RTLinux Kickstart Session

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

Abstract

RTLinux Kickstart session at the 5th Real Time Linux Workshop. This kick-start session introduces RTLinux/GPL instalation and first steps on a Slackware 9.1 system running RTLinux-3.2-pre2 (Kernel 2.4.21-rtl3.2-pre3). 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 RTLinux/GPL but, as the name says, a kick-start only. It should guide you step by step to get you up and running on RTLinux/GPL quickly. Although this document describes the steps for RTLinux/GPL on a Slackware 9.1 system, steps will more or less be the same on a different distribution, for a RTAI kick-start refer to the kickstart_rtai.pdf. Feedback, especially on trying this for other platforms.
The basic steps should work out on other systems as well, see the install_howto.html on the proceedings CD for step-by-step guide for SuSE 9.0 and RedHat 9.0 systems.

2 Slackware 9.1 Install

- Boot from CD: < ENTER >

The boot prompt is only intended for passing additional kernel parameters - norm ally 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 possible kernels to boot.

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

```
qwertz/de-latin1-nodeadkeys.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 LILO)
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.

```
ommand (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/O
```

```
 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 all ready 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 formated next, first we select the partition from the presented possibilities, which is only /dev/hda2 in our case, and hit < ENTER >, next we are ask for the method of formating, 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 formated - we are ready to fill the disk up with content. But before that we have to select a instalation 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-detecton 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 RTLinux/GPL kernel install

Mount the proceedings CD
As the default location to attache the cdrom is /mnt/cdrom (see /etc/fstab for the default on your system), you can use the command

```
root@rtl15:~ # mount /mnt/cdrom
```

and all necessary informations would be extracted from /etc/fstab - as we want to show you as much of what is happening beneath the covers - we will use the explicit mount command and create a mount point first.

```
root@rtl15:~ # mkdir /cdrom
root@rtl15:~ # mount -t iso9660 /dev/hdb /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.
Now copy and unpack the vanilla kernel linux-2.4.21 from Proceedings CD

```
root@rtl15:~ # cp /cdrom/kernel/linux-2.4.21.tar.bz2 /usr/src/
root@rtl15:~ # cd /usr/src/
root@rtl15:/usr/src # tar -xjf linux-2.4.21.tar.bz2
root@rtl15:/usr/src # mv linux linux-2.4.21-rtl3.2
root@rtl15:/usr/src # ln -s linux-2.4.21-rtl3.2 linux
```

Note that older tar versions use `-tIf and -xIf`.
Copy RTLinux from the Proceedings CD and unpack it

```
root@rtl15:/usr/src # cp /crom/rtlinux-3.2-pre3.tar.bz2 ./
root@rtl15:/usr/src # tar -xjf rtlinux-3.2-pre3.tar.bz2
root@rtl15:/usr/src # ln -s rtlinux-3.2-pre3 rtlinux
```

3.1 Patch kernel

Decompress the kernel patch

```
root@rtl15:/usr/src # cd rtlinux/patches
root@rtl15:/usr/src/rtlinux/patches # bunzip2 kernel_patch-2.4.21-rtl3.2-pre3.bz2
```

First test the kernel patch to see if it applies properly

```
root@rtl15:/usr/src/rtlinux/patchs # cd /usr/src/linux
root@rtl15:/usr/src/linux # patch -p1 --dry-run \
    < /usr/src/rtlinux/patches/kernel_patch-2.4.21-rtl3.2-pre3
```

If all goes well (sure it does...) patch the kernel now

```
root@rtl15:/usr/src/linux # patch -p1 \
    < /usr/src/rtlinux/patches/kernel_patch-2.4.21-rtl3.2-pre3
```

3.2 Configure RTLinux/GPL kernel

check with `lsmod` what essential kernel modules we need in the SuSE install (forget sound modules...) check network modules and peripherals that are essential. you can also get the config file SuSE used from the `/proc` directory (`/proc/config.gz`), but you need to check this config simply copying it may lead to problems (i.e. APM enabled...) .

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

Code Maturity Level Options
 [*] Prompt for development and/or Incomplete code/drivers

Loadable Module Support
 [ ] Set Version Information on all module symbols

Processor Type and feature
```

```
 Select EXACTLY your CPU or a generic low-end CPU (check "cat /proc/cpuinfo")

Filesystem
 [*] Reiserfs support
 [*] /dev file system support (EXPERIMENTAL)
```

save and exit - Note the reiserfs is needed because SuSE-8.0 installed the basic system on a reiserfs partition - other distributions prefer other filesystems, check in `/etc/fstab` what filesystems you will need on the system.

