How to install Xenomai?

This tutorial will detail Xenomai installation on a small Dell Optiplex 990 desktop having 16GB of RAM and an Intel Core i7-2600 CPU @ 3.40GHz × 8 cores. The machine has two 128GB Samsung SSD's. For the Ubuntu installation, 100GB of the first drive was dedicated to the operating system (mount point /) and the rest was left as swap space. On the second drive, 75GB was dedicated to user home directories (mount point /Volumes/<computer_name>) and the rest was left to backup space (mount point /Volumes/backup).

Step-by-step guide for Installing Linux 3.8.13 + Xenomai 2.6.3 (Ubuntu 12.04)

Prerequisites

First, install Ubuntu 12.04 following the guide on the CLMC wiki: http://www-clmc.usc.edu/wiki/projects/ubuntucomputersetup/Ubuntu_Computer_Setup.html

In addition to the above Ubuntu installation, make sure to do the following:

- Install ROS Groovy (http://wiki.ros.org/groovy/Installation/Ubuntu)
- Install the boost libraries, if not already installed
- Install the hermes/athena code, if working with these robots (How to install code for Hermes / Athena)

You may also want to install the following packages:

```
apt-get update
apt-get install devscripts debhelper dh-kpatches findutils kernel-package
libncurses-dev fakeroot zlib1g-dev autotools-dev autoconf automake libtool git
```

This installation will mostly follow the official Xenomai installation guide but for reference we provide a deprecated guide written by Peter for installing an older version of Xenomai on the ARM robot (https://www-clmc.usc.edu/wiki/pages/J2F7q3v/Xenomai_256_install_instructions.html)

Building a Xenomai-patched Linux kernel package

Downloading the sources/configuring xenomai

As root download Xenomai (as a Debian package) and the Linux Kernel (I used a vanilla kernel and put all the sources in /usr/src). For details on

debian packages, see http://www.xenomai.org/index.php/Building_Debian_packages

```
cd /usr/src

# download xenomai
wget -O - http://download.gna.org/xenomai/stable/xenomai-2.6.3.tar.bz2 | tar -jxf -
cd xenomai-2.6.3

# create a new debian changelog entry and build the packages in the parent directory
DEBEMAIL="your@email" DEBFULLNAME="Your Name" debchange -v 2.6.3 Release 2.6.3
debuild -uc -us

# install the resulting packages
cd ..
dpkg -i libxenomai1_2.6.3_amd64.deb libxenomai-dev_2.6.3_amd64.deb
xenomai-doc_2.6.3_all.deb xenomai-kernel-source_2.6.3_all.deb
xenomai-runtime_2.6.3_amd64.deb
```

The packages contain, respectively:

- xenomai-runtime - *Xenomai runtime utilities*
- libxenomai1 - *Shared libraries for Xenomai*
- libxenomai-dev - *Headers and static libraries for Xenomai*
- xenomai-doc - *Xenomai documentation*
- xenomai-kernel-source - *Patches and goodies for building the linux kernel*

The list of Xenomai-supported Linux kernels can be found in /usr/src/xenomai-kernel-source/ksrc/arch/x86/patches/ Here, we use kernel version 3.8.13 from kernel.org. Download and unzip this kernel and then patch it with xenomai as shown below. Note that we need to apply a patch to the xenomai source code to fix a latency issue created by running *xeno latency* in parallel with a *watch /proc/xenomai/stat* command (see http://www.xenomai.org/pipermail/xenomai/2014-March/030303.html for more details).

```
# first we download the linux kernel
cd /usr/src
wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.8.13.tar.bz2
tar -jxf linux-3.8.13.tar.bz2

# create symbolic links
ln -s /usr/src/linux-3.8.13 linux
ln -s /usr/src/xenomai-2.6.3 xenomai

# we need to patch then provided xenomai source to fix a latency issue
cd xenomai
wget --no-check-certificate https://www-clmc.usc.edu/~nrotella/Xenomai/latency.patch
patch -p1 < latency.patch

# finally, patch the linux kernel for xenomai (making sure that the adeos patch
specified matches the linux kernel version)
/usr/src/xenomai/scripts/prepare-kernel.sh --linux=/usr/src/linux
--ipipe=/usr/src/xenomai/ksrc/arch/x86/patches/ipipe-core-3.8.13-x86-4.patch
```

Kernel configuration and compilation

You can use the following kernel config if you want config_kernel_3.8.13_xeno_2.6.3, copy it to /usr/src/linux/.config and then configure the kernel:

```
cd /usr/src/linux

# obtain the old kernel config file (optional)
wget --no-check-certificate
https://www-clmc.usc.edu/~nrotella/Xenomai/config_kernel_3.8.13_xeno_2.6.3
mv config_kernel_3.8.13_xeno_2.6.3 /usr/src/linux/.config

# run the kernel config GUI and double-check options below
make menuconfig
```

You need the following options; they should be set correctly if you used the old config file but it's good to double-check. Scrolling to the option and hitting Y, N, or M (all lower-case) will Enable, Disable, or Enable (build as a kernel module) that option, respectively. If there is no asterisk in the brackets, the option is already disabled. Brackets [] mean the option can be enabled (Y) while brackets <> mean it can be enabled (Y) or enabled to be built as a kernel module (M).

Kernel Modules
When the kernel is compiled, modules (sometimes also called "drivers") are either built as loadable (=m) or built-in (=y) and are marked as such in the kernel config file (/usr/src/<kernel-name>/.config). While loadable modules can be unloaded and even blacklisted at boot time (by editing /etc/modprobe.d/blacklist), there is no (simple) way to unload a built-in module without first recompiling the kernel to make it loadable. This means that anything you might wish to load/unload on the fly should be marked with (=m).

While *modprobe -l* lists all loadable modules, you can check the file /lib/modules/<kernel-name>/modules.builtin to find built-in modules. You can get information about a module (including its location) using *modinfo.* You can check if particular modules are loaded with *lsmod* and load/unload them with *insmod/rmmod* or using *modprobe.* Note that all modules are actually installed in /lib/modules/<kernel-name> and modules can be loaded at boot time by adding them to /etc/modules.

The kernel configuration options to check are listed below. You can always edit .config directly if you mess up in the GUI (see http://www.tldp.org/HOWTO/SCSI-2.4-HOWTO/kconfig.html for more configuration details):

- Choose the correct processor (in Processor Type -> Processor Family) - I chose Core 2 / newer Xeon for an i7 machine.
- CONFIG_CPU_FREQ - Disable (in Power management->CPU Freq Scaling)
- CONFIG_CPU_IDLE - Disable (in Power management -> CPU idle PM support)
- CONFIG_ACPI_PROCESSOR - Disable(in Power management->ACPI->processor)
- CONFIG_INPUT_PCSPKR - Disable (I changed that directly in the .config file after being done with menuconfig because I couldn't find the option)
- DO NOT DISABLE MSI - It is now obsolete (cf. http://permalink.gmane.org/gmane.linux.real-time.xenomai.users/19782)
- Under Real-time Subsystem, you may want to enable (mark with M to build as a kernel module) the real-time serial driver under Drivers.

These and more Xenomai installation details can be found at http://www.xenomai.org/documentation/xenomai-2.6/html/README.INSTALL/ (official Xenomai install guide) and http://xenomai.org/2014/06/configuring-for-x86-based-dual-kernels/ (explains how to configure the installation options).

Once configuration is done, you can exit the GUI and save the configuration file and compile the kernel:

```
# compile the kernel (Concurrency level is the number of cores - do not use make -j)
CONCURRENCY_LEVEL=8 CLEAN_SOURCE=no fakeroot make-kpkg --initrd --append-to-version
-ipipe-xenomai-2.6.3 --revision 1.0 kernel_image kernel_headers
```

This may take some time (on the order of tens of minutes). Once compilation completes, in /usr/src you should have two new debian packages: *linux-image-3.8.13-ipipe-xenomai-2.6.3_1.0_amd64.deb* and *linux-headers-3.8.13-ipipe-xenomai-2.6.3_1.0_amd64.deb* corresponding the kernel source and headers.

First, add yourself to the xenomai group so you have non-root access to real-time functions by issuing

```
usermod -a -G xenomai <username>
```

and then modify the grub config (the file is called /etc/default/grub) to look as follows:

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'
GRUB_DEFAULT="2>0"
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash xeno_nucleus.xenomai_gid=125"
GRUB_CMDLINE_LINUX=""
# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"
# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console
# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480
# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true
# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"
# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"
```

The necessary changes are:

- In the line GRUB_DEFAULT: "2>0" tells the system to automatically boot into "Previous Versions > First Kernel" (this may not be necessary, depending on where the xenomai-patched kernel ends up in the grub menu).
- In the line GRUB_CMDLINE_LINUX_DEFAULT: this change is needed to pass the xenomai group id as a boot parameter so that this group can run RT commands. This is the usual place to pass arguments to the kernel.
- Optionally, you may need to disable your proprietary graphics driver by adding "nomodeset" after "quiet splash" in the same line as above. See the section on graphics drivers for more info.

Save /etc/default/grub, then run

```
update-grub
```

to update the bootloader. Finally, you must install the kernel with

```
cd ..
dpkg -i linux-image-3.8.13-ipipe-xenomai-2.6.3_1.0_amd64.deb
```


Note that if you wish to recompile the kernel in the future (for example, with different config parameters), you can use *dpkg --list | grep linux-image* to find currently-installed kernels and then use *dpkg --purge <kernel-name>* to remove the old kernel before proceeding from the kernel config step for the new kernel. For example, the remove the kernel we're about to install, one would use *dpkg --purge linux-image-3.8.13-ipipe-xenomai-2.6.3*

Reboot the machine and select the appropriate kernel with grub. When booting a correctly configured kernel, you should see messages resembling these in the kernel logs (run "dmesg | grep Xenomai"). If not, run *uname -r* to show you the loaded kernel name; you probably accidentally booted into the wrong one. If you have any issues with the Xenomai installation at this point, consult http://www.xenomai.org/documentation/xenomai-2.6/html/TROUBLESHOOTING/

