QEMU-Puppy - A Personal Portable Computer

Table of Contents

Table of Contents

QEMU-Puppy

A Personal Portable Computer

Tue Sep 18 18:40:16 UTC 2007

Erik Veenstra <qemupuppy@erikveen.dds.nl>

1. Introduction

QEMU-Puppy is an OS and a set of applications on a USB memory stick. This OS can be booted natively, or on top of an other, already installed, OS. Just borrow a PC, boot your own environment and return the PC unaffected.

"No installation, no garbage..."

Once upon a time, I realized that having your data on a USB memory stick wasn't exactly what I wanted. I wanted more. Carrying your files is nice. Carrying your applications as well would be even better. What the heck, carrying your OS and even your own machine on your memory stick would be the best!

If you have your files on your memory stick, you can take any machine and access those files. Being able to handle the files, depends on the installed software. If you don't want to depend on the installed software, you can put your own software on the memory stick as well. But most of the software needs to be installed, which is not easy, or possible, or desirable, or allowed on a lot of machines. You don't want to pollute a machine you borrowed... Having a complete OS at hand, including applications and settings and user data would take away these constraints. Just put everything on a USB memory stick and boot from it.

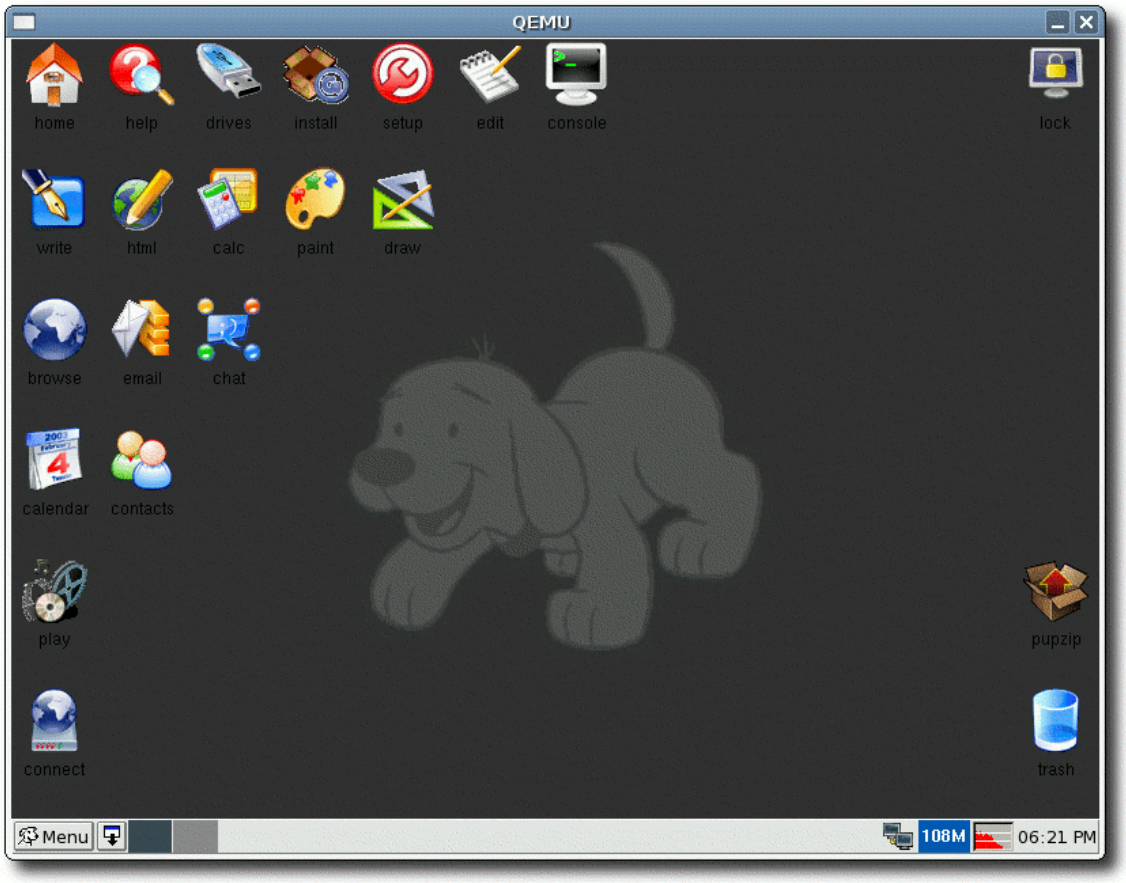
But...

A bootable USB memory stick has some disadvantages as well. First of all, not every PC is able to boot from USB. Second, if the machine does boot from USB, not all hardware is detected or configured properly, since the hardware "changes" every day. Third, booting from USB "locks" the machine: It's either the already installed OS or your OS, not both at the same time.

To get rid of these disadvantages, you can carry your own machine as well, not just your OS and your applications and your settings and your user data. You can do this by buying a laptop. But it's expensive, a physical burden and risky. The USB memory stick is cheap, light and easier to protect. A virtual machine , like QEMU, is cheap, light and easy to protect as well. With such a virtual machine, we are able to boot our OS on top of the already installed OS. Now we have two OS's running concurrently on one machine! ALT-TAB is enough to hide your machine and get back to work...

