RTAI Installation Complete Guide

Version 1 - 2/2008

João Monteiro
jmonteiro@alunos.deec.uc.pt

February 27, 2008

Abstract

This article covers all the steps needed to install RTAI on a Debian based Linux distribution, including a deep overview of the kernel configuration. The main goal is to provide all the necessary information in a detailed and comprehensive way about the subject, even for those who possess little or no knowledge about the kernel compilation process.

1 Introduction

I started creating this guide, so that in the future I could remember all the steps I made to get RTAI working on the RAC[1] Development PC. I followed a very good Brazilian guide [*Tutorial: Instalando RTAI, Renato Bensen*], but still i had to make significant modifications to have it working. Also, it doesn't include a detailed explanation on how to configure the kernel. This step was very important to me because I will need to have a very compact kernel to fit on an embedded system to fully run in the RAM. Because of this, I decided to include the tune up procedures to configure a kernel as light as possible, and still having the basic everyday needed services like USB support and wireless communication. I noticed that in the kernel configuration that comes with Ubuntu, a lot of functions are enabled to maximize the compatibility of Linux in several hardware systems. The problem is that these options tend to raise the kernel size, its compilation time, slow down the boot process to check for unnecessary services, and enable processes to check features that will never exist, resulting in unnecessary CPU time consumption. So, to help people get a light and working kernel with real time layer, I decided to go more deep on the detail through this article.

NOTE: If errors are detected, please report to my e-mail so I can fix the document.

2 System requisites

2.1 Hardware

- Processor – Intel Processor (Any should work). The RAC Development PC has a Pentium III at 700 MHz. It perfectly fits the needs. I didn't tested for other architectures.
- Disk Capacity – Relatively Large (9GB). I first used a 3.24GB hard disk, but the kernel compilation stage exceeded the 1.1GB of free space that i had on this drive.
- RAM memory – 256MB minimum is recommended. I first used 128MB, but the compilation process and OS installation is very very slow. For my setup I then used two 128MB plus two 64MB RAM cards, giving approximately 380MB of total available RAM.
- CD-ROM Drive - 50x . Slower drives will slow down the SO setup procedure.

2.2 Software

- Operative System – Xubuntu Linux 7.04. This is a very light distribution which uses Xfce as desktop manager. It will run very well on old computers such as the one I used.
- Compilation – gcc v3.4, g++ v3.4 and make . To check your gcc and g++ version, simply open a shell and type,

  ```
  $ gcc --version
  $ g++ --version
  ```

  which will print the version of these programs. I discourage using lower versions than the recommended ones. Type the following in a shell,

  ```
  sudo -s
  ```

  which will allow you to mess around with super user privileges through the installation process. Now, to get the above mentioned packages, type the following,

  ```
  # apt-get install gcc-3.4 g++-3.4 make
  ```

[1] Robotica Academica de Coimbra robot-soccer team
www.rac-uc.pt.vu

I always used the recommended version of gcc and g++, which work perfectly. Don't worry if you have version 4.1.3 or higher already installed in your system. When you do the apt-get command above, the packet manager installs the lower version in /usr/bin, but lets the newest version available and running as default. To use the recommended versions, you will need to specify it at compilation stage, as we will se. By the way, if you use newer versions of these packages with success please inform me so I can update the text and state with absolute certainty that they work.

- Basic kernel configuration menu – libncurses5-dev. This package installs the needed Curses libraries that we will need to run the kernel configuration menu. Simply type:

  ```
  # apt-get install libncurses5-dev
  ```

- Module Loader – module-init-tools. These tools will be needed to load kernel .ko modules, such as the rtai_hal.ko. Get it typing the following:

  ```
  # apt-get install module-init-tools
  ```

- RTAI Version – 3.5 . This is the newest RTAI version until now, and has patches to the recent 2.6.19 kernel version. To get this software, first enter the source directory of your Linux distribution:

  ```
  # cd /usr/src
  ```

  Now, get the RTAI tarball to this same folder,

  ```
  # wget --no-check-certificate https://www.rtai.org/RTAI/rtai-3.5.tar.bz2
  ```

  and unpack the tarball,

  ```
  # tar xvf rtai-3.5.tar.bz2
  ```

  which creates the folder rtai-3.5.tar.bz2 on your /usr/src directory.