```
[    2.023815] I-pipe: head domain Xenomai registered.
[    2.023890] Xenomai: hal/x86_64 started.
[    2.023912] Xenomai: scheduling class idle registered.
[    2.023913] Xenomai: scheduling class rt registered.
[    2.024856] Xenomai: real-time nucleus v2.6.3 (Lies and Truths) loaded.
[    2.024857] Xenomai: debug mode enabled.
[    2.025042] Xenomai: SMI-enabled chipset found, but SMI workaround disabled
[    2.025116] Xenomai: starting native API services.
[    2.025117] Xenomai: starting POSIX services.
[    2.025130] Xenomai: starting RTDM services.
```

Testing for latency issues

Run "xeno latency" to test the latency and play a bit with the graphics to see how it works. You should see results like:

```
== Sampling period: 100 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      1.615|      1.923|      9.846|       0|     0|      1.615|      9.846
RTD|      1.615|      1.923|      9.692|       0|     0|      1.615|      9.846
RTD|      1.538|      1.923|     10.230|       0|     0|      1.538|     10.230
RTD|      1.615|      1.923|     10.384|       0|     0|      1.538|     10.384
RTD|      1.615|      1.923|     11.230|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|      9.923|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|      9.923|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|     11.076|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|     10.538|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|     11.076|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|     10.615|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|     10.076|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|      9.923|       0|     0|      1.538|     11.230
RTD|      1.538|      1.923|     10.538|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|     10.923|       0|     0|      1.538|     11.230
RTD|      1.538|      1.923|     10.153|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|      9.615|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|     10.769|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|      9.153|       0|     0|      1.538|     11.230
RTD|      1.538|      1.923|     10.307|       0|     0|      1.538|     11.230
RTD|      1.615|      1.923|      9.538|       0|     0|      1.538|     11.230
RTT|  00:00:22  (periodic user-mode task, 100 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      1.615|      1.923|     11.384|       0|     0|      1.538|     11.384
RTD|      1.615|      1.923|     10.076|       0|     0|      1.538|     11.384
RTD|      1.538|      1.923|      9.538|       0|     0|      1.538|     11.384
---|-----------|-----------|-----------|--------|------|-------------------------
RTS|      1.538|      1.923|     11.384|       0|     0|    00:00:25/00:00:25
#
```

Normal values for "lat max" are from around 0-10 (microseconds) but this will vary; anything above, say, 40 is an indicator of something bad happening. It's especially important to verify that *xeno latency* outputs reasonable latencies while (A) using the "watch" command in parallel and (B) running SL.

- For (A) run the test in one terminal and, from a second, run *watch -n 0.1 cat /proc/xenomai/stat* and check the latency test results while

doing so. If you see a spike when you start watching, then the latency patch didn't work.
- For (B) install SL* and run any simulation (for example, xhermes) while running the latency test in another terminal. Move the windows around; if you see a spike in latencies, there is an issue with your graphics driver and its compatibility with X-window. Check the following section for solutions.
- If both of the above tests work fine (latencies look normal) then run SL while watching and executing the test. If all looks good, your installation was successful!

*You won't actually be able to compile SL on a Xenomai system until after installing RTNet because cb_communication depends on it. Wait until then to try this test.

Graphics drivers and Xenomai

NVIDIA GF119

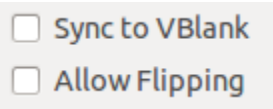
On our Dell Precision Tower T7810 we have a NVIDIA GF119 [NVS 310] graphics card with the following specs:

```
description: VGA compatible controller
product: GF119 [NVS 310]
vendor: NVIDIA Corporation
physical id: 0
bus info: pci@0000:02:00.0
version: a1
width: 64 bits
clock: 33MHz
capabilities: pm msi pciexpress vga_controller bus_master cap_list rom
configuration: driver=nvidia latency=0
resources: irq:92 memory:d6000000-d6ffffff memory:c8000000-cfffffff
memory:d0000000-d1ffffff ioport:6000(size=128) memory:d7000000-d707ffff
```

Right after installing Xenomai we have irregularly appearing latencies, when running 'xeno latency' in one terminal and 'glxgears' in another. In order to resolve this issue, we first need to reinstall the nvidia drivers and adapt some settings.

```
sudo apt-get install nvidia-current
nvidia-settings
```

A window will pop-up with several tabs on the left-hand side. Go through the tabs and make sure that following options are unchecked:



Now, we see a spike in latencies the moment we start 'glxgears' or close the window, but there are no spikes in-between.

Intel Onboard Graphics i915

On our Dell machines, we had issues with the Intel onboard graphics chip and proprietary driver (i915) causing latency spikes when running SL (due to X-window). The graphics chip specs are listed below.

```
00:02.0 VGA compatible controller: Intel Corporation 2nd Generation Core Processor
Family Integrated Graphics Controller (rev 09) (prog-if 00 [VGA controller])
    Subsystem: Dell Device 047e
    Flags: fast devsel, IRQ 16
    Memory at e0c00000 (64-bit, non-prefetchable) [size=4M]
    Memory at d0000000 (64-bit, prefetchable) [size=256M]
    I/O ports at 6000 [size=64]
    Expansion ROM at <unassigned> [disabled]
    Capabilities: [90] MSI: Enable- Count=1/1 Maskable- 64bit-
    Capabilities: [d0] Power Management version 2
    Capabilities: [a4] PCI Advanced Features
    Kernel modules: i915
```

One solution in such a case is to use an external graphics card; we were able to resolve the issue using an AMD card and associated drivers instead. However, we also lacked the PCI slot space in these small machines for a graphics card once we installed ethernet and USB 1.1 cards. As a result, we ended up removing the external graphics card and using the Xorg VESA drivers with the above Intel motherboard graphics chip. This was accomplished by adding the parameter "nomodeset" to the grub command line arguments (right after "quiet splash") in the file /etc/default/grub. If you use this workaround, you may also need to add the line "GRUB_GFXMODE=1280x1024x24" later in this file in order to get decent resolution on a larger monitor.

Installing RT-NET

Downloading and installation

(This section largely follows Jeannette's explanation at https://github.com/jbohg/kuka_lwr_rtnet but with small variations)

Download and RT-NET (I used the master branch from sourceforge) as follows:

```
# again as root
cd /usr/src
git clone git://git.code.sf.net/p/rtnet/code rtnet
cd rtnet
git checkout 7c8ba10513fe7b63873f753ab22d340bf44119a2
```

Next, configure the installation in the same way you configured the linux kernel (I copied rtnet_config to /usr/src/rtnet/.config to start with). You should make sure to do the following:

- Increase the maximum routing table entries to 64 (or however many you'll need) in Protocol Stack
- Enable real-time ethernet capturing (which builds rtcap.ko) in Add-Ons

Run the following to configure and install rtnet:

```
# download the old config file (optional)
wget --no-check-certificate https://www-clmc.usc.edu/~nrotella/Xenomai/rtnet_config
mv rtnet_config .config

#configure rtnet as needed and then compile and install
make menuconfig
make && make install

# copy the rtnet udev config
cp tools/00-rtnet.rules /etc/udev/rules.d/
```

Network configuration

At this point, you will lose network connectivity temporarily (until you edit the following two files and reboot).

Make sure to remove the network manager so that it doesn't interfere with the manual network configuration we do below.

```
apt-get remove network-manager
```

Now, setup the network interfaces and udev rules by editing the file /etc/udev/rules.d/70-persistent-net.rules. On hermes (which has a 2-port ethernet card installed) it looks like

/etc/udev/rules.d/70-persistent-net.rules

```
File Edit Options Buffers Tools Help

# This file was automatically generated by the /lib/udev/write_net_rules
# program, run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single
# line, and change only the value of the NAME= key.

# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:19.0 (e1000e)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="78:2b:cb:9e:38:fa",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="e\
th*", NAME="eth0"

# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:01.0/0000:01:00.1 (e1000e)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:15:17:dc:5d:9d",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="e\
th*", NAME="eth2"

# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0 (e1000e)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:15:17:dc:5d:9c",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="e\
th*", NAME="eth1"
```

We see that there are three ethernet interfaces: two from the external card and one built into the motherboard. How do we tell them apart? It's usually clear which two (or more) belong to the same card because they have contiguous hex MAC addresses (ATTR{address}). We can double-check with

/etc/network/interfaces

```
lshw -C network
```

to list network hardware which is currently installed. On hermes, this returns

```
root@hermes:/usr/src/rtnet# lshw -C Network
  *-network:0 DISABLED
       description: Ethernet interface
       product: 82571EB Gigabit Ethernet Controller
       vendor: Intel Corporation
       physical id: 0
       bus info: pci@0000:01:00.0
       logical name: eth1
       version: 06
       serial: 00:15:17:dc:5d:9c
       capacity: 1Gbit/s
```

