TASTE-Linux distribution documentation v1.1

Julien Delange

May 26, 2011

Contents

Chapter 1

Introduction to the TASTE Linux distribution

The TASTE Linux distribution was designed to be used for the implementation of Robotics applications. However, it can be tailored for the development of other real-time applications.

The distribution is based on Debian, stable version. On top of Debian, we build a customized kernel based on the vanilla kernel (from `http://www.kernel.org`) and the Xenomai patch.

1.1 Availability

The latest version of the distribution is available on the internet, at the following address: `http://download.tuxfamily.org/taste/linux-taste/linux-taste.img`

Chapter 2

Use the TASTE Linux distribution

2.1 Running the distribution

2.1.1 Deploy on an USB stick

You can put the distribution on a USB stick in order to be able to deploy it automatically on a system. To do so, download the image and issue the following command on Linux:

```
dd if=taste-linux.img of=/path/to/your/usb/key/device
```

It will write the taste linux image on your usb stick so that you can run the distro from the usb stick. Be careful, this would erase all existing data on your usb stick and even destroy the existing filesystem (partitions and so on).

2.1.2 Emulation with QEMU

To run the image, you can use qemu for rapid development/prototyping. The following command line would start the distro in qemu :

```
qemu -hda taste-linux.img -boot c -net nic,vlan=0  \
   -net tap,vlan=0,ifname=tap0,script=/etc/qemu-ifup -m 256 -localtime
```

Increase QEMU speed

You can run the system faster by using the virtualization functions of your architecture (if supported). In that case, use the -enable-kvm option when qemu is launched. Make sure that the following modules are loaded into your kernel (depending on your processor manufacturer): `kvm-intel` or `kvm-amd`).

Networking support within QEMU

If you want to connect your QEMU instance with your Linux system, you have to configure a bridge. On regular Debian/Linux system, you can do that be editing your network configuration. Proceed as it :

1. Edit `/etc/network/interfaces` and add the following lines (assuming that your computer is connected using `eth0` and that your network connection is configured with dhcp):

   ```
   auto br0
   iface br0 inet dhcp
   bridge_ports eth0
   bridge_fd 9
   bridge_hello 2
   bridge_maxage 12
   bridge_stp off
   ```

2. Restart the network by ussing the following command:

   ```
   /etc/init.d/networking restart
   ```

3. Edit `/etc/qemu-ifup` and put the following lines:

   ```
   #!/bin/sh
   echo "Executing /etc/qemu-ifup"
   echo "Bringing up $1 for bridged mode..."
   sudo /sbin/ifconfig $1 0.0.0.0 promisc up
   echo "Adding $1 to br0..."
   sudo /usr/sbin/brctl addif br0 $1
   sleep 2
   ```

4. Launch QEMU with these additional parameters:

   ```
   -net nic,vlan=0 -net tap,vlan=0,ifname=tap0,script=/etc/qemu-ifup
   ```

   So, the following command would start qemu with KVM support and network configured:

   ```
   qemu -hda taste-linux.img -boot c \
   -net nic,vlan=0 -net tap,vlan=0,ifname=tap0,script=/etc/qemu-ifup \
   -m 256 -localtime -enable-kvm -clock rtc
   ```

5. If you want to avoid to type your password each time the `/etc/qemu-ifup` script is invoked, you have to configure sudo. In that case, invoke `visudo` (as root) and change your configuration like this:

   ```
   Cmnd_Alias      QEMU=/sbin/ifconfig,/sbin/modprobe,/usr/sbin/brctl
   youruser ALL=NOPASSWD: QEMU
   ```

   or

   ```
   Cmnd_Alias      QEMU=/sbin/ifconfig,/sbin/modprobe,/usr/sbin/brctl
   %yourgroup ALL=NOPASSWD: QEMU
   ```

2.2 Accounts and password

The root password is:

```
taste
```

The system also defines a regular user account:

- **Username:** taste
- **Password:** taste

2.3 Directories and path

The following directories have to be considered if you plan to develop for this target/platform:

- Xenomai main directory: `/usr/xenomai`
- Xenomai libraries: `/usr/xenomai/lib`

2.4 Configuration of the analogy devices

First of all, you have to load the appropriate kernel module (for example `analogy_ni_pcimio`). Then, you have to invoke the `analogy_config` from Xenomai tools to be able to load and control the card. Invoke it like this:

```
analogy_config DEVICEID DEVICEDRIVER
```

For example, you can invoke the tool as it:

```
analogy_config analogy0 analogy_ni_pcimio
```

Then, in that case, the device would be available in the analogy framework in Xenomai under the name `analogy0`.