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

this saves the config in a way that will not be deleted by `make mrproper`.

3.3 compile and install kernel

```
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
...
a half a million lines of confusing output later...
cp /usr/src/linux-2.4.21-rtl3.2/System.map /
if [ -x /sbin/lilo ]; then /sbin/lilo; else /etc/lilo/install; fi
Added Linux *
make[1]: Leaving directory `/usr/src/linux-2.4.18-rtl3.2/arch/i386/boot'
root@rtl15:/usr/src/linux #
```

The kernel was copied to `/vmlinuz` and the modules are in /lib/modules/2.4.21-rtl3.2-pre3. We now need to edit /etc/lilo conf add entry for rtlinux. 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).

```
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 LILO 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 bzlilo 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 = /boot/vmlinuz
  root = /dev/hda2
  label = rtlinux
  append = "ide=nodma apm=off acpi=off"
  read-only
```

This will leave the default kernel set to the original distribution kernel and allow you to boot the patched RTLinux 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. Note the append line inserted - this turns off DMA for the ide discs and disable power management, generally this is a good idea for real-time systems. Note also that the very carful setting of `ide=nodma` is not a requirement of RTLinux, where as `apm=off` and `acpi=off` is a requirement if you want to guarantee hard-realtime performance.
Next we need to install the new boot-loader configuration to the disk by running lilo.

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

this should run without any errors and show you the rtlinux image.
Now we can tell the system to boot rtlinux on the next reboot - this will not permanently change the boot kernel - so by default non-rt Linux will be booted and only if we select rtlinux at the boot-prompt or by running `lilo -R` rtlinux will rtlinux boot.

```
root@rtl15:/etc # lilo -R rtlinux
root@rtl15:/etc # reboot
```

After the system comes up again - login as root. check what we have running

```
root@rtl15:~ # uname -a
Linux linux 2.4.21-rtl3.2-pre3 #5 Sun Nov 2 23:12:18 PST 2003 i686 unknown
```

4 RTLinux

RTLinux is installed from sources on the Proceedings CD, no rpm's for RTLinux around. The procedure here applies not only to the rtlinux-3.2-pre3 version but is more or less identical for other versions. If you ever run into a problem of a module or a system behaving very strange, then pleas verify the strange behavior on a clean instalation as described here, often strange behavior is due to accumulating changes and build procedures for custom modules not being clean... For question pertaining to the basic setup of RTLinux you can also contact the community via the rtlinux mailing list at `www2.fsmalbs.com/mailman/listinfo.cgi/rtl`. For the latest developments check the RTLinux/GPL developers site at `http://www.rtlinux-gpl.org`.

4.1 configure/compile rtlinux

```
root@rtl15:/etc # cd /usr/src/rtlinux
root@rtl15:/usr/src/rtlinux # make menuconfig

  Support option --->

  [*] Posix standard I/O
  [ ] POSIX Priority protection
  [*] Dev mem support
  [*] Enable debugging
  [*] rtl_printf uses printk
  [ ] Nolinux support
  [*] POSIX Signals
  [*] POSIX Timers (NEW)
  Message queue constants  --->
  [ ] RTLinux tracer support (experimental)
  [ ] Userspace Real Time
  [*] Floating Point Support
  [*] RTLinux V1 API support
  [*] RTLinux Debugger
  [ ] Synchronized clock support
```

lets leave it all defaults (shown above) for now, also the driver section can be left as it is - Save and exit menuconfig.

```
root@rtl15:/usr/src/rtlinux # make dep
root@rtl15:/usr/src/rtlinux # make 2>&1 | tee build.log
root@rtl15:/usr/src/rtlinux # make devices
```

The command `make 2$>$\&1  tee build.log`— records the entire compiler output into the file build.log, so if anything goes wrong this can help you, and also help when you report errors to the mailing list.
The make devices is only necessary for the first instalation - this creates the rtlinux specific device files in `/dev/`.