```
       width: 32 bits
       clock: 33MHz
       capabilities: pm msi pciexpress bus_master cap_list rom ethernet physical tp
10bt 10bt-fd 100bt 100bt-fd 1000bt-fd autonegotiation
       configuration: autonegotiation=on broadcast=yes driver=e1000e
driverversion=2.1.4-k firmware=5.11-2 latency=0 link=no multicast=yes port=twisted
pair
       resources: irq:43 memory:e1ba0000-e1bbffff memory:e1b80000-e1b9ffff
ioport:4020(size=32) memory:e1b60000-e1b7ffff
  *-network:1 DISABLED
       description: Ethernet interface
       product: 82571EB Gigabit Ethernet Controller
       vendor: Intel Corporation
       physical id: 0.1
       bus info: pci@0000:01:00.1
       logical name: eth2
       version: 06
       serial: 00:15:17:dc:5d:9d
       capacity: 1Gbit/s
       width: 32 bits
       clock: 33MHz
       capabilities: pm msi pciexpress bus_master cap_list rom ethernet physical tp
10bt 10bt-fd 100bt 100bt-fd 1000bt-fd autonegotiation
       configuration: autonegotiation=on broadcast=yes driver=e1000e
driverversion=2.1.4-k firmware=5.11-2 latency=0 link=no multicast=yes port=twisted
pair
       resources: irq:44 memory:e1b40000-e1b5ffff memory:e1b20000-e1b3ffff
ioport:4000(size=32) memory:e1b00000-e1b1ffff
  *-network
       description: Ethernet interface
       product: 82579LM Gigabit Network Connection
       vendor: Intel Corporation
       physical id: 19
       bus info: pci@0000:00:19.0
       logical name: eth0
       version: 04
       serial: 78:2b:cb:9e:38:fa
       size: 1Gbit/s
       capacity: 1Gbit/s
       width: 32 bits
       clock: 33MHz
       capabilities: pm msi bus_master cap_list ethernet physical tp 10bt 10bt-fd
100bt 100bt-fd 1000bt-fd autonegotiation
       configuration: autonegotiation=on broadcast=yes driver=e1000e
driverversion=2.1.4-k duplex=full firmware=0.13-4 ip=128.125.124.174 latency=0
```

```
link=yes multicast=yes port=twisted pair speed=1Gbit/s
       resources: irq:41 memory:e1c00000-e1c1ffff memory:e1c80000-e1c80fff
ioport:5080(size=32)
```

The external ethernet card is the 82571EB Gigabit Ethernet Controller which we could learn more about with *lspci -v*. It's clear from the "serial" lines that MAC addresses 00:15:17:dc:5d:9c and 00:15:17:dc:5d:9d correspond to these interfaces; we will use these for RTNet. The motherboard interface is 00:15:17:dc:5d:9c; we will use this for non-RT ethernet.

Note the MAC addresses of the ports you wish to make real-time compatible and reorder/name them as eth0, eth1... ethX starting with the lowest MAC address. Note the MAC address of the port you wish to use for non-RT ethernet and number it ethX+1 (the next highest number available). The new version of this file looks like:

/etc/udev/rules.d/70-persistent-net.rules

```
File Edit Options Buffers Tools Help

# This file was automatically generated by the /lib/udev/write_net_rules
# program, run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single
# line, and change only the value of the NAME= key.

# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0 (e1000e)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:15:17:dc:5d:9c",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="e\
th*", NAME="eth0"

# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:01.0/0000:01:00.1 (e1000e)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:15:17:dc:5d:9d",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="e\
th*", NAME="eth1"

# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:19.0 (e1000e)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="78:2b:cb:9e:38:fa",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="e\
th*", NAME="eth2"
```

In this example, we have named the non-RT port eth2; since we uninstalled the network manager (which is normally responsible for automatic network configuration) we need to manually configure the interfaces we wish to bring up at boot time (the non-RT port). This is done by editing the /etc/network/interfaces file as below.

/etc/network/interfaces

```
auto lo
iface lo inet loopback
auto eth2
iface eth2 inet dhcp
```

Now, reboot the machine and run *ifconfig* to monitor interface status. You should see that your non-realtime ethernet interface is working and you again have network connectivity!

Running RTNet

The script that we use to bring up the interface in this example is called load_rtnet and is called with

/etc/network/interfaces

```
source /usr/local/rtnet/load_rtnet <mode>
```

where <mode> can be start, stop, or restart. Create this script in /usr/local/rtnet and copy its content from below. Make sure it's marked executable before running it (use *chmod a+x load_rtnet* if necessary). Note that you'll need to edit this script a bit to make it work for your system. (old version of the script: athena_rtnet)

First, you'll need to know the MAC addresses of the ports you with to use for real-time ethernet for configuring *rteth0* and *rteth1*; we know this from the previous section. However, you'll still need to figure out the mapping between MAC addresses and physical port locations (they are not necessarily consecutive on the actual card) but this is easy to see later. You'll also need to change *eth2* to the port which is to be used for non-RT ethernet. The "cards" parameter in loading the rt_e1000 driver specifies which ports to use for RT-ethernet; these are not necessarily consecutive either.

For the Intel Corporation PRO/1000 PT Quad Port card, you'll need to make a few changes. First, change the "cards" parameter from 1,1 to 1,1,1,1 to ensure all the ports get turned into RT ports. Then, if you want to add two more ports (rteth2 and rteth3), you'd need to add lines in all the places where rteth0 and rteth1 are set up and taken down. The script below actually only makes use of rteth0 and rteth1 but you could always use more if necessary.

On hermes, we have an Intel Corporation PRO/1000 PT Dual Port Server Adapter ethernet card and use the following version of the script:

```
#! /bin/sh

RTNETPATH=/usr/local/rtnet
case "$1" in
start)
echo "Starting rtnet from $RTNETPATH"
ifconfig eth2 down
rmmod e1000e
sleep 1
insmod $RTNETPATH/modules/rtnet.ko
sleep 1
insmod $RTNETPATH/modules/rtipv4.ko
insmod $RTNETPATH/modules/rtcfg.ko
insmod $RTNETPATH/modules/rtpacket.ko
insmod $RTNETPATH/modules/rtudp.ko
insmod $RTNETPATH/modules/rt_e1000.ko cards=1,1
sleep 1
insmod /usr/local/rtnet/modules/rtcap.ko
echo "Setting up rtifconfig..."
$RTNETPATH/sbin/rtifconfig rteth0 up 192.168.1.100 netmask 255.255.255.0 hw ether
00:15:17:dc:5d:9c
$RTNETPATH/sbin/rtifconfig rteth1 up 192.168.1.100 netmask 255.255.255.0 hw ether
00:15:17:dc:5d:9d
$RTNETPATH/sbin/rtifconfig

echo "Setting up rtroute..."
# lower body

$RTNETPATH/sbin/rtroute add 192.168.1.22 0:0:0:12:34:6B dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.23 0:0:0:12:34:6C dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.24 0:0:0:12:34:6D dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.25 0:0:0:12:34:6E dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.26 0:0:0:12:34:6F dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.27 0:0:0:12:34:70 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.28 0:0:0:12:34:71 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.29 0:0:0:12:34:72 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.30 0:0:0:12:34:73 dev rteth0
```

```
$RTNETPATH/sbin/rtroute add 192.168.1.31 0:0:0:12:34:74 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.32 0:0:0:12:34:75 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.33 0:0:0:12:34:76 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.34 0:0:0:12:34:77 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.35 0:0:0:12:34:78 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.36 0:0:0:12:34:79 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.37 0:0:0:12:34:7A dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.38 0:0:0:12:34:7B dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.39 0:0:0:12:34:7C dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.40 0:0:0:12:34:7D dev rteth0
# right arm

$RTNETPATH/sbin/rtroute add 192.168.1.8 0:0:0:12:34:5d dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.9 0:0:0:12:34:5e dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.10 0:0:0:12:34:5f dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.11 0:0:0:12:34:60 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.12 0:0:0:12:34:61 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.13 0:0:0:12:34:62 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.14 0:0:0:12:34:63 dev rteth0
# left arm

$RTNETPATH/sbin/rtroute add 192.168.1.15 0:0:0:12:34:64 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.16 0:0:0:12:34:65 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.17 0:0:0:12:34:66 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.18 0:0:0:12:34:67 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.19 0:0:0:12:34:68 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.20 0:0:0:12:34:69 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.21 0:0:0:12:34:6a dev rteth0
# head

$RTNETPATH/sbin/rtroute add 192.168.1.1 0:0:0:12:34:56 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.2 0:0:0:12:34:57 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.3 0:0:0:12:34:58 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.4 0:0:0:12:34:59 dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.5 0:0:0:12:34:5A dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.6 0:0:0:12:34:5B dev rteth0
$RTNETPATH/sbin/rtroute add 192.168.1.7 0:0:0:12:34:5C dev rteth0

# hands
$RTNETPATH/rtroute add 192.168.3.8 0:0:0:32:34:5D dev rteth1
$RTNETPATH/rtroute add 192.168.3.9 0:0:0:32:34:5E dev rteth1
# ATI FT sensor
$RTNETPATH/rtroute add 192.168.4.1 00:16:BD:00:0C:2E dev rteth1

echo "Setting up multicast route..."
#changed to to map 239.0.0.2 to correct mac address (ludo)

$RTNETPATH/sbin/rtroute add 239.0.0.2 01:00:5e:00:00:02 dev rteth0
$RTNETPATH/sbin/rtroute

ifconfig rteth0 up 192.168.1.100 netmask 255.255.255.0 hw ether 00:15:17:dc:5d:9c
ifconfig rteth1 up 192.168.1.100 netmask 255.255.255.0 hw ether 00:15:17:dc:5d:9d
ifconfig
echo "Loading non-rt ethernet..."
modprobe e1000e
sleep 1
ifconfig eth2 up
;;
restart)
```