2.5 Kernel modules automatically loaded

When the system starts, the following modules are automatically loaded:

- `xeno_posix`
- `xeno_native`

2.6 Kernel modules available

The kernel modules specific to Xenomai are located in the following directories:

- `/lib/modules/KERNEL_VERSION/kernel/kernel/xenomai`
- `/lib/modules/KERNEL_VERSION/kernel/drivers/xenomai`

The following modules are available:

- `xeno_nucleus`
- `xeno_uitron`
- `xeno_vrtx`
- `xeno_native`
- `xeno_vxworks`
- `xeno_rtdm`
- `xeno_psos`
- `xeno_posix`
- `xeno_analogy`
- `analogy_s526`
- `analogy_parport`
- `analogy_8255`
- `analogy_ni_mio`
- `analogy_ni_tio`

- `analogy_ni_pcimio`
- `analogy_ni_mite`
- `analogy_fake`
- `analogy_loop`
- `xeno_rtipc`
- `xeno_rtdmtest`
- `xeno_klat`
- `xeno_switchtest`
- `xeno_irqbench`
- `xeno_timerbench`
- `xeno_sigtest`
- `xeno_can_peak_pci`
- `xeno_can_mem`
- `xeno_can_esd_pci`
- `xeno_can_ems_pci`
- `xeno_can_peak_dng`
- `xeno_can_sja1000`
- `xeno_can_plx_pci`
- `xeno_can_isa`
- `xeno_can_ixxat_pci`
- `xeno_can_virt`
- `xeno_can`
- `xeno_16550A`

2.7 Configuration of the kernel

The kernel sources are available in `/usr/src/linux`. The sources of Xenomai are also available in the `/usr/src` directory. The configuration file of the kernel is available in `/usr/src/linux/.config`.

2.8 Rebuild your own kernel

You can rebuild your own kernel by issuing the following command in the `/usr/src/linux` directory:

```
make-kpkg --initrd kernel_image
```

2.9 Communication with the machine

A ssh server is installed within the VM. File transfer operations and access to the machine can be done using ssh of any sftp-compliant program.

Chapter 3

Use the Analogy layer for interaction with acquisition data boards

The TASTE linux distribution embedds the xenomai real-time kernel. Is also includes an analogy layer which purpose is to interface with acquisition boards, such as the one supported by the comedi drivers.

This analogy layer provides user functions to communicate with acquisition boards in a uniform and smooth way. In the following, we explain where you can find relevant documentation to use the analogy layer and provide an example of the use of these boards.

3.1 Documentation of the analogy layer

User-level functions that interacts with the analogy layer of Xenomai are documented in the Xenomai API document. It is available on the Xenomai website (`http://www.xenomai.org`), either in PDF or HTML format[1].

3.2 Use the board: the `analogy_config` tool

First of all, you must define the boards that will be used by your system. You do that using the `analogy_config` tool. This tool specify which driver (kernel module) is used for each board. Each board is identified using a number, starting from 0 (so, the first board will be `analogy0`).

So, to associate an analogy device with a driver, you just have to invoke the following command:

```
analogy_config analogy_device kernel_module
```

For example:

```
analogy_config analogy0 analogy_ni_pcimio
```

[1]you can check out this page for example `http://www.xenomai.org/documentation/xenomai-2.5/html/api/index.html` and search within the `Analogy API` module

In this case, the card `analogy0` is identified as using the kernel module `analogy_ni_pcimio`. When using the analogy API, you will refer to it using the name `analogy0`.

3.3 Use the board and the analogy API

First of all, you have to include the header file `analogy.h` provided by the analogy layer of xenomai. For that, you have to use the following pre-processing code:

```
#include <xenomai/analogy/analogy.h>
```

Then, you have to open the device. This is done by using the function `a4l_open` function which opens the device and stores relevant information about it in a structure with the type `a4l_desc_t`. The following listing illustrates how to use this function (note that a return code different from 0 means that an error was raised when opening the device) :

```
a4l_desc_t arm_device;
ret = a4l_open (&arm_device, "analogy0");

if (ret != 0)
{
   printf ("[EXOARM] Error while opening the arm device, return code=%d \n", ret);
   return;
}
```

Then, once the device is opened, you need to fill its descriptor informations. This is done by calling the `a4l_fill_desc` function, like this:

```
ret = a4l_fill_desc (&arm_device);

if (ret != 0)
{
   printf ("[EXOARM] Error while calling fill_desc(), return code=%d \n", ret);
   return;
}
```

3.3.1 Getting channel information

Then, for each channel of the acquisition board, you need to get channel information. This is done with the `a4l_get_chinfo` function. For example, the following function call will retrieve information for the first channel of the device opened in the `arm_device` variable. Note that a return code different from 0 means that an error happened.

```
a4l_chinfo_t channel_infos;

ret = a4l_get_chinfo (&arm_device, 0, 0, &channel_infos);
if (ret != 0)
{
   printf ("[EXOARM] Error invoking a4l_get_chinfo\n");
}
```

3.3.2 Getting range information

Then, for each channel, you need to get the range information, which indicate to maximum and minimum value that can be acquired on a particular channel. You can get this information by calling the `a4l_get_rnginfo` function. It will store information about data range in a `a4l_rnginfo_t` structure, that will be used later when acquiring/converting data.

