KVM

From Gentoo Linux Wiki

Note: KVM has been merged with QEMU

KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 or x86_64 (AKA amd64) hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure and a processor specific module, kvm-intel.ko or kvm-amd.ko which together provide for all that is needed to get virtualized CPU support.

KVM also requires a modified QEMU or QEMU 0.11 to provide for the remaining virtualized hardware (a virtual machine) including network, disk, and video adapters as well as block devices like hard drives, cdrom or floppies.

Be aware that not all Intel or AMD CPUs have virtualization extensions and some might need to have also a BIOS setting enabled in order to be able to use it. Refer to the KVM FAQ (http://www.linux-kvm.org/page/FAQ) for more information.

This howto will explain how to install kvm on your Host machine, start the installation process of your Guest OS, configure the USB and network parts.
We will show a practical example of two Guest OS with network enabled and able to talk to each other and to the Internet.

Virtualization TOC

- **Virtualization**
  - Paravirtualization
    - **Xen**
      - DomU creation
      - Running Xen
      - Troubleshooting
      - Links and resources
    - **KVM**
    - **QEmu**
    - **VirtualBox**
    - **VMware**
      - VMware Server
      - VMware Workstation
      - Gentoo guest install
    - **MS VirtualPC addons**
  - Container-based virtualization
    - **LXC**
    - **OpenVZ**
      - VLAN

Contents

Prerequisites

Hardware

Your CPU has to support full virtualization. Intel CPUs need to have the vmx capability, AMD CPUs require svm:

```
egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

Packages

For this howto, we will need the following packages:

- app-emulation/qemu-kvm (http://gentoo-portage.com/app-emulation/qemu-kvm)
- sys-apps/usbutils (http://gentoo-portage.com/sys-apps/usbutils) : needed for `lsusb` (see #USB_support)
- net-misc/bridge-utils (http://gentoo-portage.com/net-misc/bridge-utils) : needed for `brctl` (see #Networking)
- sys-apps/usermode-utilities (http://gentoo-portage.com/Search?search=sys-apps/usermode-utilities) : needed for `tunctl` (see #Networking)
- net-firewall/iptables (http://gentoo-portage.com/Search?search=net-firewall/iptables) :

needed for `masquerade` (see #Networking)

```
emerge -av qemu-kvm usbutils bridge-utils usermode-utilities iptables macchanger
```

Optional: app-emulation/virt-manager

TODO:Add information about KSM (samepage merging) which is included in kernel 2.6.32, see also the [1] (http://www.linux-kvm.com/content/using-ksm-kernel-samepage-merging-kvm) website

Kernel configuration

The virtual machine

To enable the KVM, you need the kvm virtual machine, and a driver. Most likely you will want to select the Intel or AMD specific driver depending on what CPU your host computer has, but if you want to run out-of-tree modules (USE flag **modules** dependent package **app-emulation/kvm-kmod**), you may get along without either.

Linux Kernel Configuration: Enable KVM

```
[*] Virtualization --->
    --- Virtualization
    <M> Kernel-based Virtual Machine (KVM) support
    < >   KVM for Intel processors support
    < >   KVM for AMD processors support
```

Note: In kernel version 2.6.24-r4 and before, `Virtualization` was located under `Device Drivers`.

Networking

If you want your virtual machine to access your network, you will need to support bridging, the TUN/TAP interfaces and VLAN.

Linux Kernel Configuration: Enable bridging, TUN and VLAN

```
Device Drivers --->
    [*] Network device support --->
            <M> Universal TUN/TAP device driver support

Networking support --->
    Networking options --->
        <*> 802.1d Ethernet Bridging
        <*> 802.1Q VLAN Support
```

Paravirtualization

If you want your virtual machine to have paravirtualization guest support, which can improve virtual machine performance in some scenarios, then you will need to support paravirtualization:

Linux Kernel Configuration: Enable Paravirtualization