- Vanilla[2] Kernel - 2.6.19. This is the newest version of the kernel supported by RTAI 3.5. The recent 2.6.X version possesses very good features that improve task responsiveness, so using older versions is strongly discouraged. The vanilla Kernel is a clean version of the kernel sources, which possess no configuration at all. These will be used to build our own custom kernel. To get the vanilla, first jump to the sources directory,

  ```
  # cd /usr/src
  ```

  and get the tarball from the kernel.org servers:

  ```
  # wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.19.tar.gz
  ```

  After the kernel source download is complete, unpack the tarball[3],

  ```
  # tar xvf linux-2.6.19.tar.gz
  ```

  which unpacks the kernel sources to the /usr/src/linux-2.6.19 folder. At this point, you have a clean version of the kernel sources with 2.6.19 version.

[2] Clean – not configured – kernel sources

[3] If you are wondering what a tarball is, it's a geek term that refers to a tar archive in which files are packed.

3 Apply RTAI patch

Now that we have the necessary tools, we are able to start working. The RTAI kernel patches are applied to our vanilla kernel, so it can later support the RTAI kernel layer. To do so, enter the folder of your vanilla kernel sources,

```
# cd /usr/src/linux-2.6.19
```

and apply the patch.

```
# patch -p1 -b < ../rtai-3.5/base/arch/i386/patches/hal-linux-2.6.19-i386-1.7-01.patch
```

You may use the TAB key to auto complete the commands on the shell. After this, several lines will show that the patch is being applied.

4 Configuring the Kernel

I found it hard to fully understand some options of the kernel configuration menu. Since I needed a light kernel for my embedded system, i needed to remove a lot of default options that came by default with the Xubuntu kernel version. For instance, if you install a fresh Ubuntu on a desktop pc, the kernel comes by default prepared to load Toshiba and Dell laptop modules. What for?

For this step you will need to recognize your hardware. I will instruct you to configure the vanilla kernel on a desktop PC, which may also be useful for laptops. For the latter case, please pay attention to specific laptop modules that will be presented.

First, you will need to save your current kernel configuration file. Most of the Linux distros store a copy of this file, which will help us with a starting configuration. So, enter in the vanilla kernel source folder if you are elsewhere,

```
# cd /usr/src/linux-2.6.19
```

and make a copy of the existing configuration file to the root folder of your vanilla kernel source.

```
# cp /boot/config-2.6.19-generic .config
```

Lets finally run the kernel configuration menu

```
# make menuconfig CC=/usr/bin/gcc-3.4 CXX=/usr/bin/g++-3.4
```

The Curses based menu will show up. Now let's load the complete configuration file that we saved as .config by choosing the "Load an Alternate Configuration File" option and typing .config – if it isn't already as default. At this point we loaded the kernel configuration of the default Linux installation which is running on your system. Lets do the necessary changes.

4.1 The configuration

I haven't found a complete and working guide about this step. In every guide, something was missing, making me have to compile the kernel repeated times because an option or another wasn't or was wrongly marked. So, here i will state all the configuration options I used. The REALLY important options needed to get RTAI working without (hopefully) having to recompile the kernel are marked with a "¦‘. I will explain the most relevant options, and state ONLY the ones I marked. Back to the kernel configuration menu, do the following:

- Code maturity level options ->
  - Nothing selected
- General Setup ->
  - [*] Support for paging of anonymous memory (swap) – Support for swap (virtual memory).

  - [*] System V IPC – Allows Inter Process Communication.
  - [*] BSD Process Accounting – Allows to obtain user application information's.

- Loadable Module Support ->
  - !![*] Enable loadable module support – Allows to load modules to the kernel with the loading tools.
  - [*] Module unloading.
  - [*] Source checksum for all modules.
  - [*] Automatic Kernel module loading .