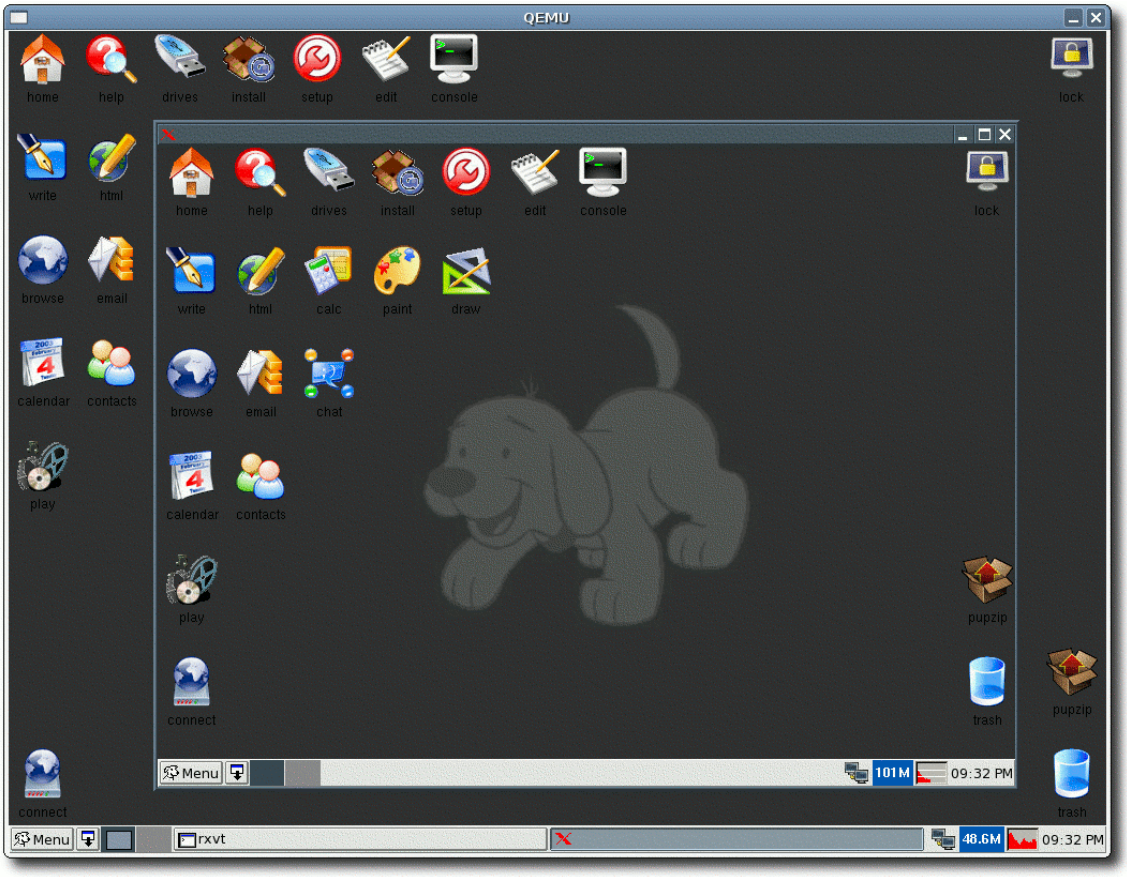
The tricky part is trying to have the OS on your USB memory stick to be able to boot **both natively and in a virtual machine**. But it can be done. And that's what makes QEMU-Puppy unique.

1. Introduction

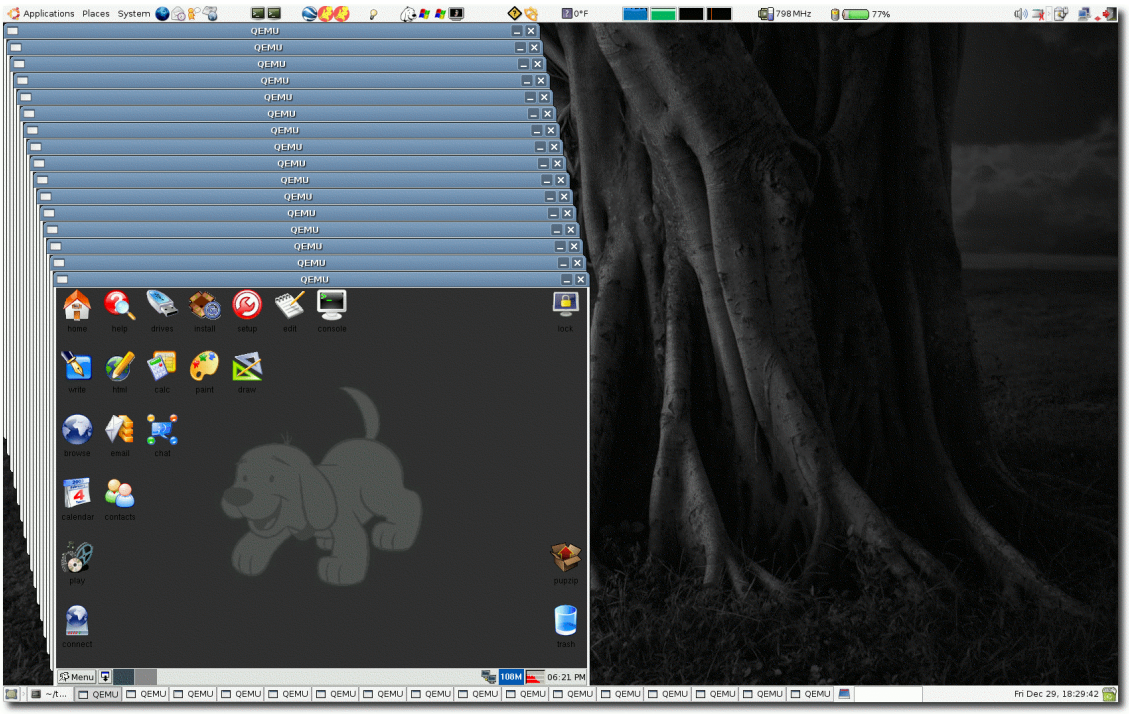


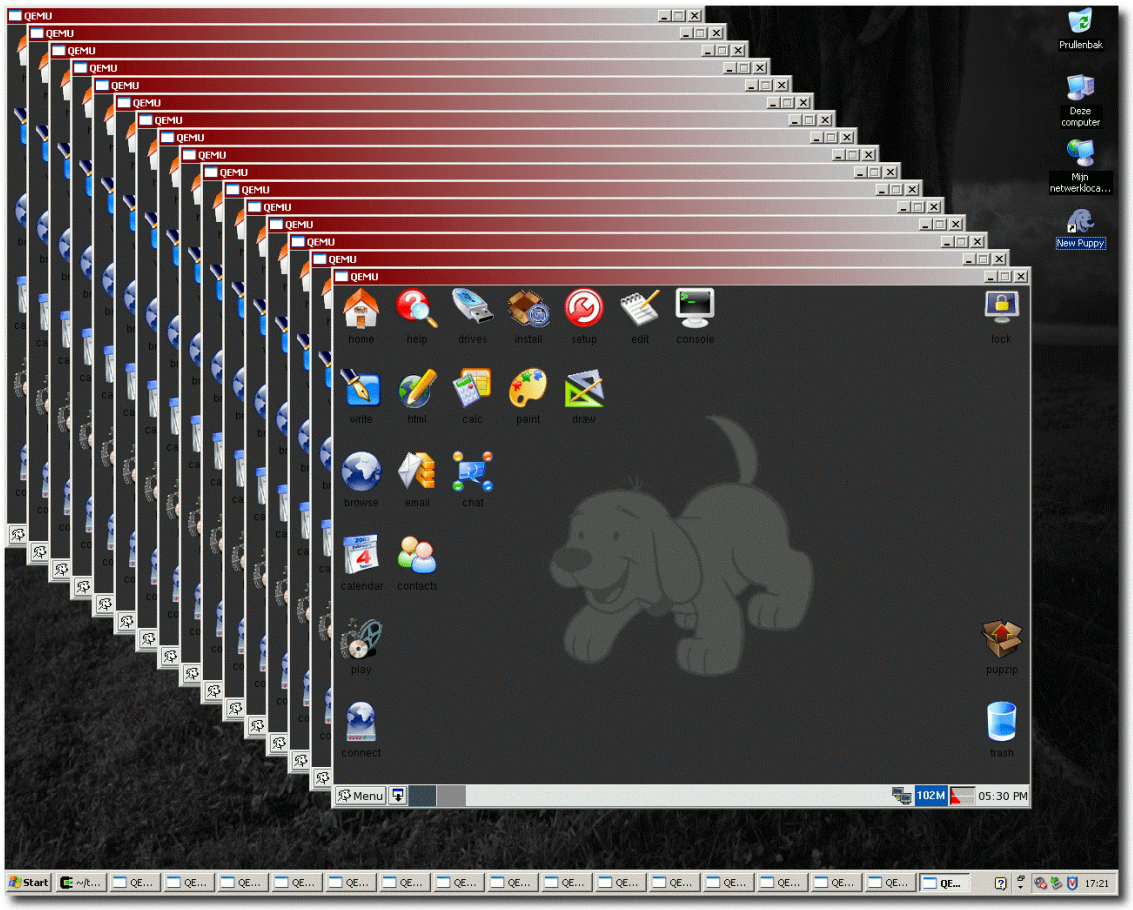
(Here you can see QEMU-Puppy, running on top of a (not visible) native Linux installation.)

