# A Guide to Installing RTAI Linux

Keith Shortridge, Anglo-Australian Observatory,
P.O. Box 296, Epping 1710,
NSW, Australia

12th November 2004

## Contents

## 1 Introduction

This guide is for people who want to install and use the RTAI hard real-time Linux system, but who've never needed to know much about Linux kernels and how to build them.

This document has its origins in my initial experiences trying to install RTAI Linux on a PC. RTAI Linux is a 'hard' real-time operating system, and we were interested in evaluating it as a possible replacement for the VxWorks real-time systems we were used to. This meant that someone had to find out how to build it, install it, program it, and test it. Everyone else took one step back, and I got the job. What I found was that RTAI Linux is an extremely capable real-time system, but you need a good background in Linux kernel building to work out just how to get it on to a machine in the first place. I did not have this background; I was an experienced Linux user and programmer, but was no Linux system person. And the documentation that went with RTAI Linux had the general air of being written by experts for experts. So this is the 'dummies guide' to installing RTAI Linux that I wish I'd had available when I started. I suspect there are others in my position: programmers who want to write real-time programs, but who do not have a Linux system background and want a simple explanation of how to get RTAI Linux running from scratch. I hope it helps.

If you can build a Linux kernel, installing RTAI is easy. If you don't know how to build a Linux kernel, getting the hang of that isn't too difficult, and maybe this will help. This guide has been arranged in what I think is the most useful way: I start with an overview of the whole process, together with a complete set of steps for building and installing RTA from scratch. If you're really lucky, this exact sequence will work for you (well, your home directory probably isn't called /home/ks, but you should be able to spot the really obvious changes to cover that). If this doesn't work, I follow this with a detailed explanation of each step. And then there's a background section to help fill in some of the stuff you need to know to understand the detailed explanations. I usually reckon that if you understand why an example is the way it is, you can see what you need to do to change it to work in your case.

Best of luck.

## 2 Installing RTAI from scratch – the basic sequence

The fact is, installing RTAI is pretty simple, but you wouldn't guess that from some of the documentation that's around. (I hope the RTA people don't take this amiss; they've done a fantastic job, but I suspect they know the system too well to describe the process from the point of view of a relative simpleton like myself.)

So, you have a PC. Lets assume it's working, but doesn't have anything on it at all. This is what you have to do.

- Select the version of RTAI you want to run, the Linux kernel version to use, and a Linux distribution that uses that kernel. (You can choose them in any order, but it's important to end up using a consistent set.)
- Install that Linux distribution, if it's not the one you use at present.
- Download a clean, kosher, version of the Linux kernel you want from one of the Linux distribution sites.
- Download the tar file for the RTAI release.
- Unpack and uncompress the files.
- Using the patch file in the RTAI release that corresponds to the kernel source you downloaded, patch the kernel source to incorporate the RTAI modifications.
- Generate a configuration file that suits the machine you're going to run on. This is traditionally the trickiest part of building a Linux kernel, and it's also the trickiest part of building RTAI.
- Build the Linux kernel (this involves just a few 'make' commands).
- Install the Linux kernel (also just a few commands).
- Configure RTAI – this corresponds more or less to configuring the Linux kernel source, but is much easier.
- Build RTAI (just one 'make' command).
- Install RTAI.
- Re-boot.

- Test.

Put like that, it looks pretty easy. In fact, it really is the case that configuring the Linux kernel is the hardest bit. If you're used to building Linux kernels, then none of this should be scary at all. Each of the steps in this list of what has to be done corresponds to a sub-section of the big section (section 3) called "Each step in detail".

Here are the actual commands I used to install RTAI. I started by working out that RTAI-3.1 supported what was then the latest stable Linux kernel release, 2.6.8.1, and that a copy of Mandrake 10.0 (which I bought at a newsagents, the CD stuck to the cover of a Linux magazine) used Linux some version of Linux 2.6, so I figured the combination of Mandrake 10,0, Linux 2.6.8.1 and RTAI 3.1 would do the job.

With Mandrake installed, I created a directory called /home/ks/kernel/RTAI where I planned to build the system. Into that I downloaded:

| | |
|---|---|
| • rtai-3.1.tar.bz2 | from www.rtai.org |
| • linux-2.6.8.1.tar.bz2 | from www.kernel.org |

That covers the first four stages in my list of what has to be done. The remaining stages cover the configuration, building and installation of the system, and the following command sequence does that. There are three interactive parts to this, and obviously what happens there isn't shown here, but these are described in full in the section of this document that explains all the steps of this sequence. Of these, only the first of the 'make xconfig' steps – the one that configures the Linux kernel itself – is tricky.

Note that this set of commands has been split up into sections that correspond to the remaining stages in the list of what has to be done. This means that each section also corresponds to a sub-section of the big section (section 3) called "Each step in detail".

