RTLinux Installation Guide

The Ohio State University
Prof. Chang-Gun Lee
Spring 2005
ECE 694z

Nick Martino
Derek Miller
Robert Foerster

RTLinux Installation Guide

This installation manual will assume that you already have a suitable Linux distribution installed and running properly on your system. Also, a basic level of Linux knowledge is assumed, including how to move about the Linux file system, and how to copy files, extract files, etc. For sake of compatibility, we suggest using RedHat 8.0 (Psyche). Any other distribution should work as well, but some commands may be different, and various other troubles may be encountered. Some brief information on installing RedHat 8.0 is available at the end of this document.

We will guide you through the process of installing RTLinux. All information contained within is combined from the RTLinux installation guide prepared by Prof. Chang-Gun Lee of The Ohio State University, the Kernel Rebuild Guide by Kwan Lowe (http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html), and our personal experiences installing RTLinux.

1. RTLinux installation

1.1 Get the right tools

Various tools will be needed to compile a new kernel, including the kernel development tools, the gcc compiler, and a number of other utilities including patch, depmod, make, and tar/bzip2. Obtaining all of the necessary tools is easiest when selecting those packages on a fresh install of RedHat 8.0. The tool versions that come packaged in the RedHat 8.0 installation work great for compiling a new kernel with RTLinux support. If you have an existing system, you will need to track down all of the necessary tools on your own. To find the minimum software requirements necessary to build the linux kernel, you can look in the ./Documentation/Changes file of the installed sources. For easy reference, the requirements to build a 2.4.X kernel are provided here:

Tool	Required Version	Check version you have installed
Gnu C	2.91.66	# gcc --version
Gnu make	3.77	# make --version
binutils	2.9.1.0.25	# ld -v
util-linux	2.10o	# fdformat --version
modutils	2.4.2	# insmod -V
e2fsprogs	1.19	# tune2fs
reiserfsprogs	3.x.0b	# reiserfsck 2>&1|grep reiserfsprogs
pcmcia-cs	3.1.21	# cardmgr -V
PPP	2.4.0	# pppd --version
isdn4k-utils	3.1pre1	# isdnctrl 2>&1|grep version

Table 1: Required tools to build a 2.4.X kernel (2)

When you have all of the necessary tools installed, you're ready to move on to the RTLinux installation.

1.2 Get the sources

Now that you have a working Linux system and the right tools for compiling a new kernel, we are ready to get started with our RTLinux installation. We first need to obtain the source for kernel version 2.4.18 and the RTLinux kernel package.

Go to **www.kernel.org/mirrors/** and download the file “linux-2.4.18.tar.bz2”.
Go to **www.fsmlabs.com/community/** and download the file “rtlinux-3.2-pre1.tar.bz2”.

To make life easier in the following steps, save the files in the /usr/src directory.

1.3 Install kernel sources – adapted from (1)

If you didn’t download the files into the /usr/src directory, please copy them into that directory now. Before we proceed, we need to make sure we’re in the /usr/src folder, so, at a command prompt, issue the following command:

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

Now that we’re in the right directory, we first extract the RTLinux archive. This can be done with the command:

```
# tar xjf rtlinux-3.2-pre1.tar.bz2
```

*note: the j option tells tar to filter the file through bzip2 to uncompress the file

This creates the directory /usr/src/rtlinux-3.2-pre1, which is filled with all of the necessary RTLinux files.

At this point, we need to put a fresh copy of the Linux kernel into /usr/src/rtlinux-3.2-pre1/linux. To do this, first copy the linux-2.4.18.tar.gz archive file into /usr/src/rtlinux-3.2-pre1. This can be done with the command:

```
# cp /usr/src/linux-2.4.18.tar.b2z /usr/src/rtlinux-3.2-pre1
```

Now we will move into the rtlinux-3.2-pre1 directory with the command:

```
# cd /usr/src/rtlinux-3.2-pre1
```

We now extract the linux kernel archive with the command:

```
# tar xjf linux-2.4.18.tar.bz2
```

This creates a directory named linux-2.4.18 in /usr/src/rtlinux-3.2-pre1. We want to create a symbolic link to this folder with the name linux so we can refer to the directory by the name “linux” instead of “linux-2.4.18” repeatedly. To create this link, issue the command:

```
# ln –fs linux-2.4.18/ linux
```

1.4 Patch the Linux kernel with RTLinux patches

All of the sources are now in their proper places and we’re ready to proceed with the real installation work. Next we will patch the linux kernel with the rtlinux patches. Move into the patches directory and unzip the kernel 2.4.18 patch:

```
# cd /usr/src/rtlinux-3.2-pre1/patches
# bzip2 –d kernel_patch-2.4.18-rtl3.2-pre1.bz2
```

After the patch has been extracted, we can patch the kernel:

```
# cd /usr/src/rtlinux-3.2-pre1/linux
# patch –p1 < ../patches/kernel_patch-2.4.18-rtl3.2-pre1
```

This will apply all of the necessary patches to the linux kernel that will allow rtlinux to perform as expected.