1. Introduction



(Here you can see QEMU-Puppy, running in QEMU-Puppy, which in turn runs on top of a (not visible) native Linux installation. A stack of 3 Linux installations...)





(Here you can see a two baskets full of puppies. Sixteen of them on a 1 GB Linux box and another sixteen on a 2 GB Windows box.)

2. Ingredients

(These ingredients come with QEMU-Puppy. You don't have to download or install them yourself.)

2.1. Puppy Linux

There are a lot of medium sized Linux distributions out there. A lot of them could be used for our purpose. I've chosen Puppy Linux, because it's flexible, it simply works and because it "feels good"...

The mission statements of Puppy Linux are (copied from the site):

- Puppy will easily install to USB, Zip or hard drive media.
- Booting from CD, Puppy will load totally into RAM so that the CD drive is then free for other purposes.
- Booting from CD, Puppy can save everything back to the CD, no need for a hard drive.
- Booting from USB, Puppy will greatly minimize writes, to extend the life of Flash devices indefinitely.
- Puppy will be extremely friendly for Linux newbies.
- Puppy will boot up and run extraordinarily fast.
- Puppy will have all the applications needed for daily use.
- Puppy will just work, no hassles.
- Puppy will breathe new life into old PCs

See the Puppy Linux site for more information about Puppy Linux.

Version used in QEMU-Puppy: 2.17.

(I've never released a version of QEMU-Puppy based on Puppy Linux 1.0.8, because the version of JWM used in Puppy Linux 1.0.8 is terribly slow.)

2.2. QEMU

From the site: "QEMU emulates a full system (for example a PC), including a processor and various peripherals. It can be used to launch different Operating Systems without rebooting the PC or to debug system code."

In other words: QEMU is a virtual machine. A virtual machine is a program that acts like computer hardware. In such a virtual machine, you can install an OS (Linux, Windows, BSD, whatever). This OS "sees" a processor, which is the real processor (I'm lying...), "sees" a hard drive, which is a big file on the host machine, "sees" a network card, which is emulated by QEMU, and so on. After shutting down the virtual machine, all that's left on the host (or USB memory stick) is just one big file: the virtual hard disk.

From the point of view of the host OS, QEMU is just a simple program which allocates a lot of memory, eats a lot of CPU cycles and opens and closes a couple of files. That's it.

See the QEMU site for more information about QEMU.

Version used in QEMU-Puppy: 0.9.0.

2.3. SysLinux

SYSLINUX is a boot loader for the Linux operating system which operates off an MS-DOS/Windows FAT filesystem.

See the SysLinux site for more information about SysLinux.

3. Installation

3.1. QEMU-Puppy

QEMU-Puppy can be used and distributed under the terms of the GNU General Public License (GPL).

(You might want to backup your old `pup_save.3fs` first...)

1.	Download qemu-puppy-2.17-1.tar.gz (86M): `torrent:linuxtracker.org` or `http:SourceForge`.
2.	Copy all 20 files (214M) in qemu-puppy-2.17-1.tar.gz to the root of your VFAT/FAT32 formatted 256M (or more) USB memory stick.
3.	Then do something like "`syslinux /dev/sda1`" (as root) or "`D:\syslinux D:`" to make your device bootable. SysLinux puts `ldlinux.sys` on the device as well. (See the SysLinux site and PuppyLinuxWiki in case you run into trouble... Don't ask me!) [1] (If you want to boot from USB natively.)

That's it. If you double click `puppy.exe` (Windows) or run `. ./puppy.sh` (Linux), QEMU-Puppy should start. You can stop reading now, if you're satisfied...

You can check the integrity of `qemu-puppy-2.17-1.tar.gz` with `qemu-puppy-2.17-1.md5sum.txt`:

```
md5sum -vc qemu-puppy-2.17-1.md5sum.txt
```

The md5sums of the individual files (except `pup_save.3fs` and `ldlinux.sys`) of the current version are listed in `qemu-puppy-2.17-1.md5sums.txt`.

Each file in `qemu-puppy-2.17-1.tar.gz` is described in the table below.

File	Description	Origin	QPM	QPL	QPW
`allinoneqemu.exe`	Complete QEMU executable for Windows	Me (but the contents is from QEMU)			R
`allinoneqemu_linux`	Complete QEMU executable for Linux	Me (but the contents is from QEMU)		R	
`devx_217.sfs`	An empty replacement for the original `devx_217.sfs`	Me	R	R	R
`devx_217.sfs.empty`	An empty `sfs` (SquashFS)	Me			
`ftpserver_qemu.exe`	The FTP server, accessible from within the virtual machine (experimental)	Me			
`ftpserver_qemu_linux`	The FTP server, accessible	Me			

	from within the virtual machine (experimental)				
initrd.gz	Initial ramdisk (EXT2, mounted on /dev/ram0, which becomes /)	Puppy Linux (patched)	R	R	R
kqemu.inf	QEMU Accelerator Module installation file	QEMU			R
kqemu.sys	QEMU Accelerator Module	QEMU			R
ldlinux.sys	Second stage bootloader	SysLinux	R		
puppy.exe	Bootstrapping program for Windows	Me			R
puppy.sh	Bootstrapping script for Linux	Me		R	
pup_save.3fs.empty.gz	An empty pup_save.3fs	Me			
pup_save.3fs	User data (EXT3, 128M)	Me	W	W	W
pup_217.sfs	Applications (SquashFS)	Puppy Linux	R	R	R
readme.txt	Just read it...	Me			
syslinux.cfg	Boot configuration for SysLinux	Puppy Linux	R		
syslinux.exe	SysLinux	SysLinux	R		
vmlinuz	Linux kernel	Linux	R	R	R
zdrv_217.sfs	Additional Drivers	Puppy Linux	R	R	R