- Block Layer ->
  - Nothing selected

- Processor type and features ->
  - [*] Generic x86 support – Better kernel performance on x86 architecture CPU's.
  - [*] Preempt The Big Kernel Lock – Reduces latency of the kernel on desktop computers.
  - [*] Interrupt pipeline – Prevent data disturbances
  - !![ ] Local APIC support on uni processors – MUST be deactivated or the error RTAI[hal]:ERROR, LOCAL APIC CONFIGURED BUT NOT AVAILABLE/ENABLED will show when running RTAI apps.
  - [*] Math emulation – Emulates co-processor for loading point operations on old CPU's.
  - [*] MTRR support
  - !![ ] Use register arguments – this MUST be deactivated.
  - [*] Compact VDSO support

- Power management options ->
  - [*] Legacy Management Debug Support
  - ACPI Support –>
    - [*] ACPI Support – Advanced Configuration and Power Interface support.
    - [M] Button
    - [M] Video
    - [M] Fan
    - [M] Processor
    - [M] Thermal Zone
  - CPU Frequency Scaling –>
    - [*] CPU Frequency scaling – Allows to change the clock frequency of the CPU on the fly.
    - [*] Relaxed speedstep capability checks – Does not perform all checks for speed up.

- Bus options ->
  - [*] PCI Support

- Executable file formats ->
  - [*] Kernel support for ELD binaries

- Networking ->
  - Networking Options –>
    - [*] Packet socket: mapped IO – Speed up communications.

        * [*] Unix domain sockets – Support UNIX sockets.
        * [*] TCP/IP networking – And all this option's derivatives will be marked.
        * [*] Network packet filtering
        * [*]QoS and/or fair queueing

- Device Drivers ->
    - Generic Driver Options –>
        * [*] Prevent firmware from being built.
        * [*] User space firmware loading support.
    - Memory Technology Devices (MTD) –>
        * [*] Write support for NFTL.
    - Plug and Play support –>
        * [*] Plug and Play support.

      The rest of the options use the default configuration of the running kernel.

- File Systems – Default configuration used.
- Kernel Hacking ->
    - !! [ ] Compile the kernel with frame pointers – MUST be deactivated The rest of the options use the default configuration of the running kernel.
- Security options – Default configuration used.
- Cryptographic options – Default configuration used.
- Library routines – Default configuration used.

This process is complex and requires that you know your hardware if you want to optimize the kernel's performance. The presented configuration should work in any desktop equipped with a x86 CPU, and all the base services like USB's, wireless and cable network etc., are available.

This configuration resulted on a vmlinuz compressed kernel file vmlinuz-2.6.19-rtai of 1.396 MB and an initrd kernel file of initrd.img-2.6.19-rtai of 5.140 MB to boot the vmlinuz kernel from the SCSI hard disk device. Very good eh?

5 Compiling the kernel

After configuring the kernel, we need to go to the compilation process. For the Ubuntu distribution, the best way to install the kernel is to compile it and create a *.deb installation file. For this, some software packages are needed.

```
# apt-get install kernel-package fakeroot
```

Now, the following commands should be run to clean and compile the kernel.

```
# make-kpkg clean
# fakeroot make-kpkg --initrd --app\end-to-version=-rtai \kernel_image kernel\_headers
```

This process can take hours to complete. In the RAC PC system, approximately 2 hours are needed.

When the process is finished, you should see a line like *echo done*. Now is the time to install the two *.deb packages that are created in the /usr/src folder by the compilation process.

```
# cd /usr/src
# dpkg -i *.deb
```

After the dpkg program finishes the installation, a new entry in the menu.lst of the grub will be added. So now if we reboot the system, this new entry will boot the Linux system with our new kernel.