4.2 Check your instalation

The regression test performs a number of sanity checks - it will not tell you if the setup is suited for hard realtime applications, it will basically tell you that the instalation worked and that rtlinux will not crash your box ;)

```
root@rtl15:/usr/src/rtlinux # ./scripts/regression.sh
```

the regressions script should ONLY return [OK], if you get anything else pleas let the community know .
After the script terminated all rtlinux modules are unloaded.
now launch rtlinux

```
root@rtl15:/usr/src/rtlinux # ./scripts/insrtl
root@rtl15:/usr/src/rtlinux # lsmod
```

Check if the modules are loaded - it should return something like:

```
root@rtl15:/usr/src/rtlinux # dmesg -c
```

Clear the kernel message ring buffer so that we can see what messages popped up after we launched rtlinux examples.
Lets start with a very simple example - "hello World" in hard-realtime.

4.3 hello.o

```
root@rtl15:/usr/src/rtlinux # cd examples/hello
root@rtl15:/usr/src/rtlinux/examples/hello # sync ; insmod hello.o
```

Why do we do `sync ; insmod hello.o`. If you load a module that you are playing with and you made a mistake your system can crash fairly easily, as the filesystem may be in a inconsistent state at this point it could loose data or even be damaged (depending on how wildly you crash your system) so it is a good habit to sync your disk before loading a kernel module as this reduces the probability of loosing data considerably.

```
root@rtl15:/usr/src/rtlinux/examples/hello # dmesg
```

check the messages that the `hello.o` module is generating with `dmesg`. To stop our realtime "hello World" we remove the `hello.o` module and clear the kernel message buffer.

```
root@rtl15:/usr/src/rtlinux/examples/hello # rmmod hello
root@rtl15:/usr/src/rtlinux/examples/hello # dmesg -c
```

4.4 rt_process.o

Now to a more usable example - `rt_process.o`. This kernel module measure the scheduling jitter of the hardware. It sets up a thread to run periodically for ntests times in a loop and report the minimum and maximum deviation of the time it actually ran to the time it should have run by writing the data to a realtime fifo (rtf). The data can be retrieved from the fifo with the monitor program. The monitor will, by default, retrieve 10000 data samples and the terminate, by passing it the -s#para meter you can tell it to grab exactly # samples.

```
root@rtl15:/usr/src/rtlinux/examples/hello  # cd ../measurements
root@rtl15:/usr/src/rtlinux/examples/measurements  #
root@rtl15:/usr/src/rtlinux/examples/measurements  # sync ; insmod rt_process.o bperiod=0
```

load the measurement module with a sync again, if you don't want to do it, then simply `insmod rt_process.o` and find out the hard way why to sync your disk before loading a module ;) The `bperiod=0` module parameter instructs `rt_process.o` to launch only one thread and not launch a background thread that would compete for the CPU.

```
root@rtl15:/usr/src/rtlinux/examples/measurements  # ./monitor -s 1000 | tee data
```

We only collect 1000 samples and put them in the file data, while at the same time displaying them on the screen.

```
root@rtl15:/usr/src/rtlinux/examples/measurements  # ./gist data | tee data.out
```

Next we make a "histogram" of this data - Note that this is not a real histogram as we don't have access to the samples but only the min/max values of every ntests-large sample (default 500) so you can't interprete this data statistically. It does give you a fairly good overview of the systems rt-performance though - especially if you produce a high-load and high interrupt situation while running this test. If you launch `rt_process` without the `bperiod=0` parameter then you run two rt-threads and you can see what influence two threads completing at the same priority will have on the worst case scheduling-jitter of this specific system-hardware. So now lets clean up - removing the module and clearing the kernel message buffer again.

```
root@rtl15:/usr/src/rtlinux/examples/measurements  # rmmod rt_process
root@rtl15:/usr/src/rtlinux/examples/measurements  # dmesg -c
```