(QPM = QEMU-Puppy on Metal, QPL = QEMU-Puppy on Linux, QPW = QEMU-Puppy on Windows. R = Read-only, W = Read-Write)

You should end up having these files on your stick (sizes may differ):

```
$ ls -l
total 219896
-rwxr-xr-x 1 erik erik   1111600 2007-02-18 13:43 allinoneqemu.exe
-rwxr-xr-x 1 erik erik    934195 2007-02-18 08:53 allinoneqemu_linux
-rwxr-xr-x 1 erik erik      4096 2007-01-02 22:34 devx_217.sfs
-rwxr-xr-x 1 erik erik      4096 2006-12-25 00:58 devx_217.sfs.empty
-rwxr-xr-x 1 erik erik    851666 2007-01-03 17:04 ftpserver_qemu.exe
-rwxr-xr-x 1 erik erik    932445 2007-01-03 17:04 ftpserver_qemu_linux
-rwxr-xr-x 1 erik erik    975489 2007-08-07 20:11 initrd.gz
-rwxr-xr-x 1 erik erik      1616 2007-02-18 13:38 kqemu.inf
-rwxr-xr-x 1 erik erik    123939 2007-02-06 22:02 kqemu.sys
-rwxr-xr-x 1 erik erik      8236 2006-06-09 10:31 ldlinux.sys
-rwxr-xr-x 1 erik erik  65646592 2007-07-19 05:12 pup_217.sfs
-rwxr-xr-x 1 erik erik    163540 2007-08-07 11:46 puppy.exe
-rwxr-xr-x 1 erik erik       382 2007-08-07 11:46 puppy.sh
-rwxr-xr-x 1 erik erik 134217728 2007-08-07 20:11 pup_save.3fs
-rwxr-xr-x 1 erik erik    140260 2007-01-03 23:01 pup_save.3fs.empty.gz
-rwxr-xr-x 1 erik erik      1828 2007-08-07 20:21 readme.txt
-rwxr-xr-x 1 erik erik       129 2007-08-05 17:54 syslinux.cfg
-rwxr-xr-x 1 erik erik     29184 2006-09-25 22:52 syslinux.exe
-rwxr-xr-x 1 erik erik   1773256 2007-07-19 05:02 vmlinuz
-rwxr-xr-x 1 erik erik  17948672 2007-07-19 05:06 zdrv_217.sfs
```

`pup_save.3fs` contains the user data and can be mounted as a loopback device on any Linux machine. You don't depend on QEMU-Puppy to get access to your files. (That's `pup_save.3fs` and not `pup001`!)

The size of `pup_save.3fs` is 128 MB by default. You might want to replace it by a bigger one. But that does have two drawbacks. First, Puppy-Linux is designed to use half the memory (of either the physical machine or the virtual machine (QEMU)) as a ramdisk. This ramdisk is the highest (and only writable) level of the union-FS stack. On shut down, this highest level (ramdisk) is merged down to the second highest level (which is pup_save.3fs). In other words, when you get a "disk full" error in Puppy-Linux, this usually only means that the ramdisk is full, while pup_save.3fs is not yet full. Rebooting the system merges the ramdisk down to pup_save.3fs and offers you an empty ramdisk. Second, Puppy-Linux doesn't check if pup_save.3fs has enough room to successfully merge the ramdisk down to pup_save.3fs! These two issues are general Puppy-Linux issues and not QEMU-Puppy specific issues.

To create a bigger `pup_save.3fs`, do this on your native Linux system:

```
$ dd if=/dev/zero of=pup_save.3fs bs=1k count=800
$ mkfs.ext3 pup_save.3fs
```

`devx_217.sfs` is replaced by an empty version, just to save space. You don't really need its contents, anyway (unless you're a developer and want to compile programs yourself). If you want to use the real `devx_217.sfs`, simply download it from the Puppy Linux site and overwrite my empty version. Make sure that there's always a `devx_217.sfs` on your stick, either the empty one or the original one. QEMU-Puppy simply doesn't boot if it can't find one.

3.2. QEMU Accelerator Module (KQEMU)

(You don't need this to run QEMU-Puppy. But if you want to improve performance, read on...)

QEMU has 3 different speed levels:

1. QEMU
2. QEMU + KQEMU
3. QEMU + KQEMU + `-kernel-kqemu`

QEMU and KQEMU are programs. `-kernel-kqemu` is an option of QEMU.

QEMU-Puppy's default speed level is 1.

Is QEMU-Puppy a risk for your Windows environment? Well, QEMU itself (and thus QEMU-Puppy) is just another program, seen from the perspective of the native OS. It opens a couple of files, does some networking, eats a lot of CPU cycles and consumes an enormous amount of memory. In other words, an ordinary Windows program.

KQEMU on the other hand, is a driver and, as such, becomes part of the OS. That *is* a risk. (I once killed a Windows 2000 installation by doing this...) You need administrator rights to add KQEMU to your kernel.

Use KQEMU at your own risk!

Using `-kernel-kqemu` even adds some more risk to the scene: I'm able to use it on a Windows 2000 machine, but it crashes another one...

What is this QEMU Accelerator Module? From the site: "The QEMU Accelerator Module increases the speed of QEMU when a PC is emulated on a PC. It runs most of the target application code directly on the host processor to achieve near native performance."

(It's not about the KDE GUI for QEMU; it's about the QEMU Accelerator Module.)

Although KQEMU is a major improvement over plain QEMU-Puppy, it does have a couple of disadvantages:

- You need administrator rights in order to be able to load the KQEMU module.
- By loading the KQEMU module, you pollute the host machine. Even worse, you pollute its kernel, on a very low level. One of the base principles of QEMU-Puppy is to return the host machine *unaffected*.

If you have full control over the host machine, you can download and install the accelerator by hand.

On Windows:

- Use Windows explorer to navigate to your USB memory stick.
- Right-click on `kqemu.inf` and install it.
- Reboot your system. (Maybe not, probably yes...)
- Open a DOS-box and enter `net start kqemu`. You need to do this after every reboot. So, you might want to do this in your startup scripts. Or start it via Windows Services.