When you do this yourself, the main things that will differ, and which you have to allow for, is that you probably don't want to build your system in a directory called /home/ks/kernel/RTAI, and you probably aren't going to be using the same 2.6.8.1 kernel I used. So all the '2.6.8.1's become whatever version you're using. And the couple of '/home/ks/kernel/RTAI' instances become whatever you chose for your directory. I could have used a lot of '$version' and '$homedir' constructs, but I prefer to see the real commands someone used.

Unpack and uncompress the files (section 3.5)

```
cd /home/ks/kernel/RTAI
bunzip2 rtai-3.1.tar.bz2
bunzip2 linux-2.6.8.1.tar.bz2
tar –xf rtai-3.1.tar
tar –xf linux-2.6.8.1.tar
```

Apply the RTAI patch to the Linux source (section 3.6)

```
cd linux-2.6.8.1
patch –p1 < ../rtai-3.1/rtai-core/arch/i386/patches/hal7-2.6.8.1.patch
```

Configure the Linux kernel (section 3.7)

```
cp arch/i386/defconfig .config
make oldconfig
make xconfig
```

Build the Linux kernel (section 3.8)

```
make clean
make bzImage
make modules
```

Install the Linux kernel (section 3.9)

```
su
make modules-install
mkinitrd /boot/initrd-2.6.8.1-adeos.img 2.6.8.1-adeos
cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.8.1-adeos
cp System.map /boot/System.map-2.6.8.1-adeos
ln –s /boot/System.map-2.6.8.1-adeos /boot/System.map
cd /boot/grub
vi menu.lst
exit
```

Configure RTAI (section 3.10)

```
cd /home/ks/kernel/RTAI
mkdir rtai-build
cd rtai-build
make –f ../rtai-3.1/makefile xconfig
```

Build RTAI (section 3.11)

```
make
```

Install RTAI (section 3.12)

```
su
make install
```

Reboot (section 3.13)

```
shutdown –r now
```

Test RTAI (section 3.14)

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

Feel a bit less over-awed now?

Because configuring the Linux kernel is potentially tricky – and time-consuming, because it can take a long time to compile, especially on a slow machine, and if it goes wrong you have to reboot, change something, then try again – you may prefer to split this up a bit.

So a slightly more conservative way of doing things might be the following:

- Download the kosher version of the Linux kernel, configure it for your machine, build it, install it, and reboot.

- If this gives you a system that's running fine, you can be confident you've managed the trickiest part, the configuration, and can use the new system for the next step.

- Download the RTAI release, patch the Linux kernel, rebuild it, install the new, RTAI-patched kernel, and reboot. The patched kernel is a perfectly good kernel – it works without needing the rest of RTAI. But when you do build RTAI, the patched kernel will support it.

- This new kernel should work fine, and you can use it to configure RTAI, install it, and run the test programs that come with it.

Doing things iteratively like this takes a bit longer, but if this is all new to you taking things one small step at a time is probably worth while.

# 3 Each step in detail

This section gives more detail about the individual stages in the build. Feel free to skip the bits you know all about.

## *3.1 Selecting versions*

You have to select versions of RTAI, Linux, and a Linux distribution to use. These have to be compatible, which is the tricky bit. Each version of RTAI supports only a limited subset of Linux kernel versions, and you have to use the exact version of one of those, and you have to have the source for that kernel version *as released by the Linux kernel people* – not the version included in your Linux distribution, which may have been modified by the people who made the distribution. And you need a Linux distribution compatible with the Linux kernel you select.

It's up to you which you decide on first. If you already have a Linux distribution and you want to stick with that, then you should probably stick with the minor version of Linux it uses. Going from, for example, a 2.4 system to a 2.6 system probably needs a compiler upgrade and you may find that you need later versions of a number of other utilities as well. In general, a given distribution should come with the tools that match the kernel version it uses. Details about the tools needed for different kernel versions can be found in the Linux kernel build HOW-TO (see references).

Basically, if you want to put RTAI onto your system and have a distribution that uses a 2.4 kernel, then unless you want to upgrade your distribution you should stick with 2.4, and you'll find that most RTAI releases support a number of 2.4 versions. Since you have to rebuild your kernel from freshly downloaded source anyway, you may as well upgrade to the latest stable 2.4 version, and you should find that the latest version of RTAI supports that. (See page 15, Which Linux versions are supported by a given RTAI version?)

If you want the very latest, then download the latest stable version of RTAI, and see which is the latest stable Linux kernel version it supports. (Again, see page 15, Which Linux versions are supported by a given RTAI version?) Then if this is a 2.6 kernel (which it will be), find a Linux distribution that uses 2.6 – the latest Fedora does, as do Mandrake 10 and others - and install that first.

## 3.2 Installing the Linux distribution

The people who put together Linux distributions do a good job of documenting the installation procedure, and I'm not going to try to improve on that. However, it is worth pointing out that you need to install all the various programming development tools – compilers, etc, and – less obviously – you should install the Qt GUI tookit if this is offered as an option, since for 2.6 systems this is needed by the convenient xconfig GUI used to configure the kernel.

