

RTAI/RTHAL Kickstart

Georg Schiesser, Florian Bruckner, Der Herr Hofrat
OpenTech EDV-Research GmbH, Austria
Liechtenstein Str 31, A-2130 Mistelbach, Austria
<http://www.opentech.at>

Abstract

Real Time Application Interface/Real Time Hardware Abstraction Layer - RTAI/RTHAL Kickstart session at the 5th Real Time Linux Workshop. This kick-start session introduces RTAI/RTHAL installation and first steps on a Slackware 9.1 system running RTAI 24.1.12 (Kernel 2.4.21). The basic steps should work out on other but Slackware 9.1 systems as well as far as known distribution specific differences were are noted along the way. This kick-start manual is Licensed under FDL V1.2 <http://www.gnu.org/copyleft/fdl.html>

1 Introduction

This manual is not a in depth introduction to installing and running RTAI/RTHAL but, as the name says, a kick-start only. It should guide you step by step to get you up and running on RTAI/RTHAL quickly. Although this document describes the steps for RTAI/RTHAL on a Slackware 9.1 system, steps will more or less be the same on a different distribution, and should still give you some guidance for installing RTAI/RTHAL on a Slackware 9.1 based Linux-box. Feedback, especially on trying this for other platforms, and a clean RTAI/RTHAL version or for other hardware architectures would be appreciated.

For a RTLinux kick-start see [kickstart_rtl.pdf](#).

2 Slackware 9.1 Install

- Boot from CD: < ENTER >

The boot prompt is only intended for passing additoinal kernel parameters - normally necessary if you have some non-standard hardware, also if the default `bare.i` kernel does not work, press [F2] at the boot prompt for a list of posible kernels to boot.

- boot: < ENTER >
- Enter 1 to select a keyboard:
- Keyboard map selection:

```
qwertz/de-latin1-noddeadkeys.map < OK >
```

- Keyboard test

```
1 < OK >
```

Not a very intuitive interface that requires to type in 1 to the text field before hitting < OK > - but thats Slackware...

You may now login as 'root'

Slackware login:

At this point Slackware is running a minimum system in a ramdisk so you actually are login into the Linux box as root at this point. So type in root and hit < ENTER >

2.1 Partitioning

As noted above Slackware boots into a minimum system loaded into a ramdisk - so you have the 'standard' GNU/Linux tools available for system setup. Slackware does not bother providing a 'User Friendly' wrapper to these functions, you simply use them on the command line and that ensures that you actually know what you are doing.

```
# fdisk /dev/hda
```

The number of cylinders for this disk is set to 15017.

There is nothing wrong with that, but this is larger than 1024, and could in certain setups cause problems with:

- 1) software that runs at boot time (e.g., old versions of LIL0)
- 2) booting and partitioning software from other OSs
(e.g., DOS FDISK, OS/2 FDISK)

Command (m for help):

To check the existing partition table use the 'p' command

Command (m for help): p

```
Disk /dev/hda: 123.5 GB, 123522416640 bytes
255 heads, 63 sectors/track, 15017 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1		1	103	827316	82	Linux swap
/dev/hda2	*	104	15016	119788672+	83	Linux

First we delete all partitions as this is going to be a pure Linux box. If there are partitions defined make sure you delete them in reverse order - so start with the highest numbered partition and delete one by one (in my case this was 2).

Command (m for help): d

Partition number (1-4): 2

Command (m for help): d

Selected partition 1

Command (m for help): 1

Next we create two new partitions one as are Linux filesystem (we will simply put it all in one big chunk for now) and one swap partition.

Command (m for help): n

Command action

e extended

p primary partition (1-4)

We respond with 'p' for a primary partition and then get the prompt for the partition number.

```
p
Partition number (1-4): 1
First cylinder (1-15017, default 1): <ENTER>
```

As our first partition should start at the first cylinder we simply hit enter.

```
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-15017, default 15017): +512M
```

On the first partition we are going to put the swap partition, so we request 512MB for the first partition, the '+' tells fdisk to increment 512MB starting at the current cylinder position, which is 1 in our case. We could give it a cylinder number too but then you must calculate the size your self..

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
```

```
p
Partition number (1-4): 2
First cylinder (64-15017, default 64):
Using default value 64
Last cylinder or +size or +sizeM or +sizeK (64-15017, default 15017):
Using default value 15017
```

The second partition is again a primary partition and will simply be the full remaining disk, which is offered by default.

If we now print the current partition table we see the two desired partitions, but they are both marked as Linux, we need one to be a swap partition.

```
command (m for help): p
```

```
Disk /dev/hda: 123.5 GB, 123522416640 bytes
255 heads, 63 sectors/track, 15017 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1		1	63	506016	83	Linux
/dev/hda2		64	15017	120118005	83	Linux

So the next step is to change our first partition to Linux swap with the 't' command.