The following listing shows how to get range informations for the first channel on the device opened using the `arm_device` variable.

```
a4l_rnginfo_t range_infos;
ret = a4l_get_rnginfo (&arm_device, 0, 0, 0, &range_infos);
if (ret != 0)
{
   printf ("[EXOARM] Error invoking a4l_get_rnginfo\n");
}
```

3.3.3 Acquiring data

Then, you acquire the data using the `a4l_sync_read` function. This function gets *raw* data, meaning that the acquired data cannot be used directly and has to be converted later on in a programming-language dependent type (for example, an `int` or a `double`).

So, when you acquire a data, you just have to provide a buffer to store the data and specify its size. In the following listing, we acquire data on the first channel and store it in the buffer `raw`, which has a length of 128 bytes.

```
uint8_t raw[128];
ret = a4l_sync_read (&arm_device, 0, 0, 0, &raw, 128);
if (ret <= 0)
{
    printf("[arm] Error while acquiring the data\n");
}
```

3.3.4 Convert RAW data to programming types

Finally, you need to convert the RAW data acquired previously to a type that can be used in your programming language. This is done with the serie of functions `a4l_rawto*`. To convert the *raw* data into a `double` type, you can use the `a4l_rawtod` function.

Conversion function requires the channel and range informations that has been taken previously (see the previous sections). The following listing illustrates how you can convert the *raw* data acquired previously into a `double` type using the channel and range informations obtained previously in the variable `range_infos` and `channe_infos`.

```
double converted_value;
ret = a4l_rawtod (&channel_infos, &range_infos, &converted_value, &raw, 1);

if (ret <= 0)
{
    printf("[arm] Error while converting the data\n");
}
```

3.3.5 Initialization and acquisition example

The following listing shows an example of the use of an acquisition board. We assume the board was initialized using the `analogy_config` and opened as `analogy0`. Then, it opens the data, get range and channel informations and finally acquires data on the first channel every second. It finally outputs the data on the standard output.

```
#include <stdio.h>
#include <xenomai/analogy/analogy.h>

int main ()
{
   int ret;
   a4l_desc_t       arm_device;

   a4l_chinfo_t   channel_infos;
   a4l_rnginfo_t  range_infos;

   double         val;
   uint8_t        raw[128];


   ret = a4l_open (&arm_device, "analogy0");

   if (ret != 0)
   {
      printf ("[EXOARM] Error while opening the arm device, return code=%d \n", ret);
      return;
   }
   arm_device.sbdata = malloc(arm_device.sbsize);


   ret = a4l_fill_desc (&arm_device);

   if (ret != 0)
   {
      printf ("[EXOARM] Error while calling fill_desc(), return code=%d \n", ret);
      return;
   }


   ret = a4l_get_chinfo (&arm_device, 0, 0, &channel_infos);
   if (ret != 0)
   {
      printf ("[EXOARM] Error invoking a4l_get_chinfo\n");
   }

   ret = a4l_get_rnginfo (&arm_device, 0, 0, 0, &range_infos);
   if (ret != 0)
   {
      printf ("[EXOARM] Error invoking a4l_get_rnginfo\n");
   }

   while (1)
   {
      ret = a4l_sync_read (&arm_device, 0, 0, 0, &raw, 128);
      if (ret <= 0)
      {
          printf("[arm] Error while acquiring the data\n");
      }

      ret = a4l_rawtod (channel_infos, range_infos, &val, &raw, 1);

      if (ret <= 0)
      {
          printf("[arm] Error while converting the data\n");
```

```
            }
        }
    }
```

Chapter 4

Development for the TASTE Linux distribution

4.1 Available software

The following development tools are available within the distribution:

- binutils
- GCC
- Xenomai with `POSIX` and `Native` skins

Chapter 5

Annexes

5.1 Links and websites

- Debian - http://www.debian.org
- Xenomai - http://www.xenomai.org
- Comedi - http://www.comedi.org
- System configuration for QEMU and network concerns: http://compsoc.dur.ac.uk/~djw/qemu.html

Bibliography

[1] T. Vergnaud, B. Zalila, and J. Hugues. Ocarina: a Compiler for the AADL. Technical report, Télécom Paris, 2006.

[2] SAE. *Architecture Analysis & Design Language v2 (AS5506A)*. SAE, jan 2009. available at `http://www.sae.org`.

[3] SAE. *Data Modeling Annex for the Architecture Analysis & Design Language v2 (AS5506A)*. SAE, nov 2009. available at `http://www.sae.org`.

[4] SAE. *Programming Language Annex for the Architecture Analysis & Design Language v2 (AS5506A)*. SAE, nov 2009. available at `http://www.sae.org`.

[5] Jerome Hugues and Bechir Zalila. *PolyORB High Integrity User's Guide*, jan 2007.

[6] Thomas Vergnaud, Bechir Zalila, and Jerome Hugues. *Ocarina: a Compiler for the AADL*, jun 2006.

[7] Xenomai Team. *Xenomai API documentation*, 2011.