4.5 shut down rtlinux

```
root@rtl15:/usr/src/rtlinux/examples/measurements  # cd /usr/src/rtlinux
root@rtl15:/usr/src/linux  # ./scripts/rmrtl
root@rtl15:/usr/src/linux  # dmesg -c
```

just to check if there were any problems unloading the rtlinux modules !
Thats it - you now are rtlinux experts.... almost.

5 Debugging

To use gdb for debugging we need to reconfigure rtlinux and recompile it. To do this we first clean up and then launch menuconfig again.

```
root@rtl15:~ # cd /usr/src/rtlinux
root@rtl15:~ # make distclean
root@rtl15:~ # make menuconfig

  Support option --->

  [*] Posix standard I/O
  [*] POSIX Priority protection
  [*] Dev mem support
  [*] Enable debugging
  [*] rtl_printf uses printk
  [ ] Nolinux support
  [*] RTLinux tracer support (experimental)
  [*] Userspace Real Time
  [*] Floating Point Support
  [ ] RTLinux V1 API support
  [ ] RTLinux Debugger
  [*] Synchronized clock support
```

save and exit.

```
root@rtl15:/usr/src/rtlinux # make dep
root@rtl15:/usr/src/rtlinux # make
```

RTLinux is now rebuilt with debugging flags, and with the RTLinux debugger module in the debugger subdirectory (`rtl_debug.o`)

5.1 RTLinux Debugger

Before we can launch the debugger we need to reload the modified rtlinux modules, we can use the scripts in the top-level rtlinux directory again.

```
root@rtl15:/usr/src/rtlinux # ./scripts/insrtl
root@rtl15:/usr/src/rtlinux #  cd debugger
root@rtl15:/usr/src/rtlinux/debugger # insmod rtl_debug.o
```

RTLinux is now ready for debugging, before we insert the actual module to be debugged lets give it a look. If you look `hello.c` and compare it with `examples/hello/hello.c` you will find some debugging specific differences.

```
#include <rtl_debug.h>
...
void * start_routine(void *arg)
{
   ...
   if (((int) arg) == 1) {
       breakpoint();
   }
   ...
```

This `breakpoint();` instruction is the point where gdb will halt and you can continue from there on, if your module has not breakpoint and no bug that causes an exception then gdb can't connect, so either code a segfault or use the `breakpoint(); function ;`)

```
root@rtl15:/usr/src/rtlinux/debugger # insmod hello.o
root@rtl15:/usr/src/rtlinux/debugger # gdb hello.o
GNU gdb 5.1.1
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-suse-linux"...
(gdb) target remote /dev/rtf10
Remote debugging using /dev/rtf10
[New Thread -1012596736]
[Switching to Thread -1012596736]
start_routine (arg=0x1) at hello.c:37
37 for (i = 0; i < 20; i ++) {
warning: shared library handler failed to enable breakpoint
(gdb) l
32
33 if (((int) arg) == 1) {
34 breakpoint();
35 }
36
37 for (i = 0; i < 20; i ++) {
38 pthread_wait_np ();
39 rtl_printf("I'm here; my arg is %x\n", (unsigned) arg);
40 }
41 return 0;
(gdb) break rtl_printf
Function "rtl_printf" not defined.
(gdb) modaddsym ../modules/rtl.o
add symbol table from file "../modules/rtl.o" at
.text_addr = 0xc88da060
(gdb) modaddsym ../modules/rtl_time.o
add symbol table from file "../modules/rtl_time.o" at
.text_addr = 0xc88e0060
(gdb) modaddsym ../modules/rtl_sched.o
add symbol table from file "../modules/rtl_sched.o" at
.text_addr = 0xc88ea060
(gdb) break rtl_printf
Breakpoint 1 at 0xc88db189: file rtl_printf.c, line 39.
(gdb) c
Continuing.

Breakpoint 1, rtl_printf (fmt=0xc8905140 "I'm here; my arg is %x\n")
    at rtl_printf.c:39
39 {
(gdb) l
34 static char initial_printkbuf [MAX_PRINTKBUF];
/* need to protect in_printkbuf from overflowing */

35 static char in_printkbuf[MAX_PRINTKBUF]; /* please don't put this on my stack*/
36 static char *printkptr = &in_printkbuf[0];
37 static spinlock_t rtl_cprintf_lock = SPIN_LOCK_UNLOCKED;
38 int rtl_printf(const char * fmt, ...)
39 {
```

```
40 rtl_irqstate_t flags;
41 int i;
42 va_list args;
43
(gdb) quit
The program is running.  Exit anyway? (y or n)
```