```
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): L
```

0	Empty	1c	Hidden W95 FAT3	70	DiskSecure Mult	bb	Boot Wizard hid
1	FAT12	1e	Hidden W95 FAT1	75	PC/IX	be	Solaris boot
2	XENIX root	24	NEC DOS	80	Old Minix	c1	DRDOS/sec (FAT-
3	XENIX usr	39	Plan 9	81	Minix / old Lin	c4	DRDOS/sec (FAT-
4	FAT16 <32M	3c	PartitionMagic	82	Linux swap	c6	DRDOS/sec (FAT-
5	Extended	40	Venix 80286	83	Linux	c7	Syrinx
6	FAT16	41	PPC PReP Boot	84	OS/2 hidden C:	da	Non-FS data
7	HPFS/NTFS	42	SFS	85	Linux extended	db	CP/M / CTOS / .
8	AIX	4d	QNX4.x	86	NTFS volume set	de	Dell Utility
9	AIX bootable	4e	QNX4.x 2nd part	87	NTFS volume set	df	BootIt
a	OS/2 Boot Manag	4f	QNX4.x 3rd part	8e	Linux LVM	e1	DOS access

```

b W95 FAT32          50 OnTrack DM          93 Amoeba          e3 DOS R/0
c W95 FAT32 (LBA)  51 OnTrack DM6 Aux  94 Amoeba BBT      e4 SpeedStor
e W95 FAT16 (LBA)  52 CP/M              9f BSD/OS         eb BeOS fs
f W95 Ext'd (LBA)  53 OnTrack DM6 Aux a0 IBM Thinkpad hi ee EFI GPT
10 OPUS             54 OnTrackDM6        a5 FreeBSD        ef EFI (FAT-12/16/
11 Hidden FAT12     55 EZ-Drive          a6 OpenBSD        f0 Linux/PA-RISC b
12 Compaq diagnost 56 Golden Bow       a7 NeXTSTEP       f1 SpeedStor
14 Hidden FAT16 <3 5c Priam Edisk       a8 Darwin UFS     f4 SpeedStor
16 Hidden FAT16     61 SpeedStor        a9 NetBSD         f2 DOS secondary
17 Hidden HPFS/NTF 63 GNU HURD or Sys ab Darwin boot     fd Linux raid auto
18 AST SmartSleep   64 Novell Netware b7 BSDI fs         fe LANstep
1b Hidden W95 FAT3 65 Novell Netware b8 BSDI swap       ff BBT
Hex code (type L to list codes): 82
Changed system type of partition 1 to 82 (Linux swap)

```

You should then check that the partition table is correct and then call the write command to actually write the new partition table to disk. Until you type 'w' for write nothing on the disk was changed - so you can quit any time by pressing < CNTRL > <C> or 'q' at the menu.

2.2 Starting setup

Slackware has a setup program on the ramdisk that you invoke by simply typing in

```
# setup
```

we then select the key-map again - see above - and proceed on to setting up our swap disk, the partition is already created and the setup script will find it, so we just need to activate it

- SWAP SPACE DETECTED

```
/dev/hda1
```

- FORMATTING SWAP PARTITION...
- SWAP SPACE CONFIGURED

```
/dev/hda1
```

The partition we set up for the Linux filesystem needs to be formatted next, first we select the partition from the presented possibilities, which is only /dev/hda2 in our case, and hit < ENTER >, next we are asked for the method of formatting, Format is OK for almost all systems, if the hard-disk is some old disk (and not too large...) you might want to select 'Check' which will actually check each block and update the bad-blocks list if necessary.

- Select Linux installation partition:

```
/dev/hda2
```

- FORMAT PARTITION /dev/hda2

```
NOTE: This will erase all data on it.
Format < OK >
Check
No
```

We suggest using ext3 filesystem, it will not save your data if you crash the kernel with a buggy kernel module, but it is a good protection against power-failure or reset button induced problems... The inode density can be left at the suggested default value.

- SELECT FILESYSTEM FOR /dev/hda2

```
ext2
ext3 < OK >
reiserfs
```

- SELECT INODE DENSITY FOR /dev/hda2

```
4096 1 inode per 4096 bytes. < OK >
2048 1 inode per 2048 bytes.
1024 1 inode per 1024 bytes.
```

- FORMATTING...
- DONE ADDING LINUX PARTITIONS TO /etc/fstab

So now the systems partitions are set up and formatted - we are ready to fill the disk up with content. But before that we have to select a installation media, which is the CD we booted from in our case, this menu question makes sense because Slackware can also be installed starting with a floppy disk and if you have a really fast university network (does something like this actually exist ?) then NFS install may be an option. In case you booted from the CD the auto-detection will work fine, so we select [auto] and hit < OK >.

- SOURCE MEDIA SELECTION

```
1 Install from a Slackware CD or DVD < OK >
2 Install from a hard drive partition
3 Install from NFS (Network File System)
4 Install from a pre-mounted directory
```

- SCANNING FOR CD or DVD DRIVE

```
auto < OK >
manual
```

- SCANNING...

If this does not kick your CD-ROM drive up then you should try manual selection.

Once you have your source media set we go to the package selection. Slackware allows you to select each package individually, which can take a very long time, so unless you really want a minimum system using the prepackaged selections is fine and will result in a system with all tools we need for real-time. The only thing we deselect here is KDE and GNOME, simply to reduce install time and because we are not concerned with X-setup for this session. We want to show you the power of the command-line, you can learn how to play with X-Windows later :)

After de-selecting KDE and GNOME we can simply install everything and hit < OK >.

- PACKAGE SERIES SELECTION