## 3.3 Downloading the Linux source

The source for the Linux kernel you want can be found at www.kernel.org. The home page will lead you directly to the source for the latest stable release, and to other stuff hot off the press, such as the latest candidate for the next stable release (what they call a 'pre-patch'). And it has a repository of all the source for all the previous releases, with HTTP and FTP links to them. If you follow down the pretty self-explanatory path through the http link, you get to, for example, www.kernel.org/pub/linux/kernel/v2.6, which has all the various 2.6 releases. The easiest thing is to download the complete tar file for the version you want, which you'll find in either .bz2 or .gz form. Take your pick.

So, for example, to get 2.6.8.1, I downloaded

www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.8.1.tar.bz2

After unpacking the file (see section 3.5) I had a directory called linux-2.6.8.1 which contains everything needed to build the kernel. I put the tar file and the resulting directory tree in my experimental /home/ks/kernel/RTAI directory. You can put it where you like. In fact, it's worth emphasising that almost all the building of both the Linux kernel and the RTAI modules can be done in an ordinary user's area. Putting it in a system location and working on them as root is *not* recommended – it isn't needed, and you should always minimise the work you do as root, on principle.

## 3.4 Downloading the RTAI files

The RTAI homepage is at www.rtai.org. As I write, this is very recently re-organised and still looks a little like a work-in-progress. I downloaded my copy of rtai-3.1 before the reorganisation, getting my copy from www.aero.polimi.it/RTAI, to which I was led by the earlier web page. The new web page has a 'download' page that presents a number of versions, classified as 'stable', 'testing' etc., but none of these contain the rtai-3.1.tar.bz2 file I downloaded.

This will no doubt be sorted out. In the meantime, you can try looking both in the older directory, which is still available, or on the new web page.

I put the tar file in my experimental /home/ks/kernel/RTAI directory, as I did the Linux source tar file. It doesn't matter where you put the source files, but the closer you keep your layout to the one I used, the fewer changes you'll have to make to my build sequence.

## 3.5 Upacking the files

I expanded the Linux source and extracted it with:

```
bunzip2 linux-2.6.8.1.tar.bz2
tar –xf linux-2.6.8.1.tar
```

And the same basic steps handle the RTAI compressed tar file, eg:

bunzip2 rtai-3.1.tar-bz2
tar –xf rtai-3.1.tar

and you will have two directories called, in my case, rtai-3.1 and linux-2.6.8.1. In the rtai-3.1 directory is a file called README.INSTALL. This is worth reading, but it's written for people who know what they're doing. The hand-holding, simple installation steps are listed at the end, but the beginning does rather throw you in at the deep end.

## 3.6 Applying the RTAI patch

The RTAI patch is applied just like any Linux kernel patch, and is used as the example in the background section "Patching the Linux kernel" (section 4.2). So if you need more explanation of what's going on in the line:

```
patch –p1 < ../rtai-3.1/rtai-core/arch/i386/patches/hal7-2.6.8.1.patch
```

you should look in that background section. One thing to note is that the RTAI patch, like many patches, adds additional configuration options, so once this patch is applied, any .config file that had been used for the kernel source is now out of date. Generally, just running 'make oldconfig' and taking the defaults will fix that, however.

## 3.7 Configuring the Linux kernel

The rôle of the .config file in controlling the Linux kernel configuration is explained in detail in section 4.4 on "Linux configuration files".

However, the problem is that when you download and unpack the kernel source, there isn't a .config file in it. So how do you get one in the first place?

Well, there are a number of options. One is to start completely from scratch with 'make config'. Don't. This tool just asks you about each option in turn – all twelve hundred odd – and if you make a mistake you have to start all over again.

As it happens, you do have at least one, and possibly two or more, possible .config files in your system. The one you definitely have is the default file provided with the Linux kernel source. Starting at the Linux source top level directory, for an Intel PC it's the file

```
arch/i386/defconfig
```

and on other systems you replace the 'i386' with something else. This provides a minimal set of definitions that can be used to build a minimal system.

At the other extreme, you probably have the .config file used to build the Linux distribution you're using. Most distributions include this, particularly if you've included the kernel source when you installed the distribution. (A Mandrake distribution has a set of files called /boot/config-2.6.3-4mdk, and these are the .config files it used.) Since the kernel included with a distribution has to run on just about anything, this .config file will have just about every option selected, mainly as modules to keep the size of the loaded kernel down.

Of the two, the minimal .config provided with the Linux source will build a kernel that will run something, but probably won't support everything you need. Since it doesn't know what ethernet adapter you have, there isn't much point in it supporting ethernet, so it won't include networking. On the other hand, it will compile relatively quickly.

By comparison, the comprehensive .config provided with your Linux distribution will support your system – after all, it's working now, isn't it? But it will take much, much longer to compile. Compilation time matters when you're configuring kernels, since there's usually a deal of "change something, rebuild, reboot and see if that works" involved. On a slow system, a slow build stage can be very frustrating. (I did my RTAI experiments on an old 450 MHz Pentium that was regarded as too slow for production work – which is why it was available for me to use – and it took about 30 minutes to build the minimal kernel and about three hours to build the comprehensive version.)