On Linux, follow the instructions as given by Fabrice to compile and install (both QEMU and) KQEMU yourself. KQEMU should be compiled against exactly the right kernel version. QEMU-Puppy (`puppy.sh`) knows how to find your home-built `qemu`.

On Linux, I have to run the following commands (as root) after every reboot. I don't know why, but, hey, who cares? Simply add it to `/etc/rc.local`.

```
$ mknod /dev/kqemu c 250 0
$ chmod 666 /dev/kqemu
```

For even better performance, you might want to start `puppy.sh` or `puppy.exe` with `-kernel-kqemu`. This dramatically improves performance. Really. But... I was able to kill the host OS (Windows only... Linux works fine.) on several occasions. **Use it at your own risk!** Don't blame me!

You don't need to install any additional files on you stick. Installing KQEMU is all about "upgrading" the host. It's not about upgrading QEMU-Puppy itself.

4. Booting

(You might want to read the section Configuration, before booting QEMU-Puppy for the first time.)

4.1. QEMU-Puppy on Metal (QPM)

Nothing new here. It's just plain old Puppy Linux. Simply boot your machine from the USB memory stick.

4.2. QEMU-Puppy on Linux (QPL)

To boot QEMU-Puppy on Linux, use this:

```
$ . ./puppy.sh
```

(You probably need to source the script with ".", because the permissions of `puppy.sh` can't be set properly as a result of the underlying VFAT/FAT32 file system.)

Or this:

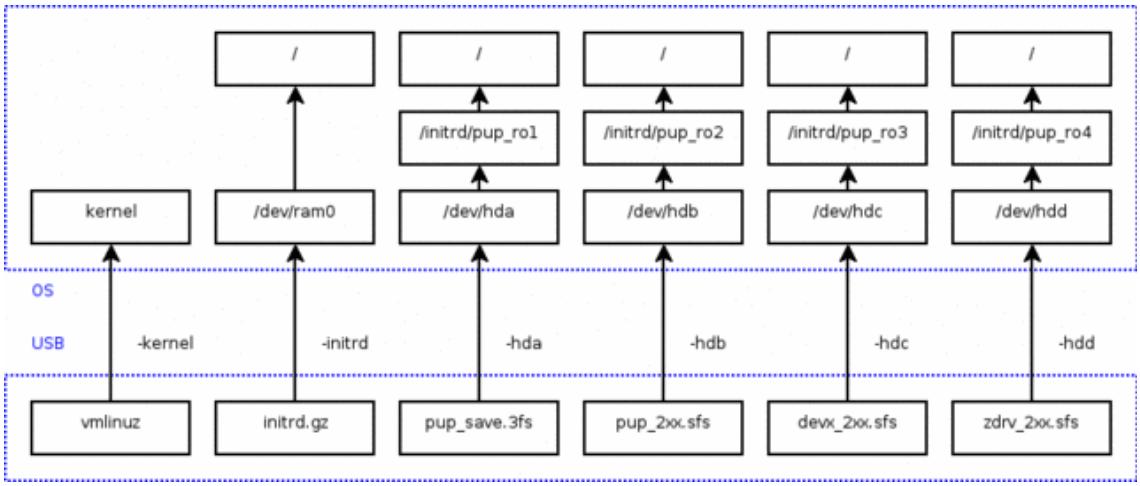
```
$ QEMU=/tmp/allinoneqemu_linux.$$

$ cp allinoneqemu_linux $QEMU
$ chmod +x $QEMU
$ trap "rm -f $QEMU" 0

$ $QEMU   -kernel vmlinuz          \
          -initrd initrd.gz        \
          -append root=/dev/ram0   \
          -hda pup_save.3fs        \
          -hdb pup_217.sfs         \
          -hdc devx_217.sfs        \
          -hdd zdrv_217.sfs        \
          -m 256                   \
          -localtime
```

(This means that QEMU is started in full system emulation mode.)

Because Puppy Linux, running in QEMU, can't access files on the USB memory stick directly, we abuse `/dev/hda` to access `pup_save.3fs`, `/dev/hdb` to access `pup_217.sfs`, `/dev/hdc` to access `devx_217.sfs` and `/dev/hdd` to access `zdrv_217.sfs`. The `rc.sysinit` in `initrd.gz` that comes with Puppy Linux is patched to handle this.



4.3. QEMU-Puppy on Windows (QPW)

To boot QEMU-Puppy on Windows, use this:

```
D:\> puppy.exe
```

Or this:

```
D:\> allinoneqemu.exe   -kernel vmlinuz          \
                        -initrd initrd.gz        \
                        -append root=/dev/ram0   \
                        -hda pup_save.3fs        \
                        -hdb pup_217.sfs         \
                        -hdc devx_217.sfs        \
                        -hdd zdrv_217.sfs        \
                        -m 256                   \
                        -localtime
```

(Since QEMU-Puppy 2.02-1, the boot mechanisms of QPL and QPW are the same.)

5. Configuration

(I only mention QEMU-Puppy specific configurations for using QEMU-Puppy in QEMU mode. The configuration of QEMU-Puppy when running on metal depends on the available hardware.)

5.1. Keyboard

Should be US, since that's the default keyboard QEMU emulates.

5.2. Video

Always use Xvesa and not Xorg. Running in 16 bit color mode is probably a bit faster than running in 24 bit color mode. I use 800x600x16 myself.

5.3. Network

a) DHCP

QEMU has a built-in DHCP server and gives QEMU-Puppy IP address 10.0.2.15.

Don't try to reach the virtual machine (10.0.2.15) from the host OS. You simply can't. QEMU is aware of the subnet it created in its own environment (10.0.2.0/24). The host OS isn't and can't be. But reaching the host machine or the Internet from within the virtual machine isn't a problem.

b) Hosts File

I added the following lines to `/etc/hosts`:

```
10.0.2.2 qemuserver
10.0.2.4 smbserver
```

Both refer to the host machine.

5.4. Clean Configuration

If you want to go back to plain old Puppy-Linux default settings, do this:

```
$ gzip -d < pup_save.3fs.empty.gz > pup_save.3fs
```

6. Tips & Tricks

6.1. File Transfer

a) FTP

FTP servers are a problem. I'll explain. It is possible to initiate a TCP/IP connection to the outside world from within QEMU. It's not possible to initiate a TCP/IP connection from the outside world to the virtual machine in QEMU (unless you run in TUN/TAP mode). That's because QEMU emulates a network and the host machine (and other machines) doesn't know and can't know about the existence of this network (10.0.2.0/24). The host machine doesn't know where to find 10.0.2.15.