```
NOTE: If you install without KDE and GNOME, you will only need disc 1.
```

- SELECT PROMPTING MODE

```
full < OK >
```

- Installing...

If you did not de-select KDE and GNOME then you will be prompted for the second disk, in our case this does not happen, so we would go right to the kernel selection.

- INSERT NEXT DISC

Continue < OK >

- Installing...

You should not necessarily select the hottest and most optimized kernel here, you should select the safest kernel for the system, for IDE based systems the `bare.i` from the cdrom is what you want.

- INSTALL LINUX KERNEL

```
bootdisk
cdrom < OK >
floppy
skip
```

- CHOOSE LINUX KERNEL

/cdrom/kernels/bare.i/bzImage < OK >

It is a wise thing to create a boot-disk for a development system, sooner or later you might damage the system with your first (buggy) kernel modules, and as we never make backups... a boot-disk is helpful. In this kickstart session we will skip this step though.

We are not going to bother with the modem, and for desk-top systems you probably will not need the hot-plug subsystem, but it does not hurt to enable it.

- MAKE BOOTDISK

```
Create
Skip < SKIP >
```

- MODEM CONFIGURATION

no modem < OK >

- ENABLE HOTPLUG SUBSYSTEMS AT BOOT?

< Yes >

We need a boot-loader to actually start the system on power-on, so next we configure the Linux LOader - LILO. If you know lilo and want some special options set, select 'expert' if you are a new-bee, take the 'simple' option, it will work in more or less all cases where you have a IDE based system.

Selecting frame-buffer console is important or you will not get the penguin logo in the top left hand corner of your screen...

- INSTALL LILO

```
simple < OK >
expert
skip
```

- CONFIGURE LILO TO USE FRAME BUFFER CONSOLE?

1024x768x256 < OK >

If you have a CD-burner in your system, which is quite common, then you want to set up that CD-ROM as a SCSI device via the ide-scsi emulation, so we pass the device specific module to LILO telling it to use scsi emulation for /dev/hdc in this case.

LILO is put on the Master Boot Record (MBR). After this step the Linux loader LILO is installed and the system could boot.

- OPTIONAL LILO append="kernel parameters;" LINE

```
hdc=ide-scsi < OK >
```

- SELECT LILO DESTINATION

```
Root
Floppy
MBR < OK >
```

The rest of the configuration is not that general and may be different in your case. First we set up the mouse and configure General Purpose Mouse-support - GPM which allows using the mouse in text mode.

- MOUSE CONFIGURATION

```
ps2 < OK >
bare 2 button serial mouse
ms 3 button serial mouse
```

- GPM CONFIGURATION < Yes >

The network configuration is site specific, so you need to get the infos from your network admin. The infos you will need are:

- Host name
- Domain name
- IP address
- Netmask
- Default gateway
- Domain Name Server (DNS)
- CONFIGURE NETWORK? < Yes >
- ENTER HOSTNAME

```
rtl15 < OK >
```

- ENTER DOMAINNAME for 'rtl15'

```
hofr.at < OK >
```

- SETUP IP ADDRESS FOR 'rtl15.hofr.at'

```
static IP < OK >
DHCP
loopback
```

- Fill in the
 - IP address
 - Netmask
 - Default gateway
 - Domain Name Server (DNS)
- CONFIRM SETUP COMPLETE < Yes >
- CONFIRM STARTUP SERVICES TO RUN < OK >
- CONSOLE FONT CONFIGURATION < No >

Setting up the clock: assume the clock is UTC and select your time-zone from the list.

- HARDWARE CLOCK SET TO UTC?
 - No
 - Yes < OK >

- TIMEZONE CONFIGURATION
 - Europe/Vienna < OK >

If you did not select the KDE and GNOME packages during instalation, you should not select KDE or gnome here... but there are a number of interesting and light weight window managers around that are worth giving a look.

- SELECT DEFAULT WINDOW MANAGER FOR X
 - xinitrc.kde < OK >
 - xinitrc.gnome

2.3 Final steps before reboot

Last thing the system needs before we can reboot is a root password, for this session you should set it to 'nopasswd', but in your company network or at home, make sure you have a reasonable root-password that will not be guessed easily.

- WARNING: NO ROOT PASSWORD DETECTED Would you like to set a root password? ; Yes ;
New password: nopasswd Re-enter new password: nopasswd
- SETUP COMPLETE ; OK ;
- EXIT ; OK ;

This terminates the setup program of Slackware, and we can reboot the system. Just to make sure the filesystems are cleared properly we do:

```
# umount -a
# <CTRL>-<ALT>-<DELETE>
```

...and don't forget to remove the CD-ROM or you will fall into an endless loop.

2.4 System Boot and user account

At the LILO prompt you can add boot-command-line parameters for the kernel. This is helpful for instance, if you set up X (which is in init 4) and your screen just flickers, then you simply type in 'linux init 3', and boot the system to text-mode only. for more info on available settings check the BootPrompt-HOWTO locate in /usr/doc/Linux-HOWTOs/BootPrompt-HOWTO on your Slackware 9.1 distribution.