Whichever option you go for, the best scheme seems to me to be

1. Copy your selected configuration file over to the top level Linux directory as .config.

2. Use make oldconfig to bring your .config file up to date – there always seem to be some new options. Generally speaking, taking the default in each case is the best bet.

3. Use make xconfig (or make menuconfig on a 2.4 system) to fine tune the configuration. Save from the GUI and quit.

I didn't get all this right first time, which is one reason I felt it was worth writing this document. Given the compilation times, after a while I got fed up of using the comprehensive .config file I got from my distribution (), and moved over to trying to fix up the minimal system so it would provide the extra things I needed. After a while, I got used to finding my way around the xconfig GUI, and discovered that I needed to select the following in addition to the defaults provided in the minimal 2.6.8.1 .config file:

**Processor type and features -> Processor family.** Check a suitable processor is selected.

**Device drivers -> Networking support -> Ethernet (10 or 100Mbit).** Click this and a load of possible ethernet device drivers appear. Pick the one that matches your ethernet adapter. (If you have 1000Mbit ethernet, of course, select that instead.) In my case, I clicked on 3COM cards and then on 3c590/3c900 series.

**Device drivers -> Block devices -> Loopback device support.** You need to select this or your kernel won't support the 'mkinitrd' step in the kernel installation, because it won't support mounting a regular file as a block device.

**Loadable module support -> Enable loadable module support.** This should already be selected by default. Also select the Module unloading sub-option, and make sure the Automatic kernel module loading option is selected. Module versioning support should be off, because apparently it causes problems with RTAI (according to the RTAI installation FAQ). Allowing the forcible unloading of modules can help if you're going to be testing experimental modules.

**Adeos support.** This only appears after you've installed the RTAI patch, and if you ran through make oldconfig it should be selected by default. Just make sure it is.

**Kernel hacking.** Make sure the Compile the kernel with frame pointers option is not selected, since I've seen reports that this is also a problem for RTAI.

The boldfaced bit in each of the above is intended to represent the path through the options as presented by xconfig.

In addition, I selected the following, because I also wanted to see how well the ordinary kernel performed and had some tests that involved the real-time clock. These aren't necessary if you're doing serious real-time, because you're going to use RTAI for that, aren't you? So these are just notes for myself, really. I have seen a suggestion that RTAI has – or used to have – problems with the preemptible kernel option.

**Processor type and features -> Preemptible kernel.** Selected.
**Device drivers -> Character devices -> Enhanced Real Time Clock support.** Selected as a module – so I can replace it as part of my tests.

## *3.8 Building the Linux kernel*

Once you have a .config file, this is really easy. You need to make the main kernel file, which is kept in /boot in a compressed form and is loaded as part of the boot process. This file is called bzImage when it is built. You also need to build all the various kernel modules. It takes three 'make commands' (four on a 2.4 Linux version).

```
make dep
make clean
make bzImage
make modules
```

The 'make dep' step is not needed under 2.6. If you try it you'll be told it isn't needed any more. Indeed, under 2.6 you can just do 'make' at this point.

All of these can be performed without any privileges. You should do them logged in as a normal user, not as root. If you are using a comprehensive .config file, then making the modules takes much, much, longer than making the basic bzImage file. If you have a minimal .config file, the modules step should be quick. It may be worth piping the output to a log file, so that after you've finished your traditional cup of coffee you can look at it and see if there were any obvious problems. Actually, I find it much easier to use the 'script' command to log the entire build process – just give the command

```
script build.log
```

before you start, and everything on the terminal gets logged to the specified file. The 2.4 build outputs far more stuff than the 2.6 build. You may see some warnings about deprecated features, and even the odd pedantic gcc complaint about missing casts and so on. These are OK – at least, there's nothing you need to do about them. Generally, I've found the compilations work, and the problems come later.

## *3.9 Installing the Linux kernel*

There are two parts to this. You have to copy the relevant files – the bzImage file and the various modules - into /boot, and you have to set up your bootstrap loader to pick them up.

In principle, this is all a lot easier under 2.6 than under 2.4. Under 2.6 this is all packaged up into just a simple 'make install', which does the lot for you, including editing your bootloader file. However, I have had the last part fail on me – I ended up with a GRUB/menu.lst file that had an extraneous entry that stopped the bootloader offering me my new kernel as a boot option. The full sequence as given below works for 2.6, but it is tedious. I'd suggest trying just 'su' and 'make install', but you may want to check the bootloader file if you have problems. Copying the various files over is trivial and easily automated – so shouldn't go wrong – but automatically editing a GRUB file is evidently more error-prone.

For some reason many distributions change the name of bzImage when they put it in /boot. It doesn't matter, because all the really matters is that your bootstrap loader, usually either LILO or – more commonly now – GRUB, has a configuration file that gives the file name being used. I'm sticking with the vmlinuz name that the other installed kernels have used.

The boot process also needs an initial RAM disk image, the 'initrd' file. This has to be created, using the 'mkinitrd' command. In addition, for various diagnostic purposes, you need the current system map file, System.map, in the /boot directory. (Usually, you copy it in with the version number appended to the name and make System.map a symbolic link to it.)