The FTP protocol has two kind of connections: the control connection and data connections. The control connection is initiated by the client and stays alive throughout the whole session. The data connection comes in two flavors: active and passive. Active data connections are initiated by the FTP server, whereas passive data connections are initiated by the client. Since an FTP server on the LAN can't find the virtual machine in QEMU, we can't use active mode. So we use passive mode.

But... A big "but"... (A big butt?...)

When the client tells the server to switch to passive mode, the server replies with IP:PORT on which it listens for a data connection. This can be e.g. 192.168.2.123:2121 (a LAN IP address) or e.g. 127.0.0.1:2121 (the localhost), depending on the origin of the connection. Since QEMU masquerades the outbound connections to the local machine as local traffic, it connects to 127.0.0.1. As a response on the request, the natively running server opens a data connection on 127.0.0.1 and tells the client that it did so in its reply. The FTP client in QEMU parses this reply and tries to open a data connection to 127.0.0.1, which is the virtual machine itself and not the host! That won't work! There's no workaround, other than hacking the code of the server or building an FTP server yourself (which I did...).

I've written an (experimental) FTP server myself, which listens on port 2121 and replies with 10.0.2.2:2121 instead of 127.0.0.1:2121. That does the trick. This server should be started by hand (`ftpserver_qemu.exe` or `ftpserver_qemu_linux`); it isn't started automatically. Now the files on the pen drive (more specific: the directory in which you started the server), but outside `pup_save.3fs`, are accessible from within the virtual machine on `ftp://10.0.2.2:2121/`. (But not from the host itself!) I've tested this with gFTP and with Midnight Commander (my favorite):

```
$ wget http://www.erikveen.dds.nl/qemupuppy/dotpups/mc.pup
$ rox mc.pup
```

Start the FTP server on the native machine by double clicking on `ftpserver_qemu.exe` (or by starting `ftpserver_qemu_linux`). Once it's started, you can start your favorite FTP client, in QEMU-Puppy, and connect to the server with the following settings:

Setting	Value
Hostname	

	qemuserver *or* 10.0.2.2
Port	2121
User	*anything you want*
Password	*anything you want*

ftpserver_qemu only offers its services to all application running in QEMU-Puppy (QPL or QPW). Other clients (running natively or on another machine) can't access the server.

(The next step/experiment is to mount this FTP server on /usb_data using LUFS or FUSE.)

b) TAR + NC

My favorite way to transfer files between two machines is the combination of TAR and NC, optionally using an SSH tunnel. TAR is a tool to convert a bunch of files and directories into a byte stream, vice versa. NC (NetCat) is a tool to push a byte stream through an IP network. Combining them lets us push files and directories through a network.

Upload to QEMU-Puppy:

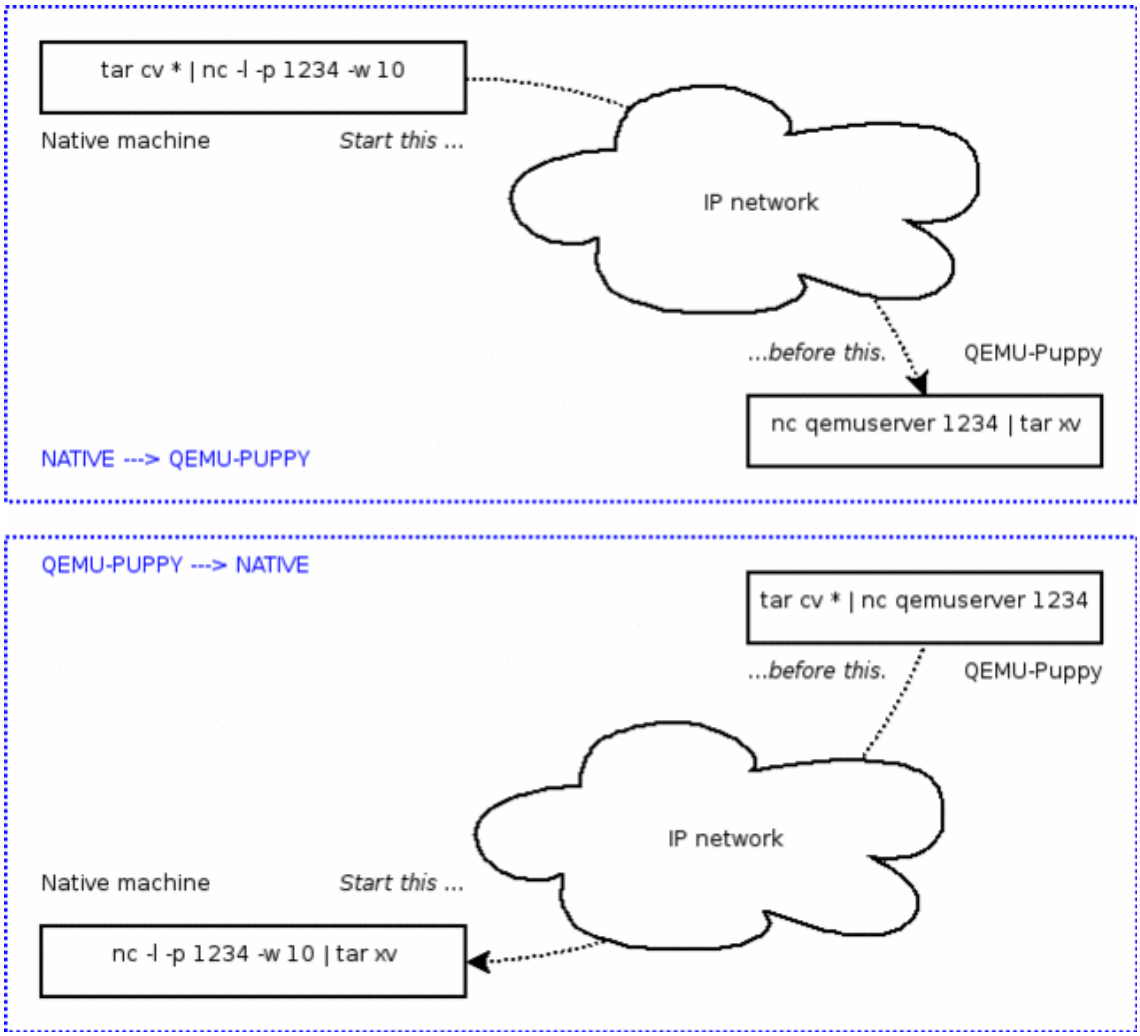
```
$ tar cv files | nc -l -p 1234 -w 10        # On host
$ nc qemuserver 1234 | tar xv               # On guest (QEMU-Puppy)
```

Download from QEMU-Puppy:

```
$ nc -l -p 1234 -w 10 | tar xv              # On host
$ tar cv files | nc qemuserver 1234         # On guest (QEMU-Puppy)
```

Linux usually comes with both TAR and NC. For Windows users, I provided tar.exe and nc.exe.

