Arch Linux PCI Passthrough

Based on the following articles:

- https://wiki.archlinux.org/index.php/PCI_passthrough_via_OVMF
- https://github.com/vanities/GPU-Passthrough-Arch-Linux-to-Windows10
- https://old.reddit.com/r/archlinux/comments/5yi7w7

Prerequisites

- IOMMU-enabled CPU
- IOMMU-enabled Motherboard
- At least 2 GPUs (preferably two different models)
- At least 2 displays

```
Note:
IOMMU is a generic name for Intel VT-d and AMD-Vi
```

Setting up devices

The following assumes that IOMMU has been enabled in the BIOS beforehand. The procedure differs from motherboard to motherboard.

To enable PCI passthrough, we'll first have to figure out the IDs of the devices to pass through. These can be determined using `lspci -nnk`. There you should find an entry like this:

```
01:00.0 VGA compatible controller [0300]: NVIDIA Corporation GP104 [GeForce GTX 1080]
[10de:1b80] (rev a1)
	Subsystem: eVga.com. Corp. GP104 [GeForce GTX 1080] [3842:6183]
	Kernel driver in use: nvidia
	Kernel modules: nouveau, nvidia_drm, nvidia
01:00.1 Audio device [0403]: NVIDIA Corporation GP104 High Definition Audio Controller
[10de:10f0] (rev a1)
	Subsystem: eVga.com. Corp. GP104 High Definition Audio Controller [3842:6183]
	Kernel driver in use: snd_hda_intel
	Kernel modules: snd_hda_intel
```

The IDs we want are these:

`01:00.0 VGA compatible controller [0300]: NVIDIA Corporation GP104 [GeForce GTX 1080] [ **10de:1b80** ] (rev a1)` `01:00.1 Audio device [0403]: NVIDIA Corporation GP104 High Definition Audio Controller [ **10de:10f0** ] (rev a1)`

Now that we know these IDs, we can adjust our kernel parameters to enable IOMMU. For that we'll have to edit the `/etc/default/grub` file. There we'll have to add `intel_iommu=on` and `pci_stub.ids=<IDS>` the `GRUB_CMDLINE_LINUX_DEFAULT` variable. An example configuration would look like this:

```
# /etc/default/grub
...
GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on pci_stub.ids=10de:1b80,10de:10f0"
...
```

(Replace `intel` with `amd` on systems with an AMD CPU)

Now update grub:

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

After rebooting, your PCI device should now be isolated and ready to use with a VM. To make sure that the devices have been isolated successfully, check the output of `lspci -nnk` again. This time, the output should look like this:

```
02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GM204 [GeForce GTX 970]
[10de:13c2] (rev a1)
	Subsystem: eVga.com. Corp. GM204 [GeForce GTX 970] [3842:2974]
	Kernel driver in use: pci-stub
	Kernel modules: nouveau, nvidia_drm, nvidia
02:00.1 Audio device [0403]: NVIDIA Corporation GM204 High Definition Audio Controller
[10de:0fbb] (rev a1)
	Subsystem: eVga.com. Corp. GM204 High Definition Audio Controller [3842:2974]
	Kernel driver in use: pci-stub
	Kernel modules: snd_hda_intel
```

The key difference being the kernel driver, which is now `pci-stub` instead of nvidia.

Setting up Environment

Install the following software:

```
sudo pacman -S qemu libvirt ovmf virt-manager
```

then enable and start libvirt:

```
sudo systemctl enable libvirtd.service
sudo systemctl enable libvirtd.socket
sudo systemctl start libvirtd.service
sudo systemctl start libvirtd.socket
```

and enable the default libvirt network (Note: it may be necessary to install the `dnsmasq` and `ebtables` packages beforehand):

```
sudo virsh net-autostart default
sudo virsh net-start default
```

Add your user to the `libvirt` group:

```
sudo gpasswd -a $USER libvirt
```

Setting up Virtual Machine

Now we can start setting up the VM itself. Start `virt-manager` and create a new VM.

If QEMU/KVM does not show up, you might have to add it using File/Add new Connection. Make sure that qemu is installed.

Insert the Windows installation disk, define the amount of memory and CPU cores the VM is allowed to have, set up storage and specify the name.

Before hitting Finish, select `Customize configuration before install`.

In the next configuration window, set the firmware to UEFI in `Overview`.

For better IO performance, add the following hardware:

- Controller:Type=SCSI,ModelVirtIO SCSI (Change Disk bus from SATA to SCSI for boot disk)
- Storage:CDROM,SATA with the ISO from here.

Now we can start the VM and start instaling Windows. Sometimes the VM doesn't correctly boot from the ISO, in which case you will enter a prompt. Just enter `exit` and choose the boot device manually should this happen.

Proceed to install Windows as usual. If you decided to use the SCSI controller, you'll need to install the drivers using the downloaded ISO file.

Don't bother installing any drivers after the installation completes and turn off the VM instead.

Passing Hardware to the VM

First off, we'll remove the virtual display and input devices. Make sure to remove the following:

- Tablet
- Display Spice
- Channel spice
- Video QXL

Then add the PCI devices you want to attach to the VM using `Add Hardware`->`PCI Host Device`. Usually this would be the GPU display and audio devices. So a total of two devices.