1.5 Configuring the kernel

This is the point in installation where things can get tricky. We are now ready to configure the linux kernel that we're going to install. This required an in depth knowledge of your hardware. Since you already have a running linux system, we can use that to our advantage to gain information about our hardware. To do this, try running the command (sample output shown):

```
# /sbin/lspci

00:00.0 Host bridge: Acer Laboratories Inc. [ALi] M1644/M1644T Northbridge+Trident (rev
01)
00:01.0 PCI bridge: Acer Laboratories Inc. [ALi] PCI to AGP Controller
00:02.0 USB Controller: Acer Laboratories Inc. [ALi] USB 1.1 Controller (rev 03)
00:04.0 IDE interface: Acer Laboratories Inc. [ALi] M5229 IDE (rev c3)
00:06.0 Multimedia audio controller: Acer Laboratories Inc. [ALi] M5451 PCI AC-Link
Controller Audio Device (rev 01)
00:07.0 ISA bridge: Acer Laboratories Inc. [ALi] M1533 PCI to ISA Bridge [Aladdin IV]
00:08.0 Bridge: Acer Laboratories Inc. [ALi] M7101 PMU
00:0a.0 Ethernet controller: Intel Corp. 82557/8/9 [Ethernet Pro 100] (rev 0d)
00:11.0 CardBus bridge: Toshiba America Info Systems ToPIC95 PCI to Cardbus Bridge with
ZV Support (rev 32)
00:11.1 CardBus bridge: Toshiba America Info Systems ToPIC95 PCI to Cardbus Bridge with
ZV Support (rev 32)
01:00.0 VGA compatible controller: Trident Microsystems CyberBlade XPAi1 (rev 82)
```

If you have this tool on your system, it should generate a list of information similar to the one shown above, but tailored to your hardware that will be of great use in properly configuring your kernel.

Also of use will be specific information about your processor (for those of you that don't know what type of processor they have). Some systems have a /proc file system that will allow you to view some raw information about the system. If /proc exists, you can issue this command to view your CPU information (sample output shown):

```
# cat /proc/cpuinfo

processor     : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 8
model name    : Celeron (Coppermine)
stepping      : 10
cpu MHz               : 997.667
cache size    : 128 KB
fdiv_bug      : no
hlt_bug               : no
f00f_bug      : no
coma_bug      : no
fpu           : yes
fpu_exception : yes
cpuid level   : 2
wp            : yes
flags         : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat
pse36 mmx fxsr sse
bogomips      : 1951.26
```

Now that you've gathered the necessary hardware information, we can begin the configuration process. We begin by wiping out all previous configurations and resetting the source directory to a pristine state. The main reason for doing this is that some files do not automatically get rebuilt, which can lead to failed builds, or at worst, a buggy kernel (2).

```
 # cd /usr/src/rtlinux-3.2-pre1/linux
# make mrproper
```

You can use a pure text mode configuration editor (not very nice), a console based windowed editor (nice), or an X window based editor (nice). If you already have X-windows started, go ahead and use the X mode, otherwise, I'd recommend using the console based window mode.

For text mode:

```
# make config
```

For console based window mode:

```
# make menuconfig
```

For pure X-windows mode:

```
# make xconfig
```

For more information on configuring the kernel, please refer to section 3 below.

1.6 Make new kernel dependencies

After you have configured your kernel with the (hopefully) correct options, we are ready to build our new linux kernel. The next step is to create the necessary include files and generate dependency information. To do this, we use the command:

```
# make dep
```

Many messages will scroll past the screen; this is normal, so don't panic. Depending on the speed of your machine, this process may take several minutes to complete. After the dependency information has been created, we can clean some miscellaneous object files by issuing the command (2):

```
# make clean
```

1.7 Build & Install the new kernel

At this point, we are finally ready to actually build the kernel. To do so, issue this command at the prompt:

```
# make bzImage
```

This process can take a large amount of time. On fast systems with small kernel configurations it may take 7-10 minutes, on slower systems with larger kernels, it may take upwards of 45-60 minutes. It's best to start the build process, go do some other work, and come back to check progress in an hour or so (2).