This is a general approach to moving data between two machines, platform agnostic. And it can be used over real networks as well. (Optionally using SSH for tunneling.) Once you get it, you can transfer files across networks, cross platform, through firewalls, into and out of virtual machines. No installation, no configuration (except for punching a hole in the firewall to let the SSH traffic go through). Always the same key strokes. Lightweight, simple and elegant.



c) QPL

I installed Samba on my local machine and patched `/etc/rc.d/rc.local` in my own version of QEMU-Puppy:

```
mkdir /scratch

smbmount //smbserver/qemu /scratch -o guest
```

So my host `/scratch` is shared with `/scratch` in QEMU-Puppy, when I start it like this:

```
$ . ./puppy.sh -smb /scratch
```

(Does this still work in QEMU-Puppy 2.xx?)

d) QPW

Although SMB is a Microsoft protocol and we can use it to transfer files from Linux to Linux, as described in the previous section, it's not possible to use it for sharing a directory when running in QPW mode. There are a couple of reasons for this, most noticeably the hard coded ports used for

SMB. This means that we have to use FTP or TAR + NC to transfer files from host to virtual machine and vice versa.

6.2. Resizing pup_save.3fs

a) Linux and Cygwin

```
$ ls -lh pup_save.3fs
-rwxr-xr-x 1 erik erik 128M 2007-08-29 16:43 pup_save.3fs

$ resize2fs -f pup_save.3fs 512M
resize2fs 1.39 (29-May-2006)
Resizing the filesystem on pup_save.3fs to 524288 (1k) blocks.
The filesystem on pup_save.3fs is now 524288 blocks long.

$ ls -lh pup_save.3fs
-rwxr-xr-x 1 erik erik 512M 2007-08-29 16:45 pup_save.3fs
```

b) Windows

(If you're a Cygwin user, see the previous section.)

On Windows, you're pretty much on your own. I'm not a Windows user. I can't help you. Really, I can't...

But I do have a couple of links:

http://www.acc.umu.se/~bosse/

http://www.fs-driver.org/index.html

http://colinux.wikia.com/wiki/ExpandingRoot

6.3. Installing other Software

Sometimes, you don't want to depend on DotPup or PupGet... And compiling everything yourself isn't funny, either...

In case we have an other Linux system on our network, e.g. QEMU-Puppy in QPL mode, we could try to copy the files of an application, e.g. Ruby, to the right locations on the Puppy Linux system. If the versions of the libraries and the kernel of the source system are close enough to the ones of Puppy Linux, it might even be possible to run the "imported" application. So, let's try it.

First problem: Which files do we need to copy?

With the following commands, we can guess the files that belong to the application:

```
$ APP=ruby
$ ls -d /usr/*/$APP* /usr/local/*/$APP*
```

Because an executable usually depends on some libraries, we investigate them as well:

```
$ ldd $(which $APP)
```

And because libraries usually depend on other libraries, we do it recursively.

Notice that these libraries are often symbolic links to other files. Don't forget to copy them as well.

It might be a good idea to exclude the files that already exist in Puppy Linux.

According to this and this, the files listed below should be available on a Linux Standard Base system. So we can skip them.

- libX11.so.6
- libXt.so.6
- libGL.so.1
- libXext.so.6
- libICE.so.6
- libSM.so.6
- libdl.so.2
- libcrypt.so.1
- libz.so.1
- libncurses.so.5
- libutil.so.1
- libpthread.so.0
- libpam.so.0
- libgcc_s.so.1

- libm.so.6
- libdl.so.2
- libcrypt.so.1
- libc.so.6
- libpthread.so.0
- ld-lsb.so.1

On my system, we have to import the following file and directory to install Ruby:

- /usr/local/bin/ruby
- /usr/local/lib/ruby/

Second problem: How do we import the files?

We can use one of the methods already described, or we can use the following scripts in Puppy Linux to import the files. `importroot` imports files relative to the root (= `/`) on both systems. `importhome` imports files relative to the home directories on both systems.

```
$ echo 'ssh erik@qemuserver tar c -C / $* | tar x -C /' > importroot
$ echo 'ssh erik@qemuserver tar c      $* | tar x -C ~' > importhome
$ chmod +x importroot
$ chmod +x importhome
```

In this case, we have to use `importroot`:

```
$ ./importroot /usr/local/bin/ruby /usr/local/lib/ruby/
```

In theory, we now have installed Ruby:

```
$ ruby -v
ruby 1.8.3 (2005-09-21) [i686-linux]
```

So, yes, we did...

I did the same trick for a couple of other applications, of which the files are listed below.

Oh, by the way, my host system runs Ubuntu Linux 5.04.

a) Ruby

Import the following files to install Ruby:

- /usr/local/bin/ruby
- /usr/local/lib/ruby/

b) Vim

Import the following files to install Vim:

- /usr/bin/vim
- /usr/bin/vimdiff
- /usr/bin/vimtutor
- /usr/lib/libgpm.so.1
- /usr/lib/libgpm.so.1.19.6
- /usr/share/vim/

We want to get rid of the old version of VI, which comes with Puppy Linux:

```
$ mv /bin/vi /bin/vi.old
$ ln -sf /usr/bin/vim /bin/vi
```

And because all files in `/bin` and `/lib` are lost after a reboot, we have to add these lines to `/etc/rc.d/rc.local` as well:

```
$ echo ''                              >> /etc/rc.d/rc.local
$ echo 'mv /bin/vi /bin/vi.old'        >> /etc/rc.d/rc.local
$ echo 'ln -sf /usr/bin/vim /bin/vi'   >> /etc/rc.d/rc.local
```

c) HTMLDOC

Import the following files to install HTMLDOC:

- /usr/bin/htmldoc
- /usr/lib/libexpat.so.1
- /usr/lib/libexpat.so.1.0.0

- /usr/lib/libfltk_images.so.1.1
- /usr/lib/libfltk.so.1.1
- /usr/lib/libXft.so.2
- /usr/lib/libXft.so.2.1.1
- /usr/lib/libXrender.so.1
- /usr/lib/libXrender.so.1.3.0
- /usr/share/htmldoc/

Because Puppy Linux comes with `libpng10.so.0.1.0.15` instead of `libpng10.so.0.1.0.18`, which is required by HTMLDOC, I imported the following files as well:

- /usr/lib/libpng10.so.0
- /usr/lib/libpng10.so.0.1.0.18

6.4. Cloning

I use the following script to create a temporary instance of QEMU-Puppy on my hard drive. Kind of cloning... All puppies can live side-by-side, concurrently, in harmony... See the screenshots in Introduction.

```
$ puppyclone test_ruby_projects
```