```
Processor type and features --->
    [*] Paravirtualized guest support  --->
            [*]   KVM paravirtualized clock
            [*]   KVM Guest support
            -*-   Enable paravirtualization code
            [*]     Paravirtualization layer for spinlocks
```

Kernel SamePage Merging (KSM)

Kernel SamePage Merging combines identical memory pages from multiple processes into one copy on write memory region, this provides KVM with a memory overcommit feature that is important to hypervisors for more efficient use of memory and an overcommit feature for memory allocation. If you need to run multiple virtual machines and memory is a limiting factor, then KSM is your ideal solution; alternatively, even in situations where memory is not a constraint KSM can provide better memory management and improve scalability:

Linux Kernel Configuration: Enable KSM

```
Processor type and features --->
    [*] Enable KSM for page merging
```

You can now compile your kernel and install the modules:

```
make && make modules_install
```

Don't forget to copy your new kernel over, using the make command, it copies the kernel to /boot and renames any old kernel with the same name:

```
# make install
```

After booting your previously built kernel be sure to check

```
# cat /sys/kernel/mm/ksm/run
```

to return 1. Otherwise ksm won't be working.

```
# echo 1 > /sys/kernel/mm/ksm/run
```

To turn ksm on.

Installation

If a stable version is not available then you will need to unmask qemu-kvm as described below if you are using "stable" or move to "unstable" and help stabilizing it.

```
# echo "app-emulation/qemu-kvm" >> /etc/portage/package.keywords
```

Configuring your USE flags (version: qemu-kvm-0.12.2-r1)

The ebuild has several configurable options that can be used to adjust it to your environment better. For more information about how to set the USE flags refer to HOWTO Use Portage Correctly. By default qemu-kvm has convenient USE flags enabled by its ebuild.

Note: The list below is incomplete. To get the most up to date information, check what USE flags mean by installing gentoolkit and running **equery uses qemu-kvm**

alsa
: Adds support for media-libs/alsa-lib (Advanced Linux Sound Architecture). This used to be enabled by default for "<app-emulation/kvm-47" but was turned off because the same functionality is available by using alsa through SDL instead.

esd
: Adds support for media-sound/esound (Enlightened Sound Daemon)

gnutls
: Adds support for net-libs/gnutls (TLS 1.0 and SSL 3.0 support). Since >=app-emulation/kvm-45, the VNC server option available from the userspace application with the -vnc option is able to use TLS to encrypt the session established with the vnc client, this flag enables support for that functionality. If disabled vnc sessions are still possible but only using unencrypted channels.

pulseaudio
: Adds support for PulseAudio sound server.

sdl
: Adds support for Simple Direct Layer (media library). If enabled (recommended), will use SDL to emulate a console, so you can interact with your virtual machine through an X session. If you are only interested on using kvm to run "headless" virtual machines then be sure to start them with either an emulated serial console going into a file (or /dev/null) or a vnc server console and disable this flag.

vde
: Enable VDE-based networking. VDE is a package that provides software versions of physical networking devices (e.g. switches, cables, etc). In the same way that KVM allows you to virtualize a computer, VDE allows you to virtualize your networking gear.

Getting Started

Adding access to /dev/kvm to your user

The /dev/kvm device is owned by root but with read/write privileges for the kvm group, so in order to be able to access it you have to add your user to it:

```
gpasswd -a <your_user_name> kvm
```

Create a virtual disk image

To create a virtual disk image, use the qemu-img tool. The following example will create a 10G Copy On Write disk image:

```
qemu-img create -f qcow2 -o preallocation=metadata gentoo-i386.img 10G
```

This image uses the qemu specific qcow2 format. It supports encryption, compression, and snapshotting and grows dynamically. For better performance, the metadata will be preallocated, allowing easier growth, but increases the initial image size on the other hand.

Installing the guest OS to your image

Gentoo example

To install gentoo into the disk image using an livecd ISO:

```
qemu-kvm -hda gentoo-i386.img -cdrom livecd-i686-installer-2007.0.iso -boot d
```

You can look at the kvm man page to see that this command says:

- run a virtual machine with:
- an emulated hard disk drive using the file gentoo-i386.img
- an emulated CDROM drive containing the emulated CDROM image livecd-i686-installer-2007.0.iso
- and boot from device "d" (i.e., the emulated CDROM drive)

Windows example

As of 2.6.29 with 84/5 I am no longer able to start windows. I had to go down to 2.6.28.
--Letharion
With kvm-78 and kernel-2.6.24, nothing special was required to install Windows XP. The following command was used:

```
qemu-kvm -hda winxp.img -cdrom /dev/cdrom -boot d
```

I have successfully installed Windows Vista Ultimate with kvm-84 and 2.6.28 like this:

```
qemu-kvm Vista.img -cdrom /home/user/Windows.iso -m 512 -boot d
```

Without the -m 512 Vista complained about too little available RAM.

I have successfully installed Windows XP with kvm-83 and 2.6.27 like this:

```
qemu-kvm winxp.img -cdrom /home/user/Windows.iso -boot d
```

Once installation is complete, and you no longer need the disc, the command simply becomes

```
qemu-kvm winxp.img -boot c
```

This ignores the CD, and requests boot from harddrive, instead of cd-device.

Kubuntu example

The latest ubuntu/kubuntu variant "Jaunty Jackelope" may have trouble with monitor drivers other than the default cirrus. You will probably also need to override the default boot options on its initial boot screen with "noacpi" (hit f6 for that option) and possibly "safe graphics" (f4 options) before booting the install. It also appeared to hang on its initial splash screen with -soundhw ac97 to enable audio.

This command line sets 2gb of memory, a tuntap interface for network, a name for the sdl window (kubuntu) so that more than one kvm sdl session can run without interference and the alternative grab sequence "ctrl-alt-shift" instead of "ctrl-alt". This is kvm-0.84 on a 2.6.28-r5 amd64 kernel where the tuntap and bridge interface are already plumbed up from init.d and conf.d/net settings:

```
qemu-kvm -hda kubuntu904.img -cdrom kubuntu-9.04-dvd-amd64.iso -m 2048 \
-net nic,macaddr=00:1d:92:ab:3f:78 -net tap,ifname=tap0,script=no,downscript=no \
-alt-grab -name kubuntu -boot d
```

After the initial installation, experiments showed that the es1370 virtual driver would allow kubuntu to boot and use audio. However the cirrus driver still appeared to be the only monitor option that would work. Thus vm display resolutions will be limited to 1024x768 until software updates from Canonical permit better driver support.

Fedora 10 example

The latest RedHat Fedora variant (at least until May 2009) does not have trouble booting the installer with acpi enabled, enhanced vga or ac97 driver support. This guest lives on tap1 alongside the kubuntu guest above:

```
qemu-kvm -hda fedora10.img -cdrom Fedora-10-x86_64-DVD.iso -m 2048 \
-net nic,macaddr=00:1d:92:ab:3f:80 -net tap,ifname=tap1,script=no,downscript=no \
-alt-grab -vga std -soundhw ac97 -name fedora10 -boot d
```

After the install, Fedora will initially have troubles with the adjustment of the graphic display and system fonts when living in a vm. This is because it is using the new approach to Xorg which relies on the monitor's EDID information to calculate available display resolutions. You will need to use yum to install system-config-display (a sore point among the Fedora community that it was left out of the default install) in order to create the missing xorg.conf file and populate it with sane settings. You will also probably need to set a specific font resolution in kde's "system-settings -> appearances" to override the bad choices made for you.

USB support

KVM allows you to add USB support to your VM by "bridging" what the kernel of your Host sees to the Guest OS.
To do so, you need to start qemu-kvm with the -usb flag on. Moreover, you will need to tell the Guest OS which usb device is being used for it.
kvm (or more adequately qemu), can pass the USB VendorID and ProductID to the Guest OS in order for it to know that there is a USB attached. To know what VendorID and ProductID your USB device is, use the lsusb utility:

```
lsusb
```

```
...
Bus 001 Device 006: ID: 08ec:2039 M-Systems Flash Disk Pioneers
...
```

In this example, the VendorID is "08ec" and the ProductID "2039".

To launch qemu-kvm with USB support and this flash disk, you can use the following command:

```
qemu-kvm -usb -usbdevice host:08ec:2039 gentoo-i386.img
```

If you have more than one USB device you would like to support, just add another "-usbdevice host:<VendorID>:<ProductID>" to the command line.

Networking

There are a few ways to allow the Guest OS to talk with each other and the outside world.

Quick & Easy Networking

This is the most convenient method of networking. If you use Network Manager and just want to get your KVM virtual machine started and accessing the Internet, without doing all kinds of network setup, simply start kvm with the "-net user" switch:

```
qemu-kvm -net nic,macaddr=00:00:00:00:00:00 -net user gentoo-i386.img
```

Now your guest will have Internet access (assuming the host does), without setting up TUN,TAP,VDE, etc. However, you will not have local LAN access.

VDE Networking

An alternative way of networking all your virtual machines is to connect them to a virtual switch. This method is slightly easier than ethernet bridging.

Firstly you must make sure you compiled KVM with the **vde** USE flag. You can then start the virtual switch by running

```
vde_switch --daemon
```

Now when you launch your guest OS you can tell them to automatically connect to the virtual switch

```
qemu-kvm -net nic,macaddr=00:00:00:00:00:00 -net vde gentoo-i386.img
```

```
qemu-kvm -net nic,macaddr=00:00:00:00:00:01 -net vde openbsd.img
```

Your guests will now be able to communicate with each other.

If you want your guests to be able to communicate with the outside world (e.g. your LAN or the internet) then you need to connect your physical interface (e.g. eth0) to the virtual switch as well. This can be done by running

```
vde_pcapplug eth0
```

(Note that vde_pcapplug is only available when you are using net-misc/vde-2.2.3 (currently masked) or above with the **pcap** USE flag. One drawback with this vde_pcapplug approach is that the host and guest can't talk to each other directly due to limitations in linux kernel)

All incoming traffic on eth0 will now be copied to the virtual switch, and all traffic from the virtual switch will be sent out over eth0. This means your guests will appear as part of your physical LAN.

Ethernet bridging

Possible Network Layouts

The principle is the following: you need to create an interface that will be the bridge between all the TUN/TAP interfaces (one per Guest OS) and the ethernet one of your Host OS. From there, you will be able to forward the traffic (using iptables) to the Internet.

The configuration described here after can be seen as follows:

Direct bridging (so guests use an IP address on the same subnet as the host):

```
             HOST
       +---------------+
       |               |      KVM GUEST1
       |               |   +--------------+
       |  +------+     |   |              |
LAN ---+--- eth0 |  +--+---+---- nic0     |      KVM GUEST2
       |  | tap0----+  |   |192.168.1.13  |   +--------------+
       |  | tap1----+  |   +--------------+   |              |
       |  +------+  |  |                      |              |
       |     br0    +--+----------------------+---- nic0     |
       |192.168.1.12   |                      |192.168.1.14  |
       +---------------+                      +--------------+
```

When use NAT/Masquerading (to hide the guests behind the host):