What we did was connect to gdb via `/dev/rtf10` (that the '`target remote /dev/rtf10`' line), then gdb stopped at the breakpoint instruction, and we tried to set a breakpoint at `rtl_printf`, but that was not known because the symbols were not loaded, so next we use a convenient modaddsym macro found in the `.gdbinit` file of the debugger directory, to load the symbol information form the rtlinux core modules. Now setting of the breakpoint works fine, and we can continue (command 'c' in gdb), stopping at the first `rtl_printf`. Next we list (command 'l' in gdb) the `rtl_printf` function. There really is not much to do in this example so we quite (command 'q' in gdb).
As we were debugging the module, it was off course not running in realtime, and if you now type dmesg, you can see that the execution of the individual threads (`hello.o` spawns two threads!) gets confused.

```
RTLinux Extensions Loaded (http://www.fsmlabs.com/)
RTLinux Debugger Loaded (http://www.fsmlabs.com/)
rtl_debug: exception 0x3 in hello (EIP=0xc89050a8),
    thread id 0xc3a50000; (re)start GDB to debug
I'm here; my arg is 1
I'm here; my arg is 1
I'm here; my arg is 1
I'm here; my arg is 2
I'm here; my arg is 1
I'm here; my arg is 2
I'm here; my arg is 1
I'm here; my arg is 2
```

So gdb is good to find a segfault or some other coding error, but it will not allow to debug race-conditions or to locate temporal misbehavior of your realtime module, and in fact the behavior of multi-threaded apps can be quite strange in gdb, even for those that work fine when run free !
We are done with our debugging intro so lets clean up.

```
root@rtl15:/usr/src/rtlinux/debugger # dmesg -c
root@rtl15:/usr/src/rtlinux/debugger # rmmod hello
root@rtl15:/usr/src/rtlinux/debugger # rmmod rtl_debug
```

5.2 RTLinux Tracer

When we reconfigured RTLinux, with menuconfig, above, we enabled the RTLinux Tracer. The problem with GDB was that it did not allow temporal debugging, and this is because gdb is taking control of the rt-thread, so this is a rt-thread under control of a non-rt executable. To allow temporal debugging the RTLinux tracer was designed that we will briefly introduce here.

```
root@rtl15:/usr/src/rtlinux/debugger # cd ../
root@rtl15:/usr/src/rtlinux # insmod modules/mbuff.o
```

The mbuff module, contributed by thomas motylewsky, is a shared memory module, allowing to share memory between rt-threads and user-space applications. The RTLinux Tracer records important events with timestamps (system events are hard-coded, like entering and exiting the scheduler routine) and user-defined events can be included in your application that trigger writes of the buffered data into shared memory when ever a user-specified condition is met. This way you can back-trace what events lead to the event that you are watching.

```
root@rtl15:/usr/src/rtlinux # cd tracer
```