```
boot: linux < ENTER >
```

After the boot messages scrolled by you get the login prompt:

```
rtl15 login: root
Password: nopasswd
```

We hope that the messages produced by Slackware after login - the so called fortunes - are politically correct, but we take no responsibility for these messages...

We need to add a regular user-account, if you want to use the GNU/Linux box via remote logins or send e-mail etc. you should not work as root, so lets ad a regular user account and then we are done.

```
root@rtl15:~ # adduser
  Login name for new user []: georgs
  User ID: 500
  Initial group [users]: < ENTER >
  Additional groups []: < ENTER >
  Home directory [/home/georg]: < ENTER >
  Shell [/bin/bash]: < ENTER >
  Expiry date []: < ENTER >
press ENTER to go ahead and make the account. < ENTER >
  Full Name []: Georg S
  Room Number []: < ENTER >
  Work Phone []: +43-12345
  Home Phone []: +12345
  Other []: < ENTER >
New password: nopasswd
Re-enter new password: nopasswd
Account setup complete.
```

A bit rough in style but it should get you up and running quickly :)

3 RTAI/RTHAL kernel install

```
root@rtl15:~ # mkdir /cdrom
```

In the system used for this HOWTO the cdrom was the secondary slave device on the IDE subsystem so /dev/hdb in this case - you must replace any references to /dev/hdb by what is given by your system configuration to find the device quickly - type in the following:

```
root@rtl15:~# dmesg | grep hd

hda: IC35L120AVV207-0, ATA DISK drive
hdb: LITE-ON COMBO LTC-48161H, ATAPI CD/DVD-ROM drive
hda: attached ide-disk driver.
hda: host protected area => 1
hda: 241254720 sectors (123522 MB) w/1821KiB Cache, CHS=15017/255/63
  hda: hda1 hda2
...
```

This tells us that the CDROM is attached as hdb.

```

root@rtl15:~ # mount -t iso9660 /dev/hdb /cdrom
root@rtl15:~ # cd /usr/src
root@rtl15:/usr/src # rm linux
root@rtl15:/usr/src # cp /cdrom/kernel/linux-2.4.21.tar.bz2 ./
root@rtl15:/usr/src # tar -xjf linux-2.4.21.tar.bz2

root@rtl15:/usr/src # cp /cdrom/rtai-24.1.12.tgz ./

```

Before unpacking a package - check where it would unpack so that you don't dump a package over existing data..

```

root@rtl15:/usr/src# tar -tzf rtai-24.1.12.tgz
rtai-24.1.12/
rtai-24.1.12/ee
rtai-24.1.12/tbx/
rtai-24.1.12/tbx/rtai_tbx_lxrt.c
rtai-24.1.12/tbx/Makefile
...

```

As we don't have a rtai-24.1.12 directory yet this location is fine. we can unpack it.

```

root@rtl15:/usr/src # tar -xzf rtai-24.1.12.tgz
root@rtl15:/usr/src # ls rtai-24.1.12/patches/
...
patch-2.4.16-rthal5f          patch-2.4.20-rthal5g
patch-2.4.17-rthal5f          patch-2.4.21-rthal5g

```

The patch-2.4.21-rthal5g is the one we want here.

Check the patch before you applied it - especially if you have a number of patches already applied its very irritating if it fails and actually modified files - one can recover from a broken patch (see man patch), but it's best to dry-run a patch first.

Now that we know the RTAI release number and patch version we should name our Linux kernel appropriately so its easy to indentify:

```

root@rtl15:/usr/src # mv linux-2.4.21 linux-2.4.21-rthal5g
root@rtl15:/usr/src # ln -s linux-2.4.21-rthal5g/ linux
root@rtl15:/usr/src # cd linux
root@rtl15:/usr/src/linux # patch -p1 --dry-run < ../rtai-24.1.12/patches/patch-2.4.21-rthal5g

```

If the dry-run was successful the next step is to actually apply the patch

```

root@rtl15:/usr/src/linux # patch -p1 < ../rtai-24.1.12/patches/patch-2.4.21-rthal5g

```

The kernel source tree is now set up for RTAI/RTHAL what needs to be done is to configure it - on Slackware 9.1 you can find the distributions config file in /boot, and on SuSE systems you can copy it from /proc/config.gz - both of these config files result in extremely large compile times though as the distributions by default compiles about every module that linux provides, which makes sense for a distribution but wastes time in our case - preferably you configure the kernel to what you really need.

The kernel configuration tool loads the kernel release default configuration which is quite minimal - so we must enable all modules and capabilities we need explicitly naturally this means you need to know what hardware you have in your system so check the CPU type by doing

```

root@rtl15:/usr/src/linux # cat /proc/cpuinfo
...
vendor_id      : GenuineIntel
cpu family     : 5
model          : 4
model name     : Pentium MMX
...

```

The lines containing the vendor_id, cpu family model and model name are the ones you need for selecting the proper CPU, the lines:

```
...
fpu                : yes
fpu_exception     : yes
cpuid level       : 1
wp                : yes
flags              : fpu vme de pse tsc msr mce cx8 mmx
...
```

Tell you what hardware features your CPU provides - in this case we can see that a timestamp counter (TSC) is available which is important for real-time systems and a hardware floating point unit (FPU) is also available so we don't need to enable floating point emulation.

For further information on the present hardware you can check the output of `lspci -v` - which will list all PCI devices in the system, which you probably want to have configured and thus need to select the modules for it:

```
root@rtl15:/etc # lspci
...
00:0b.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139/8139C/8139C+ (rev 10)
...
```

This shows us that we must select a realtek 8139 module in the network devices. If unclear which of the possible modules to select, it helps to check the help page of the individual networking modules.

```
root@rtl15:/usr/src/linux # make menuconfig

Processor type and features --->
(Pentium-III/Celeron(Coppermine)) Processor family
...
```

Make sure this really fits your CPU is you run into problems select a generic type like 586/K5/5x86/6x86/6x86MX

```
a
Processor type and features --->
...
[ ] Unsynced TSC support
--- CONFIG_RTHAL must be yes
[*] Real-Time Hardware Abstraction Layer
```

This config option enable RTHAL, other than that nothing in the kernel config is really RTAI/RTHAL specific. As this is a common problem we note it here - make sure to select the filesystem that you configured your base-OS for - if you selected ext3 then make sure you put it into the kernel ! before configuring your kernel use `lsmod` to check what your current system configuration needs.

```
File systems --->
<*> Ext3 journalling file system support          x x
[*] JBD (ext3) debugging support                  x x
```

And as noted above, in the case of the test-system we used to write up this HOWTO `lspci` found a realtek 8139 so we select that as well:

```
Network device support --->
Ethernet (10 or 100Mbit) --->
[*] EISA, VLB, PCI and on board controllers
...
<M> RealTek RTL-8139 PCI Fast Ethernet Adapter
...
```

The rest of the config should fit your hardware needs (USB,SCSI etc.) but for IDE based X86 systems you will generally be able to leave the rest unchanged for the beginning in any case the rest is not RTAI specific.

```
root@rtl15:/usr/src/linux # cp .config config_rthal5g
```

Save your configuration to a file that make mrproper will not remove - all files starting with .config will be deleted so .config.backup is a bad choice. If you change anything in the kernel config or add any patches then do a

```
root@rtl15:/usr/src/linux # make mrproper
```

As the first step - in our case we don't need this though as we are starting out with a newly unpacked kernel source tree. So we just run the commands to resolve dependencies and build the modules and kernel as well as copying it all to the proper location.

```
root@rtl15:/usr/src/linux # make dep
root@rtl15:/usr/src/linux # make modules
root@rtl15:/usr/src/linux # make modules_install
root@rtl15:/usr/src/linux # make bzlilo
...
cp /usr/src/linux-2.4.21-rthal5g/System.map /
if [ -x /sbin/lilo ]; then /sbin/lilo; else /etc/lilo/install; fi
Added Linux *
make[1]: Leaving directory '/usr/src/linux-2.4.21-rthal5g/arch/i386/boot'
```

The new kernel was copied to /vmlinuz and the modules were place into /lib/modules/2.4.21-rthal5g/ - to boot your new kernel edit /etc/lilo conf to add the new kernel entry. note that it depends on the kernels INSTALL_PATH= where it is put so in the case of the vanilla kernel the new kernel ends up in in /vmlinuz not /boot/vmlinuz (like with adeos which patches the top level Makefile to set INSTALL_PATH=/boot).

```
root@rtl15:/usr/src/linux # cd /etc
root@rtl15:/etc # vi lilo.conf
```

at the beginning of the lilo.conf in Slackware 9.1 you can find the lines

```
# VESA framebuffer console @ 1024x768x256
vga = 773
# Normal VGA console
# vga = normal
```

These should be changed to:

```
# VESA framebuffer console @ 1024x768x256
# vga = 773
# Normal VGA console
vga = normal
```

Note that the exact appearance may vary in other distributions - but the changes required are the same - these changes are necessary unless you want to configure frame-buffer support into the kernel - as this leads to some problems, especially with embedded boards, we recommend you set vga = normal unless you know exactly what this is about.

```
# End LIL0 global section
# Linux bootable partition config begins
image = /boot/vmlinuz
    root = /dev/hda2
    label = Linux
    read-only
```

Copy these 4 last lines and edit them so you end up with, as make bzilo will put the new kernel into /boot/vmlinuz we boot the original Slackware kernel by putting the image=/boot/vmlinuz-ide-2.4.22 line into the first boot selection item.

```
image = /boot/vmlinuz-ide-2.4.22
  root = /dev/hda2
  label = Linux
  read-only
image = /vmlinuz
  root = /dev/hda2
  label = rtai
  read-only
```

This will leave the default kernel set to the original distribution kernel and allow you to boot the RTAI patched kernel at the lilo prompt. Note that the naming of the kernels is distribution specific and some distributions put the new kernel in /vmlinuz not /boot/vmlinuz.