In general, it's a lot easier to support multiple kernel versions if all the files put into /boot have the kernel version name appended to them. The System.map file can be made a symbolic link, and the boot loader configuration file handles the other file names.

So here are the installation stages. All of these need to be performed as root, since you need to be able to write into the /boot directory. Your default directory should be the top level of the Linux kernel source directory.

1. Change to root

   ```
   su
   ```

   and, of course, enter the root password.

2. Install the kernel modules

   ```
   make modules_install
   ```

   The modules are installed into /lib/modules/2.6.8.1-adeos. Note that the name used depends on the version string for your new kernel ('2.6.8.1-adeos' in this case), and you need to know this for the next stage. The RTAI patch has set up the new version string, so you don't necessarily know what it's going to be, but doing an 'ls /lib/modules' will show you what's been created, and from that you can see what version string is being used.

3. Create the initrd RAM disk file.

   ```
   mkinitrd /boot/initrd-2.6.8.1-adeos.img 2.6.8.1-adeos
   ```

   The second argument, the version string, allows mkinitrd to locate the files it needs to build the disk file whose name is given in the first argument. For some reason I find I keep forgetting the .img extension for this file, and I also seem incapable of typing adeoa – sorry, adeos – properly. Just check your typing. The kernel build HOW-TO file points out that some distributions, for example, SUSE, have a version of mkinitrd that uses a slightly different syntax.

4. Copy the compressed kernel bzImage file into /boot

   ```
   cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.8.1-adeos
   ```

5. Copy the new system map file into /boot and set up the correct symbolic link for it.

   ```
   cp System.map /boot/System.map-2.6.8.1-adeos
   ln –s /boot/System.map-2.6.8.1-adeos /boot/System.map
   ```

Now you need to fix the configuration file for your boot loader. Most 2.6 systems use GRUB, which is more flexible than the traditional LILO. GRUB is controlled by a configuration file called

/boot/grub/menu.lst (that's 'L' st as in 'last', not a '1' st as in 'first'). In some systems a file called grub.conf is a link to this oddly-named file. The syntax for this file is a bit obscure, but the existing entries should show you what works for your system, and cutting and pasting to add a new entry isn't that hard.

My menu.lst file looks like this:

```
timeout 10
color black/cyan yellow/cyan
default 1

title linux
kernel   (hd0,0)/vmlinuz   root=/dev/hda2   devfs=mount   acpi=ht
resume=/dev/hda3
initrd (hd0,0)/initrd.img

title linux-secure
kernel  (hd0,0)/vmlinuz-secure  root=/dev/hda2  devfs=mount  acpi=ht
resume=/dev/hda3
initrd (hd0,0)/initrd-secure.img

title failsafe
kernel (hd0,0)/vmlinuz root=/dev/hda2 failsafe acpi=ht resume=/dev/hda3
devfs=nomount
initrd (hd0,0)/initrd.img

title linux-2.6.8.1-adeos
kernel (hd0,0)/vmlinuz-2.6.8.1-adeos root=/dev/hda2 devfs=mount acpi=ht
resume=/dev/hda3
initrd (hd0,0)/initrd-2.6.8.1-adeos.img
```

Some long lines have been wrapped here, but you can see that the new entry, for the 2.6.8.1-adeos kernel, was produced by just a bit of cutting and pasting from the default linux-secure entry. Note that both the kernel file (the vmlinuz file originally created as bzImage) and the initrd file have to be specified.

If you're using LILO, you'll make the same sort of changes to the /etc/lilo.conf file. On my system this is:

```
# File generated by DrakX/drakboot
# WARNING: do not forget to run lilo after modifying this file

boot=/dev/hda
map=/boot/map
default="linux-secure"
keytable=/boot/us.klt
prompt
nowarn
timeout=100
message=/boot/message
menu-scheme=wb:bw:wb:bw
image=/boot/vmlinuz
      label="linux"
      root=/dev/hda2
      initrd=/boot/initrd.img
      append="devfs=mount acpi=ht resume=/dev/hda3"
      read-only
image=/boot/vmlinuz-secure
      label="linux-secure"
```

```
        root=/dev/hda2
        initrd=/boot/initrd-secure.img
        append="devfs=mount acpi=ht resume=/dev/hda3"
        read-only
image=/boot/vmlinuz
        label="failsafe"
        root=/dev/hda2
        initrd=/boot/initrd.img
        append="failsafe acpi=ht resume=/dev/hda3 devfs=nomount"
        read-only
image=/boot/vmlinuz-2.6.8.1-adeos
        label="2.6.8.1-adeos"
        root=/dev/hda2
        initrd=/boot/initrd-2.6.8.1-adeos.img
        append="devfs=mount acpi=ht resume=/dev/hda3"
        read-only
```

and again the 2.6.8.1-adeos entry is a cut and paste job using the default linux-secure entry provided by Mandrake. Since I don't use LILO, I can't promise that this file works. Note that editing this file isn't enough. As it says at the top, you also have to install the new kernel by running lilo:

```
/sbin/lilo
```

And having done all this, you no longer need to be root, so 'exit' from the root process and you're back where you were before giving the 'su' command.