If we look into `rt_process.c` in the tracer directory and compare it with `rt_process` in examples / measurements then we can find the following difference. The recording of the maximum in the inner loop changes from:

```
if(diff > max_diff){
   max_diff = diff;
}
```

to include the `RTL_TRACE_USER` event recording the new absolute jitter maximum, and the trace buffer is flushed to shared memory where the user-space application can read it. (there are a few other minor changes like `#include <rtl_trace.h>` and the variable `abs_max_diff` not shown here - but those should be quite self explaining).

```
if(diff > max_diff){
   max_diff=diff;
   if(max_diff > abs_max_diff){
      abs_max_diff=max_diff;
      rtl_trace2(RTL_TRACE_USER,(long) abs_max_diff);
      rtl_trace2(RTL_TRACE_FINALIZE,0);
   }
}
```

Now every time a new maximum jitter value is encountered the buffer will be flushed, this way we can trace the path of events that leeds to a jitter maximum and thus (hopefully) locate the hot-spot in the code.
Now lets load the tracer module and the modified `rt_process.o` to watch it.

```
root@rtl15:/usr/src/rtlinux/tracer # insmod rtl_tracer.o
root@rtl15:/usr/src/rtlinux/tracer # insmod rt_process.o ; ./tracer | tee trace.log
P0 131828416 rtl_restore_interrupts        0x46 <c88e09cd>
P0    576 scheduler out           0xc88ecd64 <c88ea95d>
P0    480 rtl_restore_interrupts        0x96 <c88ea96b>
P0    512 hard sti                         0 <c88e0865>
P0    576 rtl_no_interrupts             0x286 <c88e0707>
P0    480 rtl_spin_unlock         0xc88e15f8 <c88e0212>
P0    416 rtl_restore_interrupts       0x203 <c88e021d>
P0  41184 rtl_no_interrupts            0x203 <c88e01c7>
P0    448 rtl_spin_lock           0xc88e15f8 <c88e01d6>
...
---snip---
...
P0    448 rtl_restore_interrupts        0x46 <c88e09cd>
P0    640 rtl_switch_to           0xc3ab8000 <c88ea7a0>
P0   1952 scheduler out           0xc3ab8000 <c88ea95d>
P0    640 rtl_restore_interrupts        0x92 <c88ea96b>
P0    992 rtl_restore_interrupts       0x297 <c88eaea4>
P0   3488 user                        0x1cc0 <c894f325>
That was trace # 1
```

The commands for insmod and starting the tracer are concatenated to a command sequence to make sure we launch the tracer fast enough as it happen quite frequently that the maximum is reached right at the beginning and then we don't see anything. Furthermore we redirect the tracer output to a log file (`trace.log`), to terminate the tracer type `<CNTRL>-<C>` . The tracer output will scroll by on the screen as the new absolute maximum is frequently encountered at the beginning and then output will stall, if you want to produce a new worst-case maximum, then switch to a different console (`<CNTRL>-<ALT>-<F2>`) login as root again and start something like.

```
root@rtl15:~ # ls -lR /
```

Note that the RTLinux Tracer does introduce a slight distortion on the systems realtime behavior so you will not be able to find everything this way, but its about as close to temporal debugging that you can get. After we terminated the tracer we do the usual cleanup.

```
root@rtl15:/usr/src/rtlinux/tracer #  rmmod rt_process
root@rtl15:/usr/src/rtlinux/tracer #  rmmod rtl_tracer
root@rtl15:/usr/src/rtlinux/tracer # rmmod mbuff
```

Thats it - have fun with real time Linux !

References

[1] [RTLinux/GPL on the web] *RTLinux/GPL Download Site*, `http://www.rtlinux-gpl.org`

[2] [RTLinux Thesis] Michael Barabanov, *Rtlinux*, 1996, New Mexico Tech.

[3] [RT-Synchronisatoin] V. Yodaiken: *Temporal inventory and real-time synchronisation in RTLinux/Pro*, FSMLabs Inc., 2003

[4] J. Vidal, F. Gonzalves, I. Ripoll: *POSIX TIMERS implementation in RTLinux, RTLinux-3.2-pre3, http://www.rtlinux-gpl.org*

[5] V. Yodaiken: *Priority inheritance is a non-solution to the wrong probem*, Technical report, FSMLabs Inc., 2002

[6] [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`