If you run into problems with the build -- namely the build fails with errors -- there could be any number of problems which caused this failure. One common problem is a missed step in the setup process. Go back over the steps presented in this document

and make sure you performed each step properly. If this doesn't solve the problem, there may be configuration errors in your kernel. There is some additional information about kernel configuration provided below in section 3.

*It should be noted that some systems just won't work with 'make bzImage'. This can be either due to hardware problems or old versions of lilo. If this is the case for you, you can try building the kernel with 'make zImage' if your kernel is really small (2).

We have now created the kernel, but we are not yet done. In our kernel configuration, you have the option to add features to the kernel by compiling them directly into the kernel, or by using them as loadable modules. If we have some features that are to be used as loadable modules, we need to compile and install them. Be forewarned, if you have a large number of loadable modules, this process can also take 30-60 minutes! To build the modules we run:

```
# make modules
```

The modules we need are now built, so we need to install them. We will need root privileges for this so we need to switch to root with the su command, and enter roots password, then we can install the modules:

```
# su
Password:
# make modules_install
```

The new modules will be copied into /lib/modules/2.4.18-rtl3.2-pre1

Next we need to copy the newly created kernel into the /boot directory so that we can use it. We will name our new kernel image rtzImage:

```
# cp /usr/src/rtlinux-3.2-pre1/arch/i386/bzImage /boot/rtzImage
```

1.8 Make initial RAMDisk image (may not be necessary) – adapted from (2)

If you have built your main boot drivers as modules (e.g., SCSI host adapter, filesystem, RAID drivers) then you will need to create an initial RAMdisk image, or initrd. The initrd is a way of sidestepping the chicken and the egg problem of booting -- drivers are needed to load the root filesystem but the filesystem cannot be loaded because the drivers are on the filesystem. As the manpage for **mkinitrd** states:

> **mkinitrd** creates filesystem images which are suitable for use as Linux initial ramdisk (initrd) images. Such images are often used for preloading the block device modules (such as SCSI or RAID) which are needed to access the root filesystem. **mkinitrd** automatically loads all *scsi_hostadapter* entries in */etc/conf.modules*, which makes it simple to build and use kernels using moduler SCSI devices.
>
> MKINITRD(8)

To create your initial RAMDisk image for our 2.4.18 kernel that has been patched with the RTLinux patches, issue the command:

```
# /sbin/mkinitrd 2.4.18-rtl3.2-pre1.img 2.4.18-rtl3.2-pre1
```

After making the initrd image, we need to copy it into /boot so it can be loaded during the PC boot process:

```
# cp 2.4.18-rtl3.2-pre1 /boot
```

1.9 Configure your boot loader – adapted from (2)

The next step is to configure your boot loader. The boot loader is the first program that runs when a computer is booted. For this document it is assumed that you are running an IA32 system with a standard PC BIOS. If you are running the LiLO boot loader skip to the the Section 1.9.2, otherwise simply continue reading onward. Both configurations shown below assume that /dev/hda1 is your /boot partition, and /dev/hda2 is where your root file system is located.

1.9.1GrUB Configuration

GrUB is beginning to supplant LiLO as the bootloader of choice in more recent Linux distributions. It is generally more flexible and a lot more forgiving of system errors. For example, LiLO generally requires that an alternate boot disk is used if the kernel configuration renders the system unbootable Grub allows "on-the-fly" modification of kernel location, boot parameters, kernel to boot, etc..

Once you have copied the bzImage to /boot, edit the grub configuration file located in /boot/grub/menu.lst. On some distributions /etc/grub.conf is a symbolic link to this file.

Edit the file to include your new kernel information. Keep in mind that GrUB counts starting from 0, so (hd0,1) references the first controller, *second* partition. If you have created an initial RAMdisk be sure to include it here too. You'll need to add lines similar to this at the bottom of the file:

```
        Title Real Time Linux (2.4.18-rtl3.2-pre1)
                root (hd0,1)
                kernel /boot/rtzImage ro root=/dev/hda2
                initrd /boot/2.4.18-rtl3.2-pre1.img
```

For more information, consult http://www.gnu.org/software/grub/.

1.9.2 LiLO Configuration

LiLO is an older boot loader that is being used less and less these days. The configuration file you need to edit for is located in /etc/lilo.conf on most systems. You need to append these lines to your lilo.conf file:

```
        image=/boot/rtzImage
        label=rtlinux
        root=/dev/hda2
        read-only
```

After you save the file with the new lines added, you need to run the command:

```
# /sbin/lilo
```

Running this command applies the changes you just made.