(You now have a viable Linux kernel, albeit one that supports RTAI. You could shutdown your system now and reboot to see if it works, before moving on to setting up RTAI itself.)

## 3.10 Configuring RTAI

RTAI configuration can be done as a normal user.

RTAI is configured in a similar way to the Linux kernel itself, using a variation on 'make xconfig'. The RTAI installation instructions recommend building RTAI in a separate build directory, not the actual source directory, and this is the course I followed. So I created a directory called 'rtai-build', set my default to that, and ran 'make xconfig' in that, using the –f option to make to point it at the makefile provided in the RTAI source directory.

```
cd /home/ks/kernel/RTAI
mkdir rtai-build
cd rtai-build
make –f ../rtai-3.1/makefile xconfig
```

In the absence of a .config file for RTAI itself, defaults are taken from the rtai-core/arch/i386/defconfig file. RTAI configuration feels very much like Linux kernel configuration, and this is obviously what is intended. Most of the RTAI defaults are pretty sensible. The only exception was that the default location for the Linux kernel source – which the RTAI build needs to access – is set to the /usr/src/linux directory structure, which needs to be changed if you put the kernel source in a personal location, as I did. So the only option I changed was:

**General -> Linux source tree.** I set this to /home/ks/kernel/RTAI/linux-2.6.8.1. You have to double click on entries like this to bring up a dialogue box that will allow you to set a new string.

Then you save and quit. This takes a little while, as the RTAI system configures itself and generates all the makefiles it will need. Finally, you're ready to build RTAI.

## 3.11 Building RTAI

It really doesn't come much simpler than this.

```
make
```

That's it. Done. It's pretty quick, as well. The easiest way to clean things up if you want to repeat the RTAI build – if you've built it outside the source tree, as recommended – is just to delete everything in your build directory using '/bin/rm –rf rtai-build/*'.

## 3.12 Installing RTAI

And this is pretty easy too. You need to be root to do the installation, so:

```
su
make install
```

## 3.13 Rebooting

And now, you need to reboot to use your new system. You're still logged in as root, so, take a deep breath, and

```
shutdown –r now
```

And the system will shutdown, and reboot itself. This time, if all went well, you will find you have a new option presented by GRUB when it comes up. If you didn't make the new system the default in the GRUB configuration file, you need to get in when the GRUB options come up and select your new 'adeos' system. You can then watch it load and try to start up.

IF it goes wrong..

I found it was very easy to mistype the filenames in the GRUB configuration file, so if the load doesn't even start properly, this is a good place to start looking. If you find messages about not being able to find the 'initrd' file, or the new vmlinuz file, that's almost certainly what went wrong. That's easy to fix. You'll probably find yourself rebooting a number of times into the old, tried and trusted kernel provided by your distribution.

Eventually, you'll probably find the system boots but not everything runs. The first time I tried with a comprehensive .config file, modified only slightly (or so I thought), it took me many iterations before I got a USB mouse working. The problem is that some of the .config options interact in non-obvious ways, and I had some of the necessary items in modules and some in the kernel and it didn't work. It was this sort of experimentation that made me go to the minimal .config file provided with the Linux source.

Another easy trap is to find that you've forgotten the ethernet driver you need. In this case the system will come up but the network doesn't work.

If all else fails, go back and repeat the whole sequence from scratch, but without installing the RTAI patch, and without building and installing RTAI. That at least will tell you if the problem is connected with RTAI or not. The fewer changes you make to the basic .config file the better, as well, so you could even omit the 'make xconfig' step and just use the basic file as is (although you need the 'make oldconfig' step to make sure there are no undefined options).

### 3.14 Testing

RTAI comes with a number of test programs. One of these runs a latency test, rescheduling itself at a 10KHz rate and measuring the difference for each reschedule between the actual and expected rescheduling times. If you didn't change the default in the RTAI configuration, most of RTAI is installed in /usr/realtime, and to run the latency test you need to become root (or you need to be authorised for sudo) and execute the 'run' command for the test:

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

RTAI programs run as kernel modules. If there is any indication of a problem, it may be that there was a problem with the module version – if the kernel suspects that the module was built for a different version of the kernel, it will refuse to run it. This can happen easily when reconfiguring kernels, because it is important to build the module with the same include files as were used for the kernel actually in use, including the .config file. If you change the configuration file and rebuild the kernel without rebuilding the RTAI modules, you can get this sort of problem.

Problems loading modules are usually logged in /var/log/messages, so 'tail' this file if you have any problems to see if you can see anything relevant there,

If all goes well, you should see some latency figures that don't look very impressive until you realise they're in nanoseconds, not microseconds. Then you realise this really is a hard real-time kernel you're running. If you get to this point, well done! (And a big cheer for all the people who developed RTA for you.)