```
             HOST
       +---------------+
       | 192.168.1.12  |      KVM GUEST1
LAN ---+---- eth0      |   +--------------+
       |      ^        |   |              |
       |      |        |   |              |
       |  +------+  +--+---+---- nic0     |      KVM GUEST2
       |  | tap0----+  |   |192.168.100.1 |   +--------------+
       |  | tap1----+  |   +--------------+   |              |
       |  +------+  |  |                      |              |
       |     br0    +--+----------------------+---- nic0     |
       |192.168.100.254|                      |192.168.100.2 |
       +---------------+                      +--------------+
```

Preparations

The most transparent option to allow your guests access to the internet is the "virtual hub". In this scheme, the bridge connects eth0 and your tuntap interfaces together, routing packets as if it were a real "old fashioned" hub (not a switch). The key to this approach is to make sure you have unique mac addresses on both the host's tuntap interface as well as the guest. The guest ip addresses are typically in the same subnet as the host, and they can ask for and receive a dhcp lease from the same dhcp server that the host might use if it used dhcp. That is because all arp traffic and other broadcasts are passed through the bridge between the eth0 interface and the guest taps. The guests can use the same default gateway as the host because of this transparent passage.

The following snippet from an /etc/conf.d/net file shows the setup of a bridge between eth0 and two tap devices for guests. Note that the dependency for eth0 is left out of the br0 config since it is always started earlier on this particular system.

Direct bridging:

File: /etc/conf.d/net

```
bridge_br0="eth0 tap0 tap1"
brctl_br0="setfd 0 sethello 0 stp off"
rc_need_br0="net.tap0 net.tap1"
#
#  host system is a static address at 192.168.1.12 with dns server at 34 and a router at 33
#
config_br0="192.168.1.12/24"
routes_br0="default via 192.168.1.33"
dns_domain_br0="example.com"
dns_servers_br0="192.168.1.34"
dns_search_br0="example.com"

#
# host system is dhcpcd configured
#
# config_br0="dhcp"

config_tap0="null"
tuntap_tap0="tap"
tunctl_tap0="-u joeuser"

config_tap1="null"
tuntap_tap1="tap"
tunctl_tap1="-u joeuser"

config_eth0="null"
```

Note: When bridging the guests and hosts together, there's usually no good reason to explicitly set the MAC address of the TAP interfaces on the host side. Some instructions advise setting the MAC on the host side of the TAP to the same address as the guest's emulated NIC, which usually causes reachability problems when the guest tries to communicate past the host.

Using NAT/Masquerading:

File: /etc/conf.d/net

```
bridge_br0="tap0 tap1"
brctl_br0="setfd 0 sethello 0 stp off"
rc_need_br0="net.tap0 net.tap1"
#
#  host system is a static address at 192.168.1.12 with dns server at 34 and a router at 33
#
config_eth0="192.168.1.12/24"
routes_eth0="default via 192.168.1.33"
dns_domain_eth0="example.com"
dns_servers_eth0="192.168.1.34"
dns_search_eth0="example.com"

config_br0="192.168.100.254/24"

config_tap0="null"
tuntap_tap0="tap"
tunctl_tap0="-u joeuser"
mac_tap0="00:00:00:00:00:00"

config_tap1="null"
tuntap_tap1="tap"
```

```
tunctl_tap1="-u joeuser"
mac_tap1="00:00:00:00:00:01"
```

Note: On Baselayout1 you need to set "RC_NEED" in upper case. Otherwise it wont work and on reboot net.br0 is not started.

This is essentially the conf.d/net setup for the Fedora10 and kubuntu guest examples above. Note that the bridge br0 uses a different subnet otherwise the routing table will be ambiguous. Note also that their nic definitions on the kvm command line use different mac addresses than what is set for their taps. If the same mac address had been used on both sides, the arp queries for address resolution would not work. This conf.d/net setup is also why the kvm command line says not to do anything about interface startup or takedown.

You need to create the appropriate net.br0, net.tap0 and net.tap1 initscripts. You can use ln to do this:

```
cd /etc/init.d
ln -s net.lo net.br0
ln -s net.lo net.tap0
ln -s net.lo net.tap1
```

You also need to add net.br0 to the default runlevel:

```
rc-update del net.eth0
rc-update add net.br0 default
rc-update add net.tap0 default
rc-update add net.tap1 default
```

Finally you need to make some changes to /etc/sysctl.conf and add a new init-script. The changes in /etc/sysctl.conf are to prevent the traffic from the guests to be sent to iptables to be filtered. If you want to filter the traffic from/to the guests, you can keep the file unchanged but you will have to add the correct rules to iptables. The addition needed in /etc/sysctl.conf is ...