Once this is done, we'll have to edit the configuration file to solve some issues as well as enable mouse and keyboard sharing with the host system. (You can also use dedicated devices for the keyboard and mouse, but that's pretty unwieldy)

First off, you need to figure out your mouse and keyboard IDs. You can do that by searching through `/dev/input/by-id/`. In case you device shows up as multiple devices, use `sudo cat /dev/input/by-id/<ID>`, then press some buttons or wiggle the mouse and check if anything is output. If it is, that should be the correct device.

Now we'll go on to edit the VM configuration XML file directly. This can be done by running

```
sudo EDITOR=nano virsh edit <VM_NAME>
```

There we will add the following:

Mouse + Keyboard Passthrough

At the top of the file, change

```
<domain type='kvm'>
```

to

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

and then add the following lines near the end of the file (before the closing </domain>):

```
...
<qemu:commandline>
<qemu:arg value='-object'/>
<qemu:arg value='input-linux,id=mouse1,evdev=/dev/input/by-id/MOUSE_NAME'/>
<qemu:arg value='-object'/>
<qemu:arg value='input-linux,id=kbd1,evdev=/dev/input/by-
id/KEYBOARD_NAME,grab_all=on,repeat=on'/>
</qemu:commandline>
</domain>
```

This should allow changing the keyboard and mouse from host to guest and vice-versa by pressing the left and right CTRL keys simultaneously.

Additionally, you will have to add the following to `/etc/libvirt/qemu.conf`:

```
...
user = "<USER>"
group = "kvm"
...
cgroup_device_acl = [
    "/dev/input/by-id/KEYBOARD_NAME",
    "/dev/input/by-id/MOUSE_NAME",
    "/dev/null", "/dev/full", "/dev/zero",
    "/dev/random", "/dev/urandom",
    "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
    "/dev/rtc","/dev/hpet", "/dev/sev"
]
...
```

Nvidia Driver Issues

To ensure that nvidia drivers don't cause errors from running in a VM, we'll have to add the following:

```
...
<features>
  <hyperv>
    ...
    <vendor_id state='on' value='KVM VM'/>
    ...
  </hyperv>
```

```
...
<kvm>
<hidden state='on'/>
</kvm>
</features>
...
```

Also consider adding `<ioapic driver='kvm'/>` in the `<features>` section to avoid BSODs.

Running the VM

Once the configuration is done, the VM should be ready to use.

Make sure that you have a display connected to the GPU, since the VM will only ouput using the GPU.

Once the graphics drivers are installed, the VM should have full access to the GPU.

Additional Setup

Looking Glass

To set up looking-glass, install the `looking-glass` AUR package. Then add the following to your VM XML configuration:

```
...
<devices>
    ...
  <shmem name='looking-glass'>
    <model type='ivshmem-plain'/>
    <size unit='M'>32</size>
  </shmem>
</devices>
...
```

The 32 is dependent on the resoultion to pass through. For 1080p and 1440p, 32 should be used.

Then add the following line to `/etc/tmpfiles.d/10-looking-glass.conf`:

```
f␣/dev/shm/looking-glass␣0660␣<USER>␣kvm␣-
```

and run the command

```
sudo systemd-tmpfiles --create /etc/tmpfiles.d/10-looking-glass.conf
```

Afterwards, you will have to install the driver and looking glass host. To do this, first go into Device Manager. There, go to System Devices/PCI standard RAM Controller and install the signed driver from Red Hat from here.

Afterwards install the Visual C++ Redistrubutable from here. Then install the looking glass server, downloaded from here.

Once the server is running, you should be able to use looking glass using `looking-glass-client -sF`

TODO: Figure out why it's not working right now.

SCREAM Audio

To set up scream, install the `scream` AUR package. Then add the following to your VM XML configuration:

```
...
<shmem name="scream-ivshmem">
  <model type="ivshmem-plain"/>
  <size unit="M">2</size>
</shmem>
...
```

The 2 is the identifying number and has to be the same in subsequent steps.

Then add the following line to `/etc/tmpfiles.d/10-scream-ivshmem.conf`:

```
f␣/dev/shm/scream-ivshmem␣0660␣<USER>␣kvm␣-
```

and run the command

```
sudo systemd-tmpfiles --create /etc/tmpfiles.d/10-scream-ivshmem.conf
```

Afterwards, you will have to install the driver and scream server. To do this, first go into Device Manager. There, go to System Devices/PCI standard RAM Controller and install the signed driver from Red Hat from

here.

Install the scream driver from here

Finally, add a DWORD in the regisrty editor (regedit.exe) at the following location:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Scream\Options
```

(You may need to create the Options key)

The DWORD to add is called `UseIVSHMEM` and has to have the same value as the shared memory file size (which is normally 2)

Fixing Keyboard and Mouse Performance

When using the default PS2 keyboard and mouse, it can lead to serious performance problems in games. To fix this, change

```
<input type='mouse' bus='ps2'/>
<input type='keyboard' bus='ps2'/>
```

to

```
<input type='mouse' bus='virtio'/>
<input type='keyboard' bus='virtio'/>
```

and install the drivers for the two new devices in the device manager using the drivers from here.

Note: The PS2 keyboard and mouse will be regenerated and also show up as separate devices in the device manager, but as far as I can tell, that's fine as long as the virtio devices drivers are installed correctly.

Revision #15
Created 2020-03-16 11:43:29 UTC by Hermann
Updated 2021-04-23 22:10:47 UTC by Hermann