## 4 Background

This covers a load of stuff that you need to know in order to understand some of the details of what's described in this document. Some of it you probably know already, so it starts to sound familiar feel free to skip it. There may be someone out there who'll find it useful. If not, I've done a lot of typing for nothing.

### 4.1 Linux versions

The Linux kernel has a precise system of version numbers. There is a major and a minor version number. I'm running a 2.6 release, where 2 is the major and 6 is the minor number. Major version numbers don't change very often – it's only reached 2 so far, and it's been going quite a while – which means that a change in minor version number can be quite significant. There is a convention that odd minor version numbers are experimental development versions while even minor version numbers are for production use. So you probably aren't using a development version such as 2.5, unless you're involved in kernel development, and if you are you probably aren't reading this.

Even within a minor version, there are upgrades, and a third (and even a forth) sub-version number is used to indicate these, indicating patches for security and other reasons. So in fact, the version I'm running is based on 2.6.8.1, which at the time I downloaded it was the latest stable release. The kernel version is actually a string formed from the version number and other information about patches etc. For a while I was running a version called 2.6.9-rc4 which was release candidate 4 for the version that might one day become 2.6.9.

The command 'uname –r' displays the version of the current kernel. For example, on my new RTAI system, I get:

```
$ uname -r
2.6.8.1-adeos
```

where the 'adeos' was appended when the kernel source was patched to add the RTAI modifications.

## 4.2 Patching the Linux kernel

Patches are released for the Linux kernel for various reasons. The people working on kernel development issue patches that implement new features or problem solutions. Such a patch is needed to make the kernel work with RTAI.  So to install RTAI you need to know how to patch the system.

A patch is released as a .patch file (or more usually as a compressed form of a .patch file, typically a .patch.bz2 file). These files contain the differences between the kernel code that is to be patched and the resulting, patched code. The 'patch' utility, which applies a patch, checks that the files it is patching are as expected, and logs any exceptions it finds. The important thing to note is that a patch is very specific; it takes one exact version of the kernel source and changes it into a different version. If you try to apply a patch to even a slightly different version of the code, patch will notice and you will have problems. So make sure that the patch you're about to apply is for the version of the kernel source that you have. Once the patch is applied, you'll have a different version, with a different version string. As shown in the previous section, when I applied the correct RTAI patch to the Linux kernel version '2.6.8.1', the result was the code for the version '2.6.8.1-adeos'.

There are some things you need to know about the using the 'patch' command on the Linux kernel. One is that the way the patches are created means you have to run the 'patch' command from the top level Linux source directory – the one created when you untar the Linux kernel source – the one that has the .config file in it, and you have to specify the '-p1' option.  This makes it handle the file names properly. The other is that the 'patch' command isn't normally given the name of the patch file as an argument. It needs to have the patch file fed to it as standard input. (Actually, it does have a –i patchfile flag that you could use instead). Now look at the patch command in the example sequence at the start of this document:

```
cd linux-2.6.8.1
patch –p1 < ../rtai-3.1/rtai-core/arch/i386/patches/hal7-2.6.8.1.patch
```

Here, I start in the directory containing both the Linux source tree (linux-2.6.8.1) and the RTAI source tree (rtai-3.1). I go into the top level Linux source directory, and run the 'patch' command with its input coming from the patch file, deep in the RTAI directory, that matches the Linux version.

Also worth noting is that 'patch' can also take a '--dry-run' flag, which checks that the patch can be applied OK, but doesn't actually patch anything. Cautious people do this before running the command for real – if it's going to have any problems, it's better to know about it before you mess up all the source files rather than after.

## 4.3 Which Linux versions are supported by a given RTAI version?

Each RTAI release comes with patch files set up for a number of different Linux kernel versions. However, I found it hard to find anything that documents which Linux versions are supported by any given RTAI release. The one sure way to find out is to download the RTAI release you think may be the one you want, unzip and untar it, and look to see which patch files it contains.

To do this, you need to know is that the RTAI patch files are held in a directory called

```
rtai-core/arch/i386/patches
```

(If you're using a non-Intel 386 architecture, there are other supported architectures in the rtai-core/arch directory. I'm using i386 for all the examples in this document because it's the best-supported RTAI architecture, and it's the only one I have access to.)

So you can find out which Linux versions are supported by listing the files in the patch directory. For RTAI 3.1, you get:

```
$ ls rtai-3.1/rtai-core/arch/i386/patches
hal16-2.4.24.patch  hal16-2.4.26.patch  hal6-2.6.7.patch
hal16-2.4.25.patch  hal16-2.4.27.patch  hal7-2.6.7.patch
hal7-2.6.8.1.patch
```

And you might guess (correctly) that Linux versions 2.4.24 to 2.4.27 are supported, as are 2.6.7 and 2.6.8.1. The 'hal' in the 'hal16', 'hal6', 'hal7' prefixes refers to 'hardware abstraction layer', which is connected with the way RTAI manages to take control of the system while leaving Linux under the impression that it's in charge, and so the different 'hal' prefixes refer to different version of 'adeos' which is the name of the software layer that accomplishes this trick. For a little more detail, see the section on 'Adeos'.