```
root@rtl15:/etc # lilo
Added Linux *
Added rtai
```

Lilo allows you to select the kernel that will be booted on the next reboot with the -R flag - this only changes the selected kernel for the boot but does not permanently change anything so if it fails you just can reboot into the non-rtai distribution kernel.

```
root@rtl15:/etc # lilo -R rtai
```

now reboot the new kernel !

```
root@rtl15:/etc # reboot
```

4 Booting the RTAI system

After the system is back - we first check if our RTAI patched kernel is actually running - actually Slackware 9.1 tells you at login what you are running... but any way.:

```
Last login: Tue Nov  4 17:02:04 2003 from piglet.hofr.at
Linux 2.4.21-rthal5.
root@rtl15:~# uname -a
Linux rtl15 2.4.21-rthal5 #2 Tue Nov 4 17:03:13 CET 2003 i686 unknown unknown GNU/Linux
```

If you want to see details then use:

```
root@rtl15:~ # dmesg | more
```

Paging through the boot output you should check for

```
Linux version 2.4.21-rthal5 (root@rtl15) (gcc version 3.2.3) #2 Tue Nov 4 17:03:
13 CET 2003
BIOS-provided physical RAM map:
...
```

Which tells you that this really is the new kernel you compiled :) other than that there is no trace of rtai in the systems boot process.

So we are set for actually using RTAI now. If you have any problems with RTAI or want more information on what RTAI supports, subscribe to the RTAI mailing list at <http://www.aero.polimi.it/rtai/contact/index.html>. There also is a bug report page here if you find a bug in RTAI.

It is advisable to check the RTAI mailing list archive before posting questions... the mailing list archive is found at: <https://mail.rtai.org/pipermail/rtai/>

Mount the proceedings CD with the command:

```
root@rtl15:~# mount -t iso9660 /dev/hdb /cdrom
```

5 Building RTAI

We already unpacked RTAI to get the patches so the build steps are

- Configuration
- Compilation
- Load RTAI core modules

5.1 Configuring RTAI

```
root@rtl15:/usr/src/rtai-24.1.12 # make menuconfig
sh ./configure --menu
Enter location of Linux source tree [/lib/modules/2.4.21-rthal5g/build]: <ENTER>
```

Next you are presented the config menu of RTAI

Prompt for experimental code is something you should turn on as there are essential features you will otherwise miss - not sure if all of this is really that experimental or it just never got its status changed (i.e. rtnet).

```
Code maturity level options --->
[*] Prompt for experimental code
```

Turn on some test codes that are usable to perform a preliminary check on the system - so go into:

```
Tests and Examples --->
[*] Compile examples in kernel space
[*] Compile examples in user space
[*] Compile tests
```

The rest can be left at the default settings for now.

Scheduler will default to UP scheduler.

Lxrt module will default to old LXRT.

```
NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE
NOTICE                                                                    NOTICE
NOTICE Run:                                                                NOTICE
NOTICE   ./setsched [up | mup | smp | smpapic | smp8254 | newlxrt] NOTICE
NOTICE to set your scheduler of choice, after you have done 'make'. NOTICE
NOTICE                                                                    NOTICE
NOTICE It is compulsory to do it always as now it mates the chosen NOTICE
NOTICE scheduler to the appropriate user space support,                   NOTICE
NOTICE i.e. LXRT/NEWLXRT.                                                 NOTICE
NOTICE                                                                    NOTICE
NOTICE If nothing is done you will have the combination of the UP NOTICE
NOTICE scheduler and LXRT set for you.                                     NOTICE
NOTICE                                                                    NOTICE
NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE_NOTICE
```

Nothing to add really - we will use the defaults UP and LXRT for now but you should be aware of the fact that this is not the latest development.

5.2 Compiling RTAI

Not too much to say about the compile step it just

```
root@rtl15:/usr/src/rtai-24.1.12 # make dep
root@rtl15:/usr/src/rtai-24.1.12 # make
```

There is a `make install` as well which we will execute to create the device files `/dev/rtf*` - but we will continue work in the source tree.

```
root@rtl15:/usr/src/rtai-24.1.12 # make install
```

```
...
```

WARNING:

```
You are missing the line 'prune rtaisyms' in /etc/modules.conf,
which means that each time depmod is run, it will print the warning:
  depmod: /lib/modules/2.4.21-rthal5g/rtaisyms is not an ELF file
```

So we do what we were told...

```
root@rtl15:/usr/src/rtai-24.1.12 # vi /etc/modules.conf
```

Add the line in, note that on Slackware 9.1 `/etc/modules.conf` seems to be empty on default installations - at least during testing of this kickstart manual it was, so don't be surprised if you find it empty

```
prune rtaisyms
```

5.3 Loading the RT core modules

Before loading a kernel module you should sync your disk just do make sure that no data is lost if the system hangs.

```
root@rtl15:/usr/src/rtai-24.1.12 # sync
```

The RTAI modules can either be loaded one by one or you can use the `ldmod` script in the top level `rtai` directory for convenience as it has the right order set.