6 The embedded style

I had a problem when booting the new kernel, the Xubuntu boot screen didn't show up while booting. The screen went black while booting the system until the Xfce desktop manager is loaded. For the embedded system that I will further target, I don't need the Xfce nor the boot screen to load at startup, so i did the following:

1. Reboot and select the second option of the grub bootloader : Ubuntu, kernel 2.6.19-rtai (recovery mode).
2. Open the grub menu.lst with nano:

   ```
   # nano /boot/grub/menu.lst
   ```

3. Remove "quiet splash" after the "ro" directive of the kernel line on our new kernel entry.
4. Press CTRL+X and then press "Y" and ENTER.
5. Reboot

   ```
   # reboot
   ```

If the same happened in your case, now you should see the boot process in command line. For me this fits best, since i want to see the whole boot process.

Now, i wanted to remove the XFCE from starting as all the devices are loaded. For this, I did the following,

```
update-rc.d -f gdm remove
```

And finally, I have a boot screen by command line, and no X starting involuntarily. Note, however, that you can log-in in command line and then type "xstart" to start the X.

7 Configuring RTAI

If the above steps are completed, boot up your new kernel. Now, enter the rtai-3.5 folder and create a new one for build:

```
$ sudo -s
# cd /usr/src/rtai-3.5
# mkdir build
# cd build
```

Now lets configure the RTAI.

```
# make -f ../makefile CC=/usr/bin/gcc-3.4 CXX=/usr/bin/g++-3.4
```

The following options should be verified in the ncurses menu that will show up:

- General -> Installation directory – Leave the default as /usr/realtime
- General -> Linux Build Tree – The path to the configured kernel /usr/src/linux-2.6.9

All set, now just exit and reply YES to save your configuration. Now, install RTAI:

```
# make install
```

Now reboot your computer and boot our new kernel with RTAI installed.

8 Testing the installation

To get things working, you will need to run a script on Ubuntu startup that will link folders that are not created nor linked as the RTAI installs (BUG?). For this, do "startx" if you are at the command prompt to start Xfce. The desktop manager will allow you to copy and paste the scripts. Do the following:

1. Run "sudo -s" to gain root privileges.
2. run mousepad or any text editor you like.
3. Copy and paste the following code on it.

```
 #!/bin/bash
mkdir /dev/rtf
for n in `seq 0 9`
do
         f=/dev/rtf/$n
         mknod -m 666 $f c 150 $n
done
```

4. Save it in your home folder /home/Your_User_Name with the name rt_script
5. Go to the home directory:

```
# cd /home/Your_User_Name
```

6. Set the rt_script file executable:

```
# chmod +x rt_script
```

7. Run the script

```
# ./rt_script
```

This will create the directories rtf0..rtf9 that, for unknown reasons, are not created by RTAI but are needed by it. Now you will need to run a script on Ubuntu startup that will create the nodes to these folders.

1. Run mousepad or any other text editor.
2. Copy and paste the following code on it.

```
#!/bin/bash
mknod -m 666 /dev/rtai_shm c 10 254
for n in `seq 0 9`
do
         f=/dev/rtf$n
         mknod -m 666 $f c 150 $n
done
```

3. Save it in /etc/init.d/ with the name start_rt
4. Make the script executable:

```
# chmod +x /etc/init.d/start_rt
```

5. Make it run on startup:

```
sudo update-rc.d start_rt start 51 S .
```

NOTE: Do not forget the dot at the final of the above command.

This script will make the nodes at system startup so you don't have to do it over and over again. In some cases this problem does not happen but since it happened to me, I felt the need to share this. The symptom was something like "Error:Cannot open /dev/rtf3" when running the real time test application.

Now you will have to load the rtai_hal.ko module:

```
# cd /usr/realtime/testsuite/modules
# insmod rtai_hal.ko
```

If everything goes OK, there should be no output messages after the insmod command.

It's time to finally test our installation. For this, do the following:

```
# cd /usr/realtime/testsuite/kern/cd latency
# ./run
```

If everything is correctly installed, you should see something like,

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH|    lat min|    ovl min|    lat avg|    lat max|    ovl max|   overruns
RTD|       -628|       -628|       1625|      10933|      10933|          0
RTD|        805|       -628|       1609|      13756|      13756|          0
RTD|       -781|       -781|       1622|      17424|      17424|          0
```

And that's it! Happy programming :) For questions, doubts, critics or bug reports about this article feel free to e-mail me.