```
$RTNETPATH/load_rtnet stop
$RTNETPATH/load_rtnet start
;;
stop)
echo "Stopping rtnet from $RTNETPATH"
echo "Shutting down rtifconfig..."
$RTNETPATH/sbin/rtifconfig rteth0 down
$RTNETPATH/sbin/rtifconfig rteth1 down
rmmod rtcap.ko
rmmod rt_e1000.ko
sleep 1
rmmod rtpacket.ko
rmmod rtcfg.ko
rmmod rtudp.ko
rmmod rtipv4.ko
#rmmod rtloopback.ko

rmmod rtnet.ko

;;
*)
echo $"usage: $0 {start|stop|restart}"
exit 3
;;
```

```
esac
:
```

The script run with "start" works roughly as follows: first, all ethernet interfaces are brought down with *ifdown.* Next, the non-RT ethernet module (here e1000e) is unloaded and the RTnet modules are loaded. The "cards" parameter specifies which and how many ports will become real-time (the rest will be non-real time). The real-time ethernet ports are configured with *rtifconfig* and assigned IP addresses which are associated with the port MAC addresses. Routes are then added to the routing table with *routeadd.* rteth0 handles all routed other than the hands and the ATI force/torque sensor. Next, *ifconfig* is called with the same argument as *rtifconfig* so that we can access interface information for the RT ports via *ifconfig.* The non-RT ethernet is then set up by loading the non-RT driver (here e1000e), which automatically gets loaded by the remaining ethernet device. Running the script with "stop" basically deconfigures the RT ethernet interfaces in the opposite order. Running it with "restart" simply stops and then starts the RT ports.

Run the startup script (it may take a few seconds to finish) and then run *dmesg* to see which port numbers were assigned to which MAC addresses (this is done automatically on the hardware level by the device driver). You should see some output like this with *dmesg* after starting RTNet:

```
[  953.206279] *** RTnet 0.9.13 - built on Nov  5 2014 14:49:59 ***
[  953.206279]
[  953.206282] RTnet: initialising real-time networking
[  954.208995] RTcfg: init real-time configuration distribution protocol
[  954.212949] Intel(R) PRO/1000 Network Driver - version 7.1.9
[  954.212952] Copyright (c) 1999-2006 Intel Corporation.
[  954.260555] e1000: 0000:01:00.0: e1000_probe: (PCI Express:2.5Gb/s:Width x4)
00:15:17:dc:5d:9c
[  954.336507] RTnet: registered rteth0
[  954.336508] e1000: rteth0: e1000_probe: Intel(R) PRO/1000 Network Connection
[  954.384213] e1000: 0000:01:00.1: e1000_probe: (PCI Express:2.5Gb/s:Width x4)
00:15:17:dc:5d:9d
[  954.460152] RTnet: registered rteth1
[  954.460153] e1000: rteth1: e1000_probe: Intel(R) PRO/1000 Network Connection
[  955.459158] RTcap: real-time capturing interface
[  955.460574] rt_e1000 0000:01:00.0: irq 41 for MSI/MSI-X
[  958.476208] e1000e: Intel(R) PRO/1000 Network Driver - 2.1.4-k
[  958.476210] e1000e: Copyright(c) 1999 - 2012 Intel Corporation.
[  958.476239] e1000e 0000:00:19.0: setting latency timer to 64
[  958.476300] e1000e 0000:00:19.0: Interrupt Throttling Rate (ints/sec) set to
dynamic conservative mode
[  958.476334] e1000e 0000:00:19.0: irq 43 for MSI/MSI-X
[  958.679851] e1000e 0000:00:19.0 eth0: (PCI Express:2.5GT/s:Width x1)
78:2b:cb:9e:38:fa
[  958.679854] e1000e 0000:00:19.0 eth0: Intel(R) PRO/1000 Network Connection
[  958.679888] e1000e 0000:00:19.0 eth0: MAC: 10, PHY: 11, PBA No: E041FF-0FF
[  958.829211] e1000e 0000:00:19.0: irq 43 for MSI/MSI-X
[  958.929060] e1000e 0000:00:19.0: irq 43 for MSI/MSI-X
[  958.929230] IPv6: ADDRCONF(NETDEV_UP): eth2: link is not ready
[  961.879152] e1000e: eth2 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: Rx/Tx
[  961.879186] IPv6: ADDRCONF(NETDEV_CHANGE): eth2: link becomes ready
```

Very importantly, check that the MAC addresses in the lines preceding "RTnet: registered rteth0" and "RTnet: registered rteth1" match those used for *rtifconfig* (and later *ifconfig*) in the startup script. If they do not, change them to match (otherwise the association between port name and MAC address will be wrong on the IP layer level and none of your messages will actually get sent). While we assume this in our script above, RTNet doesn't necessarily choose rteth0 and rteth1 to use those interfaces with the two lowest/consecutive MAC adresses!

Note that first the RT ethernet ports are configured, then the non-RT port (eth2 here). Note that it may take a little time for the non-RT port to complete DHCP and regain connectivity.

We can find out the MAC address-portname association very easily. When you plug the cable into one of the RT ports, you should see something like "rt_e1000: rteth0 NIC Link is Up 100 Mbps Full Duplex" in the kernel log. Move the cable around all the port

In our case, we use the Intel Pro/1000 PT Dual Port ethernet card which has (you guessed it) two ports. The default driver for use with this Intel card is e1000e and the RTNet driver we use is rt_e1000.ko; there is also an rt_e1000e.ko which can be used with newer Intel cards if you run into problems (though note that the "cards" parameter is obsolete for this newer driver so you need to figure out a way to specify which ports should become real-time).

Running RTNet at startup (optional)

In order to run this script at startup, we need to copy it (making sure we have already marked it with executable bits) to the directory /etc/init.d which is where all startup scripts reside. In order to ensure that the script actually gets run, though, we need to go create symbolic links to it in each of the /etc/rcN.d directories (where rc stands for run control and N is from 0-6 plus S). Briefly, the numbers refer to the different runlevels the system can be initialized into.

When a linux machine is started, the kernel loads the root filesystem and reads a configuration file (traditionally /etc/inittab, though Ubuntu uses /etc/init/rc-sysinit.conf) to know which runlevel it should boot into using the *init* command. All scripts linked to in the corresponding /etc/rcN.d directory are then run. Different linux distributions define these runlevels differently, but most reserve level 0 for shutdown and 6 for reboot and 1 is not meant to be used; the user generally doesn't touch anything other than levels 2-5, with 5 traditionally being the GUI interface (in Ubuntu, the default is 2 but 2-5 are all GUI). There is also a special runlevel called S which specifies scripts which get run before the default runlevel is initialized - this is similar to 1 and should only be used if you need to do something before all the other runlevels.

We set up runlevels for the RTNet script by creating the symbolic links in the rcN.d directories shown below, either manually or using the *update-rc.d* command:

```
cp /usr/local/rtnet/load_rtnet /etc/init.d
cd /etc/rc0.d
ln -s ../init.d/load_rtnet K20load_rtnet && cd ../rc1.d
ln -s ../init.d/load_rtnet K20load_rtnet && cd ../rc2.d
ln -s ../init.d/load_rtnet S20load_rtnet && cd ../rc3.d
ln -s ../init.d/load_rtnet S20load_rtnet && cd ../rc4.d
ln -s ../init.d/load_rtnet S20load_rtnet && cd ../rc5.d
ln -s ../init.d/load_rtnet S20load_rtnet && cd ../rc6.d
ln -s ../init.d/load_rtnet K20load_rtnet && cd ../rcS.d
ln -s ../init.d/load_rtnet K20load_rtnet
```

The naming of symbolic links works as follows:

```
[K | S] + nn + [string]
```

Here, K denotes that the service should be killed and S denotes that it should be started. The number nn is a two-digit number from 01-99 which denotes the priority of the service with 01 being the highest priority. This can be used to ensure that services which depend on others run later, though for us here it is irrelevant. Finally, string denotes the name of the script placed in /etc/init.d. We see above that the service is killed in levels 0, 1, 6 and S and started in all other levels. Note that "start" is automatically passed as an argument to the script when the link name starts with S and "stop" is passed when the link name starts with K. For more information on runlevels in linux, check http://www.tldp.org/HOWTO/HighQuality-Apps-HOWTO/boot.html. For info on what happens in Ubuntu specifically, check https://help.ubuntu.com/community/UpstartHowto.

Testing RTNet (on the SARCOS robot)

Once you have installed the hermes/athena code, you can test RTNet by running a test communication loop with the SARCOS robot. This program, called test_communication_loop, is installed in cb_communication and should be run as root. Run the program, select the default frequency and specify whether you're communicating with the lower or upper body via the config file parameter. If you see no error messages, hit ENTER and take a look at the newly-created file log_file.dat. There, you should see ones (1's) in the rightmost columns, indicating a successful message passing between computer and robot GDC cards. If you have any issues, first open wireshark to monitor ethernet traffic and then run the program again to diagnose where the problem lies.

Installing usb4rt

Background

In order to use real-time USB support, you must have a USB 1.1 compatible card ie one which is UHCI-based since usb4rt only provides a UHCI driver at this time. Any PCI-based USB card made by Intel/VIA will work. Of course, you need PCI space for both the real-time ethernet card and for the USB card and potentially also an external graphics card (3 PCI slots total). Real-time USB 2.0 support is not yet available in Xenomai.

At this time, the real-time USB 1.1 drivers have been tested only with the Microstrain 3dm-gx3-25 and 3dm-gx3-45 IMUs; in theory, any other USB 1.1-compatible device can be used as well. Here we focus on using these IMUs.

A few notes on linux USB (you may choose to skip past this)
The USB protocol is a master/slave system controlled by a single host. This means USB devices cannot directly talk to one another which alleviates issues of collision avoidance, etc. USB 1.0 was released in 1996, developed by a number of companies. USB 1.1 was released in 1998 and allows for a max bandwidth of 12Mbits/s in "Full Speed" mode but in reality is about 8.5Mbits/s under ideal conditions due to overhead (and may be even slower). There is a "Low Speed" mode limited to 1.5Mbit/s. USB 2.0 was released in 2000 and featured a "Hi Speed" mode operating at 480Mbit/s (in reality, 280Mbit/s or 35Mbyte/s due to overhead) and a legacy "USB 1.1 Full Speed" mode. USB 3.0 was released in 2008 and added a "Super Speed" mode for 5Gbit/s (realistically, 4Gbit/s or 500Mbyte/s).

The motherboard provides a USB host controller to act as master using a Host Controller Driver (HCD) to interface with the hardware-level Host Controller Interface (HCI). For USB 1.1, there were two HCIs in use, depending on hardware: Compaq's Open Host Controller Interface (OHCI) and Intel's proprietary Universal Host Controller Interface (UHCI). UHCI requires simpler hardware and thus relies on the controller more heavily, increasing CPU load slightly. Intel/VIA used UHCI while most other companies used OHCI. For USB 2.0, a single HCD called Enhanced Host Controller Interface (EHCI) was developed. On early setups, EHCI was used for "Hi Speed" mode and either OHCI or UHCI was used for legacy modes. On modern setups, EHCI implements OHCI/UHCI virtually to indirectly provide legacy support (this is possible because all USB ports are routed through a Rate Matching Hub or RMH). USB 3.0 uses the Extensible Host Controller Interface (XHCI) and supports all the above modes. The HCD you use depends on the HCI your controller hardware uses; use *lspci -v | grep HCI* to list controllers and their HCIs and choose your HCD accordingly. On newer kernels, there exist *ehci-hcd*, *uhci-hcd* and *ohci-hcd*; you can check if any of these modules (drivers) are loaded with *lsmod | grep hcd* and load/unload them with *insmod/rmmod* or using *modprobe.* Note that all modules are actually installed in /lib/modules/<kernel-name> and modules can be loaded at boot time by adding them to /etc/modules. You can get information about a module (including its location) using *modinfo.*

Physically, a computer has a small number of USB ports; however, the standard supports up to 127 devices, so hubs are used to extend the number of ports. In theory, self-powered USB devices can pull 500mA per device. When a USB device is attached, it is assigned a unique device number (1-127) and its device descriptor (which contains device information) is read. Devices are assigned classes, one of which is Human Interface Device (HID) for peripherals such as a keyboard, mouse, etc. To get detailed device information, use *lsusb -v* (for information about a single device, use *lsusb -D /proc/usb/bus/001/005* for example which gives info about device 5 on bus 1). You can easily see assigned device numbers in the kernel log with *dmesg | grep USB*.

The following is a very good source on low-level linux USB information (along with lots of other linux info): http://www.makelinux.net/ldd3/chp-13

Installation

First, install your USB 1.1-compatible card. You can then install the usb4rt (real-time USB modules) with

```
cd /usr/src/

# clone the repository
git clone git://git.kiszka.org/usb4rt.git

# stay on master branch
cd usb4rt

# configure the driver (the last option points to the xenomai install directory; it
would be /usr/xenomai by default but since we used debian packages this isn't the
right place to look)
./configure --enable-drv-cdc-acm --enable-bandw-reclam --with-xeno-user-dir=/usr/

# build and install
make && make install
```

If you run into usb4rt issues later and need more info for debugging, add --enable-dbg-common to the above configuration command and recompile.

Configuration

Next, we need to unbind the UHCI driver from the 1.1-compatible card and load the real-time drivers in their stead. Run *dmesg* | *grep uhci* and you should see output which looks like:

```
[    2.101228] uhci_hcd: USB Universal Host Controller Interface driver
[    2.101248] uhci_hcd 0000:05:00.0: setting latency timer to 64
[    2.101252] uhci_hcd 0000:05:00.0: UHCI Host Controller
[    2.101255] uhci_hcd 0000:05:00.0: new USB bus registered, assigned bus number 1
[    2.101288] uhci_hcd 0000:05:00.0: irq 16, io base 0x00004020
[    2.101318] usb usb1: Manufacturer: Linux 3.8.13-ipipe-xenomai-2.6.3 uhci_hcd
[    2.101421] uhci_hcd 0000:05:00.1: setting latency timer to 64
[    2.101425] uhci_hcd 0000:05:00.1: UHCI Host Controller
[    2.101428] uhci_hcd 0000:05:00.1: new USB bus registered, assigned bus number 2
[    2.101461] uhci_hcd 0000:05:00.1: irq 17, io base 0x00004000
[    2.101492] usb usb2: Manufacturer: Linux 3.8.13-ipipe-xenomai-2.6.3 uhci_hcd
```

This tells you which USB hubs and PCI addresses your USB 1.1-compatible card is attached to and the IRQ lines it's using with the non-RT uhci_hcd driver. The reason there are two UHCI Host Controllers is because the USB 1.1 ports on the card got split over two hubs/PCI addresses/IRQs here. To figure out which ports correspond to which hub, plug in a device and look for a line in *dmesg* such as:

```
[ 6689.424925] usb 2-1: new full-speed USB device number 9 using uhci_hcd
```

The "usb 2-1" means that the device was attached to hub 2 at port 1. If you move the device around the USB ports on the card, you'll see this message change - this is a simple way to tell which USB ports get mapped to which hubs.

To confirm this IRQ assignment, run *cat proc/interrupts* which should produce output like:

```
root@perseus:~# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6
CPU7
  0:         15          0          0          0          0          0          0
0   IO-APIC-edge      timer
  1:          3          0          0          0          0          0          0
0   IO-APIC-edge      i8042
  4:         16          0          0          0          0          0          0
0   IO-APIC-edge
  8:          1          0          0          0          0          0          0
0   IO-APIC-edge      rtc0
  9:          0          0          0          0          0          0          0
0   IO-APIC-fasteoi   acpi
 12:          4          0          0          0          0          0          0
0   IO-APIC-edge      i8042
 16:        546          0          0         91          0          0          0
0   IO-APIC-fasteoi   uhci_hcd:usb1, ehci_hcd:usb3, nouveau
 17:        932          0      69308          0          0          0          0
0   IO-APIC-fasteoi   uhci_hcd:usb2, ehci_hcd:usb4, snd_hda_intel
 41:        274          0          0          0      97294          0          0
0   PCI-MSI-edge      eth4
 42:      12461      30010          0          0          0        123          0
0   PCI-MSI-edge      ahci
 43:         22          0          0          0          0          0          0
0   PCI-MSI-edge      mei
 44:        256          0          0          0          0          0          0
0   PCI-MSI-edge      snd_hda_intel
NMI:        411        279        327       1257        109        138        128
259   Non-maskable interrupts
LOC:     914002     978334    1362324    1418384     244281     333905     244260
281251   Local timer interrupts
SPU:          0          0          0          0          0          0          0
0   Spurious interrupts
PMI:        411        279        327       1257        109        138        128
259   Performance monitoring interrupts
IWI:          0          0          0          0          0          0          0
0   IRQ work interrupts
RTR:          1          0          0          0          0          0          0
0   APIC ICR read retries
RES:      84444      78911      84011      82623      28408      25959      31037
27421   Rescheduling interrupts
CAL:        764        785        676        873        860        884        908
897   Function call interrupts
TLB:       2999       3590       2938       3112       3488       3581       3598
3535   TLB shootdowns
TRM:          0          0          0          0          0          0          0
0   Thermal event interrupts
THR:          0          0          0          0          0          0          0
0   Threshold APIC interrupts
MCE:          0          0          0          0          0          0          0
0   Machine check exceptions
MCP:         23         23         23         23         23         23         23
23   Machine check polls
ERR:          0
MIS:          0
```

The modules which use each interrupt are listed at the end of each line. Note that in this example we have uhci_hcd listed twice: once for

interrupt 16 and once for 17. The "usb1" and "usb2" indicate the USB hub to which each is associated. Also note that we have a USB 2.0 module (ehci_hcd) on hub 3 as well as the open-source nouveau graphics driver sharing IRQ 16 while we have a USB 2.0 module (ehci_hcd) on hub 4 as well as a sound driver sharing IRQ 17.

Anyway, the point is that we need to unbind uhci_hcd from either hub 1 or 2, clear everything else on that interrupt line (to prevent conflicts) and then load the usb4rt drivers. Ideally, your machine allows you to assign IRQs to hardware in the BIOS, in which case you can assign the USB 1.1 card its own IRQ to prevent conflicts. Most machines don't allow you to do this, though - you're stuck with the above IRQ assignments. This means we have to carefully choose which IRQ to use. In the output of *cat /proc/interrupts*, we see that we can use 16 or 17; on 16, we'd have to unbind the hub3 EHCI controller and the graphics driver while on 17 we'd have to lose the hub4 EHCI controller and sound driver. If you're very lucky, you'll see nothing other than UHCI controllers on these IRQ lines; in this case you don't have to worry about IRQ sharing and you can skip the following section.

Resolving IRQ sharing

Based on the output of *cat /proc/interrupts* we see that our UHCI controllers are sharing IRQ lines 16 and 17 with the video and sound drivers, respectively. We now have two choices:

1. Unbind the ehci_hcd driver from hub3 and add "nomodeset" to the kernel boot parameters (in /etc/default/grub after "quiet splash") so that the onboard graphics chip gets used in place of the nouveau graphics driver. Run *update-grub* and reboot.
2. Move the mouse, keyboard, etc to ports connected to hub3 and unbind ehci_hcd from hub4 instead. Also unbind the sound driver, disabling sound output.

In our case we went with option (1) because we needed to use the onboard graphics chip and VESA driver anyway (in order to open up a PCI slot for the USB 1.1 card and resolve latency issues).

Now that we have an IRQ line shared only by EHCI and UHCI controllers, we finally want to unbind the EHCI controller to free up the line. To do this, we need to make sure none of our USB devices (mouse, keyboard, etc) are attached to the hub associated with this EHCI controller. Running *lsusb* produces the following output:

```
Bus 002 Device 002: ID 1a40:0101 Terminus Technology Inc. 4-Port HUB
Bus 003 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 004 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 003: ID 0424:2502 Standard Microsystems Corp.
Bus 004 Device 004: ID 0424:2602 Standard Microsystems Corp. USB 2.0 Hub
Bus 004 Device 005: ID 0424:2228 Standard Microsystems Corp. 9-in-2 Card Reader
Bus 004 Device 012: ID 046d:c05a Logitech, Inc. Optical Mouse M90
Bus 004 Device 013: ID 046d:c31d Logitech, Inc.
```

We see that there are four root hubs, each attached to a different hub. A root hub is always the first device on its hub and has the vendor code "1d6b" (Linux Foundation). Devices attached to a particular root hub have device numbers higher than 001. We see above that the Logitech mouse and keyboard devices are devices 012 and 013 on Bus 004, which corresponds to the USB 2.0-compatible root hub called hub4 (you could also run *dmesg | grep Mouse* and *dmesg | grep Keyboard* which should tell you the same thing). We're now sure that our mouse and keyboard are attached to hub 4. Since we're going to unbind the driver for hub3, this is fine (otherwise, we'd first have to move the USB devices to different ports).

To unbind a driver from a PCI port, we use *echo -n "<PCI ADDRESS>" > /path/to/driver.* From *dmesg | grep ehci* and *dmesg | grep uhci* we find out that the PCI address of the hub3 USB 2.0 port is 0000:00:1a.0 and the address of the hub1 USB 1.1 port is 0000:05:00.0 - we need to unbind them both and then load the real-time USB 1.1 driver in their stead.

Running usb4rt

The following bash script "load_rtusb" is used to set up real-time USB - it unloads the non-RT drivers and loads the usb4rt modules. Once you chmod it to be executable, usage is *source load_rtusb <stop/start/restart>*. You will need to change the PCI addresses in the "unbind" lines to clear the appropriate IRQ as discussed above. Plug in the device and run the script.

```
#! /bin/sh

RTUSBPATH=/usr/local/usb4rt/
case "$1" in
start)
echo "Starting usb4rt from $RTUSBPATH"
echo -n "0000:05:00.0" > /sys/bus/pci/drivers/uhci_hcd/unbind
echo -n "0000:00:1a.0" > /sys/bus/pci/drivers/ehci-pci/unbind
sleep 1
insmod /usr/local/usb4rt/modules/rt_usbcore.ko
sleep 1
insmod $RTUSBPATH/modules/rt_uhci_hcd.ko
insmod $RTUSBPATH/modules/rt_cdc_acm.ko vendor=0x199b product=0x3065 start_index=0
# For the 3DM-GX3-45 model, comment the above line and uncomment the following line.
#insmod $RTUSBPATH/modules/rt_cdc_acm.ko vendor=0x199b product=0x3a65 start_index=0
;;
restart)
$RTUSBPATH/load_rtusb stop
$RTUSBPATH/load_rtusb start
;;
stop)
echo "Stopping usb4rt from $RTUSBPATH"
rmmod rt_cdc_acm
rmmod rt_uhci_hcd
rmmod rt_usbcore
sleep 1
echo -n "0000:05:00.0" > /sys/bus/pci/drivers/uhci_hcd/bind
echo -n "0000:00:1a.0" > /sys/bus/pci/drivers/ehci-pci/bind
;;
*)
echo $"usage: $0 {start|stop|restart}"
exit 3
;;
esac
:
```

You should see the following ouput in the system log (dmesg) after running the above start script:

```
[15247.217455] uhci_hcd 0000:05:00.0: remove, state 1
[15247.217462] usb usb1: USB disconnect, device number 1
[15247.217609] uhci_hcd 0000:05:00.0: USB bus 1 deregistered
[15247.217647] ehci-pci 0000:00:1a.0: remove, state 1
[15247.217652] usb usb3: USB disconnect, device number 1
[15247.217653] usb 3-1: USB disconnect, device number 2
[15247.224546] ehci-pci 0000:00:1a.0: USB bus 3 deregistered
[15248.223476] ********** Realtime USB-Core Module 0.0.5 ***********
[15248.223478] RT-USBCORE: Max 16 Controller
[15248.223479] RT-USBCORE: Max 128 USB-Devices
[15248.223480] RT-USBCORE: Common debugging: disabled
[15248.223481] RT-USBCORE: Queue head debugging: disabled
[15248.223482] RT-USBCORE: Transfer descriptor debugging: disabled
[15248.223482] RT-USBCORE: Time debugging: disabled
[15248.223487] RT-USBCORE: Control-URB @ 0xffff8800867afc00 with  8 Byte Packet-Size
[15248.223488] RT-USBCORE: Control-URB @ 0xffff8800867ac000 with 16 Byte Packet-Size
[15248.223489] RT-USBCORE: Control-URB @ 0xffff8800867af800 with 32 Byte Packet-Size
[15248.223490] RT-USBCORE: Control-URB @ 0xffff8800867af200 with 64 Byte Packet-Size
```

```
[15248.223491] RT-USBCORE: Initialize Hub-List
[15248.223492] RT-USBCORE: Initialize Controller-List
[15248.223492] RT-USBCORE: Loading Completed (1152 Byte allocated)
[15249.222373] ********* Realtime Driver for Universal Host Controller 0.0.5
**********
[15249.222377] RT-UHC-Driver: Searching for Universal-Host-Controller
[15249.222384] USB Universal Host Controller found : Vendor = 0x1106, Device = 0x3038,
IRQ = 16, IO-Port = 0x00002020 (32 Bytes)
[15249.222385] RT-UHC-Driver: Request IO-Port @ 0x00002020 (32 Byte) for UHC[0] ...
[OK]
[15249.222389] RT-UHC-Driver: Request RTDM IRQ 16 ... [OK]
[15249.222450] RT-USBCORE: Register Host-Controller Driver
[15249.222451] RT-USBCORE: Host-Controller added to USB-Controller-List
[15249.222477] USB Universal Host Controller found : Vendor = 0x1106, Device = 0x3038,
IRQ = 17, IO-Port = 0x00002000 (32 Bytes)
[15249.222478] RT-UHC-Driver: Request IO-Port @ 0x00002000 (32 Byte) for UHC[1] ...
[BUSY]
[15249.222480] RT-UHC-Driver: Loading Completed (48 Byte allocated)
[15265.615417] usb 2-2.3: USB disconnect, device number 12
[15272.630661] RT-USBCORE: Unregister Host-Controller Driver 0
[15272.683142] RT-UHC-Driver: Delete  RTDM IRQ 16
[15272.683149] RT-UHC-Driver: Release IO-Port 0x00002020 (32 Byte)
[15272.683153] RT-UHC-Driver : Unloading complete (0 byte allocated)
[15272.683880] RT-USBCORE: Unloading Completed (0 Byte allocated)
[15273.681754] uhci_hcd 0000:05:00.0: UHCI Host Controller
[15273.681760] uhci_hcd 0000:05:00.0: new USB bus registered, assigned bus number 1
[15273.681807] uhci_hcd 0000:05:00.0: irq 16, io base 0x00002020
[15273.681840] usb usb1: New USB device found, idVendor=1d6b, idProduct=0001
[15273.681842] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[15273.681843] usb usb1: Product: UHCI Host Controller
[15273.681845] usb usb1: Manufacturer: Linux 3.8.13-ipipe-xenomai-2.6.3 uhci_hcd
[15273.681846] usb usb1: SerialNumber: 0000:05:00.0
[15273.681932] hub 1-0:1.0: USB hub found
[15273.681936] hub 1-0:1.0: 2 ports detected
[15273.684378] ehci-pci 0000:00:1a.0: setting latency timer to 64
[15273.684382] ehci-pci 0000:00:1a.0: EHCI Host Controller
[15273.684385] ehci-pci 0000:00:1a.0: new USB bus registered, assigned bus number 3
[15273.684397] ehci-pci 0000:00:1a.0: debug port 2
[15273.688269] ehci-pci 0000:00:1a.0: cache line size of 64 is not supported
[15273.688277] ehci-pci 0000:00:1a.0: irq 16, io mem 0xdcc70000
[15273.700211] ehci-pci 0000:00:1a.0: USB 2.0 started, EHCI 1.00
[15273.700230] usb usb3: New USB device found, idVendor=1d6b, idProduct=0002
[15273.700234] usb usb3: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[15273.700237] usb usb3: Product: EHCI Host Controller
[15273.700240] usb usb3: Manufacturer: Linux 3.8.13-ipipe-xenomai-2.6.3 ehci_hcd
[15273.700242] usb usb3: SerialNumber: 0000:00:1a.0
[15273.700325] hub 3-0:1.0: USB hub found
[15273.700327] hub 3-0:1.0: 3 ports detected
[15273.704997] uhci_hcd 0000:05:00.0: remove, state 1
[15273.705001] usb usb1: USB disconnect, device number 1
[15273.705119] uhci_hcd 0000:05:00.0: USB bus 1 deregistered
[15273.705147] ehci-pci 0000:00:1a.0: remove, state 1
[15273.705149] usb usb3: USB disconnect, device number 1
[15273.709151] ehci-pci 0000:00:1a.0: USB bus 3 deregistered
[15274.708025] ********** Realtime USB-Core Module 0.0.5 **********
[15274.708028] RT-USBCORE: Max 16 Controller
[15274.708029] RT-USBCORE: Max 128 USB-Devices
[15274.708029] RT-USBCORE: Common debugging: disabled
[15274.708030] RT-USBCORE: Queue head debugging: disabled
```