```
root@rtl15:/usr/src/rtai-24.1.12 # ./ldmod
```

checking with `lsmod` you should get the following output:

```
root@rtl15:/usr/src/rtai-24.1.12 # lsmod
Module                Size Used by    Not tainted
rtai_sched            45140  0
rtai_fifos            16424  0
rtai                  10796  1 [rtai_sched rtai_fifos]
```

Now we check the output of `dmesg` - this should show us that `rtai` has been successfully started up by performing its timing calibration for internal use.

```
root@rtl15:/usr/src/rtai-24.1.12 # dmesg
```

```
...
```

```
==== RT memory manager v1.3 Loaded. ====
```

```
***** STARTING THE UP REAL TIME SCHEDULER WITH LINUX *****
***** FP SUPPORT AND READY FOR A PERIODIC TIMER *****
***<> LINUX TICK AT 100 (HZ) <>***
***<> CALIBRATED CPU FREQUENCY 2002474000 (HZ) <>***
***<> CALIBRATED TIMER-INTERRUPT-TO-SCHEDULER LATENCY 2689 (ns) <>***
***<> CALIBRATED ONE SHOT SETUP TIME 2010 (ns) <>***
```

You should check that the values RTAI is reporting fit your hardware - this will generally be the case if your hardware is not really broken, if the values are totally wrong then the box is probably simply not suited for real-time - if it does happen - don't forget to drop the RTAI developers a e-mail report of the problematic platform !

6 Testing RTAI and Running RTAI applications

The first things we do with the new system is run some very simple programs - fist some system test and then a 'hello world' in real-time. It should be noted that the hello.c example is not part of the official RTAI distribution - it was added here to provide a simple example for the kickstart session, if you don't have access to the Proceedings CD of the 5th Real Time Linux Workshop and would like a copy, contact the Real Time Linux Foundation at www.realtimelinuxfoundation.org.

In the framework of this kickstart session we can't give an in depth intro to RTAI programming - but we want to show you the first steps. RTAI comes with quite a few examples (unfortunately not all that well documented), that can be used as a bases for working your way into RTAI. Also consult the RTAI_Manual and other documents available in the Documentation subdirectory of the RTAI distribution.

6.1 test_suite

To run these example programs and verify that the system actually works, lets run the RTAI test suite to validates the functionality of all components first.

To run this test suite

```
root@rtl15:/usr/src/rtai-24.1.12 # cd examples
root@rtl15:/usr/src/rtai-24.1.12/examples# ./test_suite
```

should launch a script that will prompt you for a number of different test programs all located beneath the examples directory. Basically just follow the instructions given, if all tests pass without errors you can assume that your system is up and running properly.

If any of the tests do fail on you - drop a note to the RTAI mailing list, reporting errors is just as important for developers as hearing success stories. If you report an error or problem to the mailing list please include:

- Distribution you are using - Slackware 9.1
- Hardware platform - X86
- Kernel version - 2.4.21
- Kernel patch - patch-2.4.21-rthal5g
- RTAI version - 24.1.12
- Application that failed - examples/test_suite

In addition to the above information try to give a verbose error description and especially describe how it can be reproduced !

For RTAI there also is a web-interface for reporting bugs, but this should only be used if you are shure you have a bug and not for getting help on problems you might have. The bug-report interface is to be found at: <http://www.aero.polimi.it/rtai/contact/index.html> -> [Report Bug].

6.2 hello.o

```
root@rtl15:/usr/src/rtai-24.1.12/examples # cd hello
```

This examples subdirectory Makefile and module code can serve as a prototype for the first steps with your own RTAI code, also the mailing list of the RTAI users found at www.rtai.org is the place to ask any questions on RTAI.

This is a trivial RTAI module - just launch on periodic thread and let it print a message every time it wakes up - this is about as simple as it can get. This framework all though simple provides a lot of what one will need for real rt-processing, just that the `rt_printk` needs to be replaced by what every you need to do in real-time.

We will introduce this example in a bit more detail, for full coverage of the RTAI API see the RTAI manual, to be found in the RTAI distribution in the Documentation directory `Documentation/RTAI_Manual.pdf`.

6.2.1 declarations and header files

The below set of header files is about the minimum possible, `module.h` is from the Linux kernel and not RTAI related, it provides all the module macros like `MODULE_LICENSE`, you need it for every kernel module. `rtai.h` is a must for any RTAI kernel module and in most cases you will also need `rtai_sched.h` which contains all the rt-task related function prototypes.

```
/* linux kernel header files we always need */
#include <linux/module.h>
```

```
/* RTAI specific header files we need */
#include <rtai.h>
#include <rtai_sched.h>
```

Your kernel modules should have a license set, preferably GPL, the author noted with an e-mail address so users can flame you if it fails, and a short description that can be queried with the module utilities.

