully merged in QEMU, so you should look at the QEMU mailing list archive to have all the necessary information.

7 Index

(Index is nonexistent)