```
#
# Setup bridge interface for KVM
#
net.bridge.bridge-nf-call-arptables=0
net.bridge.bridge-nf-call-iptables=0
net.bridge.bridge-nf-call-ip6tables=0
```

The new init-script is to allow br0 to forward the traffic from your guests. This can unfortunately not be added to the /etc/sysctl.conf because the br0 interface does not exist when those changes are applied. The best way I have found to fix this is to add a file called /etc/init.d/bridge_forward with the following content ...

```
#!/sbin/runscript

depend() {
        need net.br0
}

start() {
        ebegin "Turning on forwarding for bridge interface"
        /sbin/sysctl net.ipv4.conf.br0.forwarding=1 >/dev/null 2>&1
        eend $?
}
```

```
stop() {
        ebegin "Turning off forwarding for bridge interface"
        /sbin/sysctl net.ipv4.conf.br0.forwarding=0 >/dev/null 2>&1
        eend $?
}
```

Don't forget to add the script into default run level

```
rc-update add bridge_forward default
```

If you have net.br0 configured up with dhcpcd. You may need to delete dhcpcd service from default run level to prevent ambiguous route problem.

```
rc-update del dhcpcd default
```

The alternative to the virtual hub is to enable masquerade on your machine:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

The disadvantage of this approach is that the initial inbound connections to your guests will not work without dnat rules. This can become quite involved and generally will get you to look at using shorewall for the management.

Start your Guest OS

This section will show how to launch two Guest OS with kvm and the setting up. If you need the USB support, just append the USB options as shown earlier to the commands here after.

Launch KVM Guest1 with network enabled

Now that the requirements are included in the kernel or the modules compiled, we need to load the modules:

```
modprobe kvm_intel
modprobe tun
```

Let us create the bridge interface br0 and bring it up:

```
emerge -av bridge-utils
brctl addbr br0
ifconfig br0 192.168.100.254 netmask 255.255.255.0 up
```

The next step is to create the TAP interface:

```
emerge -av usermode-utilities
tunctl -b -u joeuser -t tap0
```

Now that we have the TAP interface and the bridge one, we need to link them:

```
brctl addif br0 tap0
```

And finally, we bring tap0 up with the "promisc" mode:

```
ifconfig tap0 up 0.0.0.0 promisc
```

Ok, you should now have br0 and tap0 appearing with ifconfig:

```
ifconfig
```

```
br0       Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:192.168.100.254  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:81 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6469 (6.3 KiB)  TX bytes:410 (410.0 B)

eth0      Link encap:Ethernet  HWaddr <your_hw_address>
          inet addr:192.168.1.5  Bcast:0.0.0.0  Mask:255.255.255.255
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:17

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:117562 errors:0 dropped:0 overruns:0 frame:0
          TX packets:117562 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8250309 (7.8 MiB)  TX bytes:8250309 (7.8 MiB)

tap0     Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:81 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:7669 (7.4 KiB)  TX bytes:410 (410.0 B)
```

Now, you can boot your VM with the following parameters:

```
qemu-kvm -net nic,macaddr=00:00:00:00:00:00 -net tap,ifname=tap0,script=no,downscript=no gentoo-i386.img
```

The two "-net" switches have a different meaning but are both needed:

- `nic,macaddr=00:00:00:00:00:00`: we specify the options related to the network interface of the Guest OS. Please note that the macaddr needs to be specified
- `tap,ifname=tap0,script=no`: we specify how kvm should work with the network on the Host side. The "script=no" means we will not use the given scripts in /etc/kvm/kvm.ifup as we already tuned everything before.

When using NAT, do not forget to configure KVM Guest1 (192.168.100.1) with the br0 interface (192.168.100.254) as the default gateway. (And you probably also wanna take the DNS-settings from /etc/resolv.conf --Letharion)

When using bridging, set KVM Guest1 (192.168.100.1) with the same default gateway as the host.

Launch KVM Guest2 with network enabled

What needs to be done as we already loaded the modules and the bridge interface, we just need to create the TAP/TUN interface and link it to br0:

```
tunctl -b -u joeuser -t tap1
brctl addif br0 tap1
ifconfig tap1 up 0.0.0.0 promisc
```

Now, you can boot your second VM with the following parameters:

```
qemu-kvm -net nic,macaddr=00:00:00:00:00:01 -net tap,ifname=tap1,script=no openbsd.img
```

Warning: You need to have another **macaddr** value if you want your Guest OS to be able to talk to each other!
Here, the first Guest OS has **00:00:00:00:00:00** and the second **00:00:00:00:00:01**.

When using NAT, do not forget to configure KVM Guest2 (192.168.100.2) with the br0 interface (192.168.100.254) as the default gateway.

When using bridging, do not forget to configure KVM Guest2 (192.168.100.2) with the same default gateway as the host.

Now, your two Guest OS should be able to ping each other.

Script to ease the configuration

If you want to ease the process of configuring your machine (load the modules, create the bridge interface, the TUN/TAP, ...), here is an init.d script:

File: /etc/init.d/kvm

```
#!/sbin/runscript
# Copyright 1999-2008 Gentoo Foundation
# Distributed under the terms of the GNU General Public License v2
# $Header: $

NUM_OF_DEVICES=5
USERID="<your_user>"

depend() {
        need net
}

start() {
        ebegin "Loading the kvm module"
        /sbin/modprobe kvm
        eend $? "Failed to load the kvm module"
        ebegin "Loading the kvm_intel module"
        /sbin/modprobe kvm_intel
        eend $? "Failed to load the kvm_intel module"
        ebegin "Loading the tun module"
        /sbin/modprobe tun
        eend $? "Failed to load the tun module"
        ebegin "Setting up the bridge device (br0)"
        /sbin/brctl addbr br0
        /sbin/ifconfig br0 192.168.100.254 netmask 255.255.255.0 up
        eend $? "Failed to create the bridge interface"
        for ((i=0; i < NUM_OF_DEVICES; i++)); do
                ebegin "Setting up the tap interface: tap$i"
                /usr/bin/tunctl -b -u $USERID -t tap$i >/dev/null
                eend $? "Failed to create the tap interface: tap$i"
                ebegin "Linking the bridge interface with tap$i"
                /sbin/brctl addif br0 tap$i
                eend $? "Failed to link the bridge interface to tap$i"
                ebegin "Bring tap$i interface up"
                /sbin/ifconfig tap$i up 0.0.0.0 promisc
                eend $? "Failed to bring tap$i up"
        done
        ebegin "Allowing Internet access"
        echo "1" > /proc/sys/net/ipv4/ip_forward
        eend $? "Failed to allow forwarding"
        iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
        eend $? "Failed to allow masquerade (eth0)"
        eend 0
}

stop() {
        for ((i=0; i < NUM_OF_DEVICES; i++)); do
                ebegin "Bring tap$i interface down"
                /sbin/ifconfig tap$i down
                eend $? "Failed to bring tap$i down"
                ebegin "Unlinking the bridge interface with tap$i"
                /sbin/brctl delif br0 tap$i
                eend $? "Failed to unlink the bridge interface to tap$i"
                ebegin "Removing the tap interface: tap$i"
                /usr/bin/tunctl -d tap$i >/dev/null
                eend $? "Failed to remove the tap interface: tap$i"
        done
        ebegin "Stopping the bridge device (br0)"
        /sbin/ifconfig br0 down
        /sbin/brctl delbr br0
        eend $? "Failed to stop the bridge interface"
        ebegin "Unloading the tun module"
        /sbin/modprobe -r tun
        eend $? "Failed to unload the tun module"
        ebegin "Unloading the kvm_intel module"
        /sbin/modprobe -r kvm_intel
        eend $? "Failed to unload the kvm_intel module"
        ebegin "Unloading the kvm module"
        /sbin/modprobe -r kvm
        eend $? "Failed to unload the kvm module"
        ebegin "Stopping Internet access"
        echo "0" > /proc/sys/net/ipv4/ip_forward
        eend $? "Failed to cancel forwarding"
        iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE
        eend $? "Failed to remove masquerade (eth0)"
        eend 0
}