```
[15274.708031] RT-USBCORE: Transfer descriptor debugging: disabled
[15274.708032] RT-USBCORE: Time debugging: disabled
[15274.708037] RT-USBCORE: Control-URB @ 0xffff8800c8ea6400 with  8 Byte Packet-Size
[15274.708038] RT-USBCORE: Control-URB @ 0xffff8800c8ea5c00 with 16 Byte Packet-Size
[15274.708039] RT-USBCORE: Control-URB @ 0xffff8800c8ea4800 with 32 Byte Packet-Size
[15274.708039] RT-USBCORE: Control-URB @ 0xffff8800c8ea5a00 with 64 Byte Packet-Size
[15274.708040] RT-USBCORE: Initialize Hub-List
[15274.708041] RT-USBCORE: Initialize Controller-List
[15274.708042] RT-USBCORE: Loading Completed (1152 Byte allocated)
[15275.706935] ********* Realtime Driver for Universal Host Controller 0.0.5
**********
[15275.706939] RT-UHC-Driver: Searching for Universal-Host-Controller
[15275.706943] USB Universal Host Controller found : Vendor = 0x1106, Device = 0x3038,
IRQ = 16, IO-Port = 0x00002020 (32 Bytes)
[15275.706944] RT-UHC-Driver: Request IO-Port @ 0x00002020 (32 Byte) for UHC[0] ...
[OK]
[15275.706947] RT-UHC-Driver: Request RTDM IRQ 16 ... [OK]
[15275.707004] RT-USBCORE: Register Host-Controller Driver
[15275.707005] RT-USBCORE: Host-Controller added to USB-Controller-List
[15276.152282] RT-USBCORE: Registering CTRL-URB (8 Byte) @ Host-Controller 0
[15276.153471] 000:00-000: Setting Address 1
[15276.153537] 000:00-001: Switch to URB with 64 Byte Packet-Size
[15276.184155] 000:00-001: Manufacturer : MicroStrain, Inc.
[15276.184394] 000:00-001: Product : 3DM-GX3-25 Orientation Sensor
[15276.184839] 000:00-001: Serial : 6223.03946______
[15276.184840] 000:00-001: Device 1 configured
[15276.184841] --- DUMP USB-DEVICE @ ffffffffa057b3b8 -------------------------
[15276.184842]  Number          : 1
[15276.184843]  Vendor          : 0x199b
[15276.184844]  Product         : 0x3065
[15276.184845]  Root-Hub-Port   : 0x00
[15276.184846]  Class           : 0x02
[15276.184846]  Subclass        : 0x00
[15276.184847]  Protocol        : 0x00
[15276.184848]  In use          : no
[15276.184849]  Status          : 0x00
[15276.184850]  Speed           : 0x02
[15276.184850]  Host-Controller @ 0xffff880093eaec00
[15276.184851]
[15276.184852]  Endpoint         ---0 ---1 ---2 ---3 ---4 ---5 ---6 ---7 ---8 ---9
--10 --11 --12 --13 --14 --15
[15276.184854]  Ctrl-Mask IN        X    -    -    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184856]  Ctrl-Mask OUT       X    -    -    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184857]  Bulk-Mask IN        -    X    -    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184859]  Bulk-Mask OUT       -    -    -    X    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184861]  Int-Mask IN         -    -    X    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184862]  Int-Mask OUT        -    -    -    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184864]  Iso-Mask IN         -    -    -    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184866]  Iso-Mask OUT        -    -    -    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184868]  Toggle-Mask IN      -    -    -    -    -    -    -    -    -    -
-    -    -    -    -    -
```

```
[15276.184870]  Toggle-Mask OUT     -    -    -    -    -    -    -    -    -    -
-    -    -    -    -    -
[15276.184871]  Max-Packet IN    0064 0064 0008 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
[15276.184873]  Max-Packet OUT   0064 0000 0000 0064 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
[15276.184874] --- END USB-DEVICE --------------------------
[15276.184875] --- LIST USB-DEVICES -------------------------
[15276.184876]  No HCD VENDOR   PROD  CLS SCLS PROT ->CTRL CTRL-> ->BULK BULK->  ->INT
INT-> ->ISOC ISOC-> STATE SPEED USED
[15276.184878] 001 000 0x199b 0x3065 0x02 0x00 0x00 0x0001 0x0001 0x0002 0x0008 0x0004
0x0000 0x0000 0x0000 00000 00002    -
[15276.184880] --- END USB-DEVICES --------------------------
[15276.184895] USB Universal Host Controller found : Vendor = 0x1106, Device = 0x3038,
IRQ = 17, IO-Port = 0x00002000 (32 Bytes)
[15276.184897] RT-UHC-Driver: Request IO-Port @ 0x00002000 (32 Byte) for UHC[1] ...
[BUSY]
[15276.184900] RT-UHC-Driver: Loading Completed (48 Byte allocated)
[15276.185863] --- LIST USB-DEVICES -------------------------
[15276.185865]  No HCD VENDOR   PROD  CLS SCLS PROT ->CTRL CTRL-> ->BULK BULK->  ->INT
INT-> ->ISOC ISOC-> STATE SPEED USED
[15276.185867] 001 000 0x199b 0x3065 0x02 0x00 0x00 0x0001 0x0001 0x0002 0x0008 0x0004
0x0000 0x0000 0x0000 00000 00002    X
```

```
[15276.185868] --- END USB-DEVICES -------------------------
[15276.185869] rt_cdc_acm[0]: device found, vendor=0x199b, product=0x3065
```

If you see a line such as the one below at the end of your system log at this point, you likely have an IRQ sharing issue.

[602.383262] Xenomai: xnintr_irq_handler: IRQ16 not handled. Disabling IRQ line.

In this case, refer to previous sections and double-check your setup. Some more information on IRQ sharing and potential ways to solve it can be found here: http://xenomai.org/2014/06/what-if-xenomai-and-linux-devices-share-the-same-irq/

You should also find the device rtser0 now exists in the folder /proc/xenomai/rtdm:

```
root@perseus:~# cat /proc/xenomai/rtdm/rtser0/information
driver:        rt_cdc_acm
version:    0.5.0
peripheral:    USB CDC ACM
provider:    USB4RT
class:        2
sub-class:    -1
flags:        EXCLUSIVE  NAMED_DEVICE
lock count:    0
```

Now, double check all IRQs and the non-RT devices they are assigned to with *cat /proc/interrupts.* Double check the real-time assignments using *cat /proc/xenomai/irq* and compare the IRQ used by rt_uhci with the output of the previous file. Below we see that there are no non-RT devices using line 16 and that rt_uhci is now using line 16 for usb4rt as desired!

```
root@perseus:~# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6
CPU7
  0:         15          0          0          0          0          0          0
0   IO-APIC-edge      timer
  1:          3          0          0          0          0          0          0
0   IO-APIC-edge      i8042
  4:         16          0          0          0          0          0          0
0   IO-APIC-edge
  8:          1          0          0          0          0          0          0
0   IO-APIC-edge      rtc0
  9:          0          0          0          0          0          0          0
0   IO-APIC-fasteoi   acpi
 12:          4          0          0          0          0          0          0
0   IO-APIC-edge      i8042
 16:        546          0          0         91          0          0          0
0   IO-APIC-fasteoi   ehci_hcd:usb3, nouveau
 17:        932          0      69308          0          0          0          0
0   IO-APIC-fasteoi   uhci_hcd:usb2, ehci_hcd:usb4, snd_hda_intel
 41:        274          0          0          0      97294          0          0
0   PCI-MSI-edge      eth4
 42:      12461      30010          0          0          0        123          0
0   PCI-MSI-edge      ahci
 43:         22          0          0          0          0          0          0
0   PCI-MSI-edge      mei
 44:        256          0          0          0          0          0          0
0   PCI-MSI-edge      snd_hda_intel
NMI:        411        279        327       1257        109        138        128
```

```
259   Non-maskable interrupts
LOC:     914002     978334    1362324    1418384     244281     333905     244260
281251   Local timer interrupts
SPU:          0          0          0          0          0          0          0
0   Spurious interrupts
PMI:        411        279        327       1257        109        138        128
259   Performance monitoring interrupts
IWI:          0          0          0          0          0          0          0
0   IRQ work interrupts
RTR:          1          0          0          0          0          0          0
0   APIC ICR read retries
RES:      84444      78911      84011      82623      28408      25959      31037
27421   Rescheduling interrupts
CAL:        764        785        676        873        860        884        908
897   Function call interrupts
TLB:       2999       3590       2938       3112       3488       3581       3598
3535   TLB shootdowns
TRM:          0          0          0          0          0          0          0
0   Thermal event interrupts
THR:          0          0          0          0          0          0          0
0  A few notes on linux USB (you may choose to skip past this): Threshold APIC
interrupts
MCE:          0          0          0          0          0          0          0
0   Machine check exceptions
MCP:         23         23         23         23         23         23         23
23   Machine check polls
ERR:          0
MIS:          0


root@perseus:~# cat /proc/xenomai/irq
IRQ         CPU0        CPU1        CPU2        CPU3        CPU4        CPU5
CPU6        CPU7
 16:        1000           0           0           0           0           0
0           0         rt_uhci
16640:     1007875     1032112     1434126     1527423      266541      420504
264432      313195         [timer]
16641:           2           1           1           1           1           1
1           3         [reschedule]
16642:           0           1           1           1           1           1
1           1         [timer-ipi]
16643:           0           0           0           0           0           0
0           0         [sync]
16707:           8           0           1           0           0           0
0           0         [virtual]
```

Running usbrt at startup (optional)

Again, as we did for the RTNet script, we copy the usb4rt script to /etc/init.d/ and make links in /etc/rcX.d as follows:

```
cp /usr/local/usb4rt/load_rtusb /etc/init.d
cd /etc/rc0.d
ln -s ../init.d/load_rtusb K20load_rtusb && cd ../rc1.d
ln -s ../init.d/load_rtusb K20load_rtusb && cd ../rc2.d
ln -s ../init.d/load_rtusb S20load_rtusb && cd ../rc3.d
ln -s ../init.d/load_rtusb S20load_rtusb && cd ../rc4.d
ln -s ../init.d/load_rtusb S20load_rtusb && cd ../rc5.d
ln -s ../init.d/load_rtusb S20load_rtusb && cd ../rc6.d
ln -s ../init.d/load_rtusb K20load_rtusb && cd ../rcS.d
ln -s ../init.d/load_rtusb K20load_rtusb
```

Using usb4rt with more than one device

Using usb4rt with more than one device is simple: just change the line:

```
insmod $RTUSBPATH/modules/rt_cdc_acm.ko vendor=0x199b product=0x3065 start_index=0
```

to the following:

```
insmod $RTUSBPATH/modules/rt_cdc_acm.ko vendor=0x199b,0x199b product=0x3065,0x3065
start_index=0
```

where we have told the driver that two devices will be used. According to the driver, 16 devices can be added in this way.

Testing usb4rt

If everything worked, then your IMU light should be slowly blinking (indicating idle mode). Finally, run the test script in /usr/src/usb4rt/apps/test-3dm-gx3-25 with *test-3dm-gx3-25 rtser0* and check that the output looks correct.

NOTE: Older versions of usb4rt (on the branch cdc-acm) required you to limit your system RAM to 4GB. Failing to do this created communication issues because data got read into memory which the driver didn't know about, causing the read command to never return. To limit the memory, simply edit /etc/default/grub and add "mem=4G" to the GRUB_CMDLINE_LINUX_DEFAULT line. This is **NOT** necessary for the current version of usb4rt (master branch).

Debugging usb4rt

If for some reason the above instructions did not work, you can find long output with common debug enabled when drivers have been successfully loaded here. You may also try the following useful commands to find the problem:

1. Use *lsusb* to list USB devices, then choose the one you're interested in and use *lsusb -vd <mfgname>:<devname>* to get more details about it.
2. You can create your own device rules in /etc/udev, similar to 70-persistent-net-rules.d (the "70" specifies the order in which these rules are used).

Issue Log

- problem with /proc/xenomai/stat (watching this file creates crazy latencies). FIX: Use the patch latency.patch and recompile the kernel.

- With Athena and the motherboard intel graphics chispet the latencies are very high when using x-window. This problem seem to be removed when using an AMD graphics card instead (the one that is inside hermes). However, the onboard graphics chipset works fine on Perseus with the VESA driver which gets enabled by adding "nomodeset" to the GRUB kernel command line arguments. Hypothesis: the proprietary Intel driver installed on Athena created issues, so just don't install any drivers. This allows us to use the 2 PCI slots for the ethernet and USB 1.1 cards.
- With Hermes and Athena, both of which had external graphics cards installed, the above solution works but only with specific monitors - strange. Been working with the following links... probably need to configure the graphics settings per-monitor which is annoying.
    - http://community.linuxmint.com/tutorial/view/842
    - http://superuser.com/questions/451855/opengl-on-ubuntu
    - https://bugs.launchpad.net/ubuntu/+source/consolekit/+bug/475503

- Both rtnet and usb4rt seem to work fine with SL and communicating with the GDC cards/IMU on perseus. The lower body simulator runs very smoothly and shows no latency issues. The upper body simulator is very laggy when starting but also show no latency issues. The real lower body shows latencies of about 15-20us on average (normal is 5-10us) while the real upper body shows latencies of 20-30us on average (slightly higher). However, this seems reasonable.

- I collected data from the IMU while running SL by inserting some code into ImuInterface.cpp which rt_fprintf's the difference between consecutive IMU reads to a logfile. We expect to see roughly 1ms with periodic portions of 2ms latencies (when rt_uhci and the device are "out of sync" and playing catch-up as Jan explained). This is indeed what we see; check http://www-clmc.usc.edu/~nrotella/Xenomai/good_log.log for the data. The explanation for the spikes is as follows:

```
The
UHCI controller and the IMU have separate, unsynchronized clocks.
Apparently, the IMU is a bit slower.

UHCI    |        |        |        |
IMU  |          |          |          |
                ^^^^^^^^^^^^
You see, there can be a gap in the transmission of up to one extra ms
(the USB cycle period). The delta between the clocks should be small and
not monotonously increasing. So when is comes to the scenario above, the
IMU may first miss a frame, then catch up in the next one (period - 1
ms), then miss again, all this over several cycles, until it finally
"stabilizes" again, delivering periodically.
```

Random Notes:

Reinstalling Ubuntu while preserving partitions

If you wish to reinstall Ubuntu but keep your home directory and other partitions (for example, a backup) intact, you can do so as follows. First, make sure you know which partitions correspond to which mounting points by checking the Disk Utility; this will be essential in reinstalling. Next, create a bootable USB stick with the flavor of Ubuntu you already have installed (we use 12.04). Boot from this USB stick - this is done by inserting the drive into the machine, rebooting and holding the proper key during the boot (on Dell desktops, F12) to reach the BIOS. Once in the BIOS, choose to boot from the USB drive (usually from the Legacy BIOS option, not UEFI BIOS). When the Ubuntu LiveCD menu comes up, choose "Install Ubuntu" and continue through the options until you choose "Something Else" which lets you customize the installation. At this point, you will see the machine's partition table as it remains from the current installation. Set up the partition types (generally ext4) and mounting points (/, /Volumes/COMPUTER_NAME, and optionally /Volumes/backup and whatever else) exactly as they were when you checked Disk Utility. Now, mark ONLY the root partition (to which Ubuntu is installed) to be formatted. NOTHING ELSE SHOULD BE FORMATTED! You've now preserved the structure of your installation and can continue with the install process. When the machine reboots after installation is complete, remove the USB stick and hope everything works!

Solving the "invalid arch independent ELF magic" GRUB issue

If you reach a scary "grub rescue" prompt with the error "invalid arch independent ELF magic," fear not. This indicates that your grub install got messed up, but your new Ubuntu installation and data are fine. This may have happened if you chose the wrong USB boot (UEFI instead of Legacy) from the BIOS; if the following short procedure doesn't work, you can try reinstalling and selecting this boot option instead. Reboot the machine (hard reset if necessary) and insert the install USB again, boot from the USB in the BIOS and now choose "Try Ubuntu." Note that the grub rescue prompt may prevent you from reaching the BIOS; you probably just didn't hit F12 (or whatever key) at the right time. I managed to get to the BIOS every time by repeatedly tapping F12 repeatedly as soon as the boot process began. Once you manage to get to a desktop using the Try Ubuntu option, open a terminal and run the following commands, replacing /dev/sdXY with the mounting point of / chosen in your installation (for example, if / is mounted at /dev/sdb1 then X=b, Y=1):

```
sudo mount /dev/sdXY /mnt

sudo grub-install --boot-directory=/mnt /dev/sdX
```

Reboot and everything should work normally again! If not, you may also have to install the *grub-efi-amd64* package from the live CD. If you absolutely cannot reach the BIOS to get to the live CD, it should also be possible to boot directly from the grub rescue prompt (check online for commands).

SSH woes

If you get the warning "No xauth data; using fake authentication data for X11 forwarding." when you try to ssh into another machine, you can simply disable X11 forwarding by editing your ~/.ssh/config file (which may not yet exist - create it if necessary). Make sure it read as follows:

```
Host *
    ForwardAgent yes
    ForwardX11 no
```

where the wildcard * is probably not the best idea but it prevents you from needing a list of ssh hosts.

Random useful things to know:

- To run single commands during the boot process, add them to the script /etc/rc.local which gets run at the end of each multiuser runlevel (traditionally 2-5).
- If you get the error "Error while getting interface flags: no such device" it likely means that the appropriate driver for your card is not loaded (verify this by running *lshw -C network* and see that the card is marked UNCLAIMED). Fix it with *modprobe e1000e*
- To manually bind a PCI device to its driver, run *echo -n "PCI address" > /sys/bus/pci/drivers/e1000e/bind* where the PCI address can be found in the 70-persistent-net.rules file. To manually unbind, run *echo -n "PCI address" > /sys/class/net/eth4/device/driver/unbind*. Note that this isn't only for PCI devices/drivers but anything else too. You can only bind/unbind loadable modules.
- If you have graphics issues, install the package "hwinfo" and run *sudo hwinfo --framebuffer.* You may also need to install the package "hal" if you get related errors. This command tells you video modes which your graphics card can support; you can try setting these manually using their hex Mode values and adding "vga=XXX" to your GRUB_CMDLINE_LINUX_DEFAULT grub line with XXX replaced by the decimal value of the hex Mode value of the video mode.
- If you run into issues such as an infinite black or purple screen preventing you from reaching a login prompt, it may be due to the fact that you're booting Ubuntu from an SSD. One workaround is to add a delay by adding "sleep 2" (for two seconds, or more if needed) in the file `/etc/init/lightdm.conf` above "exec lightdm." A better solution is to switch to using gdm which seems to work fine with SSD's. This is apparently a common problem which happens randomly. If all else fails, drop to a virtual terminal as root and run "service lightdm restart."
- To switch to gdm, first install it and then run *dpkg-reconfigure gdm* and then *service lightdm stop* and finally *service gdm start*.

How to install Xenomai?

Realtime software guidelines