```
/* make it GPL */
MODULE_DESCRIPTION("Hello world in real-time");
MODULE_AUTHOR("Der Herr Hofrat,<der.herr@hofr.at>");
MODULE_LICENSE("GPL");
```

With the above entries you can call `modinfo` on `hello.o`

```
root@rtl15:/usr/src/rtai-24.1.12/examples/hello# modinfo hello.o
filename:    hello.o
description: "Hello world in real-time"
author:      "Der Herr Hofrat,<der.herr@hofr.at>"
license:     "GPL"
```

The rest are some global variables we need, noteworthy is the `EXPORT_NO_SYMBOLS`; which is also from `module.h` and makes all symbols local to this module, so there are no conflicts with other modules, this generally is a good idea if you use generic variable names like 'period' .

The `#define ONE_SHOT` instructs RTAI to use one-shot clock mode and not not periodic mode (see `init_module` below).

```
int period = 100000;
int stack_size = 3000;
```

```
EXPORT_NO_SYMBOLS;
```

```
#define ONE_SHOT
```

```
static RT_TASK thread;
```

6.2.2 The rt-thread

The actual rt-task is not too wild, but this is the basic setup you will find even for complex tasks. An endless loop, executing the application specific code, in our case just a `rt_printf`, and putting the task to sleep until the timer expires and wakes up the task.

```
static void fun(int t)
{
    while(1) {
        rt_printk("I'm here, my arg is %d\n",t);
        rt_task_wait_period();
    }
}
```

6.2.3 init_module

In `init_module` all application specific resources must be set up, the kernel set up the default module related resources for us and then calls our `init_module` function.

- So we set up one-shot mode
- Then initializes a the rt-task, called `thread` in this example.
- Assign the timer to a specific CPU, on a uni-processor box this is kind of useless on an SMP system this lets you distribute tasks among available CPUs.
- Next we mark the task for periodic execution, note that this periodic execution is done by reprogramming the one-shot timer every period.
- Finally we mark the thread runnable on CPU 0 and now the task is actually running.

```
int init_module(void)
{
    RTIME tick_period;
    RTIME now;

#ifdef ONE_SHOT
    rt_set_oneshot_mode();
#endif
    rt_task_init(&thread, fun, 0, stack_size, 0, 0, 0);
    tick_period = start_rt_timer(nano2count(period));
    rt_assign_irq_to_cpu(TIMER_8254_IRQ, 0);
    now = rt_get_time() + tick_period;
    rt_task_make_periodic(&thread, now + tick_period, tick_period);
    rt_set_runnable_on_cpus(&thread, 0);
    return 0;
}
```

Now that we went through the initialization and the task code, lets compile and load it.

```
root@rtl15:/usr/src/rtai-24.1.12 # cd examples/hello
root@rtl15:/usr/src/rtai-24.1.12/examples/hello # make
root@rtl15:/usr/src/rtai-24.1.12/examples/hello# insmod hello
```

Note that `init_module` must return with 0 or the kernel assumes a failure and issues an error on `insmod` of the module.

To view the output of the `hello.o` kernel modules use `dmesg`

```
root@rtl15:/usr/src/rtai-24.1.12/examples/hello# dmesg
I'm here, my arg is 0
I'm here, my arg is 0
I'm here, my arg is 0
I'm here, my arg is 0
I'm here, my arg is 0
...
```

If you see this then the module is running properly !

6.2.4 cleanup_module

In `cleanup_module` all resources that the application allocated in `init_module` need to be released again. First we return the interrupt that was assigned to cpu 0 back to symmetric interrupt management, that is all cpus in the system will receive this interrupt again for processing not only one specific cpu, naturally this is irrelevant for uni-processor systems.

Then we stop the `rt-tasks` timer, resetting it to periodic mode.

Finally we delete the `rt-task`.

```
void cleanup_module(void)
{
    rt_reset_irq_to_sym_mode(TIMER_8254_IRQ);
    stop_rt_timer();
    rt_task_delete(&thread);
}
```

After this module specific cleanup function that was called when we do:

```
root@rtl15:/usr/src/rtai-24.1.12/examples/hello# rmmod hello
```

the kernel calls a few additional maintenance functions to clean up module related resources in the kernel and this terminates our `hello.o` kernel module and this kickstart session.

Feedback/comments/suggestions to <der.herrhofr.at>, flames to `/dev/null`.

References

- [1] [RTAI on the web] Paolo Mantegazza et al. <http://www.aero.polimi.it/rtai/team/index.html>
RTAI Download Site, <http://www.aero.polimi.it/RTAI/>
- [2] [RTAI API] E. Bianchi, L. Dozio, P. Mantegazza, *A Hard Real Time support for LINUX*, Dipartimento di Ingegneria Aerospaziale Politecnico di Milano, Part of the official RTAI distribution, [Documents/RTAIManual.pdf](#)
- [3] [Embedded RT-Guide] Harman Bruyninckx, *Real-Time and Embedded Guide*, K.U. Leuven, Mechanical Engineering people.mech.kuleuven.ac.be/bruyninc/rthowto/
- [4] [Multiboot howto] G. Schiesser, F. Bruckner, A. Staub, N Mc Guire, *Multiboot Howto*, OpenTech EDV-Research GmbH, 2003, <http://www.opentech.at/howtos/multiboot-howto/html/index.html>