restart() {
        stop
        start
}
```

Troubleshooting

Modprobe kvm-amd gives general protection fault

If CONFIG_KVM=n, kvm-kmod cannot be built, while if CONFIG_KVM=y, there is a possible conflict between in-kernel kvm and out-of-tree modules. Personally, I get general protection fault trying

```
modprobe kvm-amd
```

So, it should be CONFIG_KVM=m.

My x86 Gentoo Guest dies with a kernel panic

KVM doesn't emulate the MMR registers needed for performance metrics and therefore it will generate a General Protection Fault if they were used. A workaround was added to the Linux kernel 2.6.22.5 and therefore upgrading the kernel in your guest will fix this problem. If you can't upgrade your kernel try using the **nolapic** or **nmi_watchdog=0** boot parameters instead.

But I can't boot my guest anymore

If you are using the USE-Flag **qemu** then you can use qemu + kqemu instead to try to fix any problems with the guest that affected kvm, if that doesn't work try using **kvm -no-kvm** or **qemu -no-kqemu** as a last resort.

modprobe kvm modules error

Linux Kernel Configuration: KVM modules error

```
Processor type and features  --->
   Preemption Model (Voluntary Kernel Preemption (Desktop))
```

must not been set as No Forced Preemption (Server)

When you use **AMD CPU** and get such an output error in:

```
tail -f /var/log/messages
```

```
...
kvm: Unknown symbol intel_iommu_domain_alloc
kvm: Unknown symbol intel_iommu_detach_dev
kvm: Unknown symbol intel_iommu_page_mapping
kvm: Unknown symbol intel_iommu_context_mapping
kvm: Unknown symbol intel_iommu_iova_to_pfn
kvm: Unknown symbol intel_iommu_domain_exit
...
```

then turn off the experimental modules could help:

Linux Kernel Configuration: KVM experimental modules

```
PCI driver for virtio devices (EXPERIMENTAL)
Virtio balloon driver (EXPERIMENTAL)
```

What about when you're using an **Intel CPU**, these drivers are off, and you STILL get these same errors?? Or is kvm-81 just broken?

A bug on KVM's tracker indicated that CONFIG_DMAR may be to blame. (http://sourceforge.net /tracker/?func=detail&atid=893831&aid=2405145&group_id=180599)

Can I boot an Operating System (Windows, Linux, etc) residing on a partition instead of an image?

The key here is you must specify the hard disk (or lvm) (e.g. /dev/sda) that the OS resides on and not the specific root partition (e.g. /dev/sda3). If you boot multiple operating systems through grub normally, you should be greeted with a grub screen where you can select which OS you would like to boot.

```
qemu-kvm -vga std -m 512 -hda /dev/sda
```

External References

- Qemu (kvm) internal network setup (http://blog.cynapses.org/2007/07/12/qemu-kvm-internal-network-setup/)
- http://q.deltaquadrant.org/index.php/KVM
- Kernel Based Virtual Machine Homepage (http://www.linux-kvm.org/)
- Qemu Homepage (http://bellard.org/qemu/)
- HOWTO: Setting up QEMU on Ubuntu with TUN/TAP and NAT (http://ubuntuforums.org/showthread.php?t=179472)
- KVM / QEMU Easy Routed Networking (TAP interface without bridge) (http://tjworld.net/wiki/Linux/KvmQemuEasyRoutedNetwork)
- HOWTO: Using KVM with KSM on archlinux (https://wiki.archlinux.org/index.php/KVM#Enabling_KSM)

Retrieved from "http://en.gentoo-wiki.com/wiki/KVM"
Category: Virtualization