And that's how I knew I could expect RTAI 3.1 to work with the 2.6.8.1 Linux kernel.

## *4.4 Linux configuration files*

The information in this section can be found in all sorts of places, since none of it is specific to RTAI. The point is that the whole Linux build process – in particular, what bits get included in the kernel – is controlled by a configuration file at the top level of the Linux source tree called .config. What I'm calling the 'top level of the Linux source tree' here is the directory – in my case called linux-2.6.8.1 – that is the top level of the files that are extracted from the linux tar file downloaded from www.kernel.org. This is the directory where almost all of the work involved in building the kernel takes place. And you can put it anywhere you like.

Each configurable option in the Linux kernel has a C pre-processor variable associated with it. Most options need to be set to "yes", "no" or "module", where "yes" means compile it into the basic kernel, "no" means leave it out completely, and "module" means include it in a loadable module. Some options, but not many, are numeric values or strings that provide more detailed control. Each option has a symbolic name, in upper case, whose meaning is not always obvious from the name. The .config file is a C pre-processor file with a line for each option, usually setting it to 'y' or 'm' for "yes" or "module" or a comment to the effect that the option is not set. Clearly getting this right makes a big difference. My RTAI .config file has 1,279 lines in it, which means there are a lot of options.

Once you have a .config file, there are some useful tools that can be used to modify it and fine-tune it for your system, all invoked as variants of the 'make' command:

- **make oldconfig.** This takes an existing .config file, usually one used for an earlier release of Linux on your system, and asks you about all the 'new' configuration settings – ones for which the old file doesn't have a setting, and so which presumably represent new options added in later releases. All it does is prompt you for the various undefined options one by one, with a bit of on-line help. You can't go back to a previous question without starting over again, although you can fix it up later using 'xconfig'.

- **make xconfig.** This is the rather nice GUI configuration tool provided in Linux 2.6. It lets you move around the options, which are grouped fairly conveniently, and provides on-line help when you click on them. Selecting some options brings up some others – for example, if you select one of the ethernet options, say support for 100Mbit ethernet, you find yourself presented with a bewildering array of options for drivers. This GUI is built using Qt, so you need that installed on your system.

- **make menuconfig.** Is the older, rather clunkier GUI provided in Linux 2.4. Similar to xconfig, but not so easy on the eye.

### 4.5 ADEOS

ADEOS stands for – wait for it – Adaptive Domain Environment for Operating Systems, and this is the way the latest versions of RTAI get control of the system while still managing to co-exist with Linux. I'm not getting into the details here (partly because I don't understand them), but if you like to know exactly what's going on in the software you're using, the place to look is www.gna.org/adeos. (And although you'd think that acronym couldn't possibly be thought up by more than one person, ADEOS is also used for the Advanced Earth Orbiting Satellite – it's amazing what Google can turn up.)

Still, since the RTAI patched version of Linux ends up with a version string with 'adeos' in it, it's nice to have some idea what the word means.

## 5 References

There is a wealth of information available on the Web, particularly about Linux. I found the following particularly useful, but there will be many, many other equally useful places to look that I simply never found. Somewhere out there, no doubt, is a better version of this document, and I could have saved myself a lot of trouble if I'd found it.

**The RTAI homepage.** www.rtai.org. The obvious place to look, although at the moment (Nov 2004) it's undergoing quite significant changes and not everything seems to have settled down yet. There is an FAQ on installation and configuration, but it doesn't cover a lot yet.

**The Linux Kernel Archives.** www.kernel.org. The place for your Linux kernel shopping. If they don't have it you don't want to be using it.

**Kernel build HOW-TO.** www.digitalhermit.com/linux/Kernel-Build-HOWTO.html. This is the standard HOW-TO guide to building and installing Linux kernels. This is a must-read document if you've not built a Linux kernel before.

**Linux Weekly News.** lwn.net. A weekly Web publication (subscription required for the absolutely latest hot-off-the-press news, or you can wait a week). Has an excellent kernel section.

**LinuxDevices.com** www.linuxdevices.com Geared to embedded systems, but has published some good summary articles about using Linux in real-time. The home page has a link to a series of quick reference guides, one of which is to real-time Linux.

**Captain's Universe – Programming.** www.captain.at/programming. An eclectic collection of stuff all in one site, but it has a programming section that has a number of examples showing how to do

things in Linux, with a number of RTAI-related examples. Most of the examples look a bit like my basic command sequence – not much explanation, but the important things to do are there.

**Adeos.** home.gna.org/adeos. The ADEOS home page. It doesn't contain much you need to know in order to install RTAI, but it's good to know something about the underpinnings of the system – even if it's only what 'ADEOS' stands for.

**Kernel newbies.** www.kernelnewbies.com. A site for people new to the Linux kernel. It has some useful information – in particular, I found their FAQ entry on patching the kernel very helpful.

**Google.** www.google.com. It's amazing how many times you can type a question into Google and get an answer. You can type 'RTAI kernel frame pointers' in, for example, and get a load of links to do with whether these are a problem or not. A great window onto the Web.