Did you know that each clone on my machine only costs me about 172K of disk space (without the user data)?

```
ID="$1" ; shift

TEMPDIR=~/tmp                                                  # MANDATORY
CLONEDIR=$TEMPDIR/puppyclone.$ID                               # MANDATORY
EXTRACTEDDIR=~/projects/qemupuppy2/main/usb                    # MANDATORY, this
PUPPYTARGZ=~/projects/qemupuppy2/results/qemu-puppy.tar.gz     # MANDATORY, or this
DEVX=~/attic/devx_215.sfs                                      # OPTIONAL

[ ! "$ID" ] && {
  echo "Usage: $(basename $0) ID [--generate]"

  exit 1
}

[ "$1" = "--delete" ] && {
  rm -rf $CLONEDIR

  echo "Puppy $ID has died."

  exit 0
}

[ "$1" = "--generate" ] && {
  echo "Just a minute..."

  rm -rf $CLONEDIR      || exit 15
  mkdir -p $CLONEDIR    || exit 15
```

```
  if [ -d "$EXTRACTEDDIR" ]; then
    for f in $EXTRACTEDDIR/*; do
      [ ! -d "C:/" ] && ln -sf $f $CLONEDIR/
      [   -d "C:/" ] && cp -af $f $CLONEDIR/
    done

    rm $CLONEDIR/pup_save.3fs
    cp -af $EXTRACTEDDIR/pup_save.3fs $CLONEDIR/pup_save.3fs
    # ??? cp -af ~/Desktop/pup_save_empty.3fs $CLONEDIR/pup_save.3fs    # ???
  else
    nice tar xzf $PUPPYTARGZ -C $CLONEDIR
  fi

  [ "$DEVX" ] && {
    rm -f $CLONEDIR/$(basename $DEVX)

    [ ! -d "C:/" ] && ln -sf $DEVX $CLONEDIR/devx_215.sfs
    [   -d "C:/" ] && cp -af $DEVX $CLONEDIR/devx_215.sfs
  }

  echo "Puppy $ID is born."

  exit 0
}

[ ! -d "$CLONEDIR" ] && {
  $0 $ID --generate
}

cd $CLONEDIR    || exit 16

chmod +x puppy.sh

[ ! -d "C:/" ] &&   ./puppy.sh -kernel-kqemu -usbdevice tablet -smb /scratch $*
[ -d "C:/"   ] &&   ./puppy.exe -usbdevice tablet $*
```

(This script is built for my environment. It doesn't necessarily work in yours.)

7. DotPups

It's not directly related to QEMU-Puppy, but I've built a couple of DotPups myself:

Download	Depends on	Package	Version	More information and Licenses	Built
htmldoc.pup	-	HTMLDOC	1.8.27	http://www.htmldoc.org/	19 January 2007
kqemu.pup	-	KQEMU	1.3.0pre11	http://www.qemu.org/	7 February 2007
mc.pup	-	GNU Midnight Commander	4.6.1	http://www.ibiblio.org/mc/	19 January 2007
qemu.pup	sdl.pup	QEMU	0.9.0	http://www.qemu.org/	7 February 2007
ruby.pup	-	Ruby	1.8.5-p2	http://www.ruby-lang.org/en/	19 January 2007
sdl.pup	-	SDL	1.2.11	http://www.libsdl.org/	7 February 2007
vim.pup	-	VIM	7.0	http://www.vim.org/	19 January 2007
socat.pup	-	Socat	2.0.0-b1	http://www.dest-unreach.org/socat/	6 July 2007

Download a DotPup and install it with:

```
$ wget http://www.erikveen.dds.nl/qemupuppy/dotpups/something.pup
$ rox something.pup
```

8. Internals

8.1. Patches of Puppy Linux

`initrd.gz` contains the startup scripts of Puppy Linux. One of these scripts, `/sbin/init`, searches for `pup_save.3fs`, `pup_217.sfs`, `devx_217.sfs` and `zdrv_217.sfs`. I patched this script, so Puppy Linux knows where to look for these files in case it is booted in QEMU (`/dev/hda`, `/dev/hdb`, `/dev/hdc` and `/dev/hdd`, respectively (see `abuse`)). The old functionality (for booting from USB directly) is not affected. This patch is (for now...) kind of a proof-of-concept and isn't really nice at all. Hopefully, the new functionality will be added to Puppy Linux itself. Barry knows how to do it properly...

Puppy Linux sets a couple of variables in `/etc/rc.d/PUPSTATE`. In QPL and QPW mode, these are left empty, except for `PUPMODE` (which is set to `13`) and `QPM` (which is QEMU-Puppy specific).

I also added keyboard, video and network settings, which you probably had to do yourself in order to be able to use QEMU-Puppy in QEMU mode. These settings are described in Configuration.

8.2. AllInOneQEMU

I used EEE to create AllInOneQEMU. EEE embeds all files QEMU needs to be able to run: `qemu.exe`, `sdl.dll` and a couple of BIN-files. Have a look at the QEMU site for licenses and sources.

`allinoneqemu.exe` and `allinoneqemu_linux` can be used like `qemu.exe` itself. At least, that's my goal...

This is the EEE-file to build AllInOneQEMU for Windows. This is the EEE-file to build AllInOneQEMU for Linux.

If you are on Linux and you've setup a QEMU environment yourself and `qemu` is available on your `$PATH`, then your installation of QEMU will be used instead of `allinoneqemu_linux`. This allows you to setup an environment with the QEMU Accelerator Module, which gives your Puppy a bit more speed.

8.3. puppy.exe

This is the EEE-file to build `puppy.exe`. For Linux, just use `puppy.sh` (see QEMU-Puppy on Linux (QPL)). There's no need for `puppy_linux`.

9. License

10. Known Issues

- QEMU-Puppy probably won't boot natively (QPM) from USB memory sticks bigger than 1 GB. Quote from the SysLinux site: "SysLinux will not work (and will refuse to install) on filesystems with a cluster size of more than 16K (typically means a filesystem of more than 1 GB.)"
- QEMU-Puppy 1.0.9-1 introduced an "*invisible box*" in which the mouse seems to be trapped. Making big, aggressive circles with the mouse usually frees the mouse from its jail.
- When running in QPW mode, "*grabbing keyboard and mouse*" doesn't always mean "*receiving keyboard and mouse events*". Somehow, the events might get queued. Doing a lot of alt-tabs usually helps. It's probably a QEMU problem.
- The FTP-server doesn't work with Mozilla. Use Gftp.

Notes

[1] *The Windows version of SysLinux is included in QEMU-Puppy; the Linux version isn't, since Linux usually comes with SysLinux.*