For more information on configuring LiLO, consult http://www.freeos.com/articles/2701/ and http://www.linuxheadquarters.com/howto/basic/lilo.shtml.

1.10 Try to boot RTLinux

Reboot your system and select RTLinux from your boot loader's menu. If everything has been done properly to this point, you should be able to boot with no problems.

1.11 Configure RTLinux – adapted from (1)

Now that we have booted RTLinux, we need to do some final configurations for it to be usable. To do this, we will switch to the RTLinux directory and run the make program to configure the settings. Just accept the defaults.

```
# cd /usr/src/rtlinux-3.2-pre1
# make menuconfig
```

1.12 Compile RTLinux

In a manner similar to how we compiled the kernel earlier, we now need to compile RTLinux itself. We first need to make the dependencies, compile RTLinux, make all necessary devices, and then install everything. We can do this with these commands (in order presented):

```
# cd /usr/src/rtlinux-3.2-pre1
# make dep
# make
# su
Password:
# make devices
# make install
```

1.13 Run Regression Tests – adapted from (1)

Now that we have booted the RTLinux enabled kernel and compiled RTLinux itselt, it is time to test RTLinux. To do this, switch to /usr/src/rtlinux-3.2-pre1/ and run scripts/regression.sh.

```
# cd /usr/src/rtlinux-3.2-pre1/
# ./scripts/regression.sh
```

You should get [OK] for all tests. If you do not, something is wrong with your installation. Unfortunately, there is no easy way to find out what is wrong, so you should probably just start back at the beginning and try again.

2. Run RTLinux Applications

2.1 Insert dynamic modules – adapted from (1)

To perform the following steps you will need root privileges, so please su as root now. We now want to ensure that everything is working properly, so we will run some example programs found in /usr/src/rtlinux-3.2-pre1/examples. Before we can run the programs, we need to load some dynamic rtlinux kernel modules such as mbuff, rtl_fifo, rtl, rtl_posixio, rtl_sched, and rtl_time. Each of these modules can be loaded using the insmod command, but to save time, it is much easier to use the script named 'rtlinux', which can insert all of these modules for you. Using 'rtlinux status' will show you which modules are currently loaded. Using 'rtlinux start' will attempt to insert all of the aforementioned modules into the kernel. We want to check the currently loaded modules, attempt to insert them all, and then verify that they have been loaded:

```
# rtlinux status
# rtlinux start
# rtlinux status
```

If you can see all of the modules loaded (they will have + signs by them), you are done. Otherwise, you might see error messages while you were doing "rtlinux start". These errors may be due to incorrect kernel configurations. See Section 3 concerning kernel configuration and try different settings. After you reconfigure the kernel, try to rebuild the Linux kernel again and proceed through all of the necessary steps.

If you were successful in inserting all RTLinux modules, go to examples/hello directory and make hello.o. Then, insert hello.o into the kernel to run it:

```
# cd /usr/src/rtlinux-3.2-pre1/examples/hello
# make
# insmod hello.o
```

To see if the hello module has been successfully inserted into the kernel, list the modules using:

```
# lsmod
```

In order to see the messages from the hello module, you should do this in text mode. If you are in X mode, you can check the output message from hello.o by typing

```
# dmesg
```

Finally, you can remove the hello module by typing

```
# rmmod hello
```

Note that .o should not be used in rmmod.

3. Notes on Linux Configuration – adapted from (1)

Configuring the Linux kernel is the most difficult part in making RTLinux run successfully. The required configuration is specific to the hardware that comprises your system, so it will generally be different for each installation. The following information should be regarded as suggestions, not a definitive guide. It may not work for your system. For your first attempt, you should only change the options listed below and keep the rest set as the defaults. If that doesn't work properly, then incrementally change options that seem suspicious. If your configuration does work, feel lucky and proceed on to Section 2, which is about running and using RTLinux.

3.1 A quick note on file systems and the kernel

In order for your new RTLinux enabled system to work properly, your kernel must have the ability to read the file system which is running on your system. Some file systems that you may be using are Reiserfs, ext2 journaling file system, and the ext3 journaling file system. To see which file system your drive is formatted with, issue this command (sample output shown – items of interest in bold):

```
# cat /etc/fstab

LABEL=/         /          ext3     defaults    1 1
LABEL=/boot     /boot      ext3     defaults    1 2
```

One issue we ran into when building our new kernel is that 2.4.X kernels do not have ext3 file system support natively, and ext3 is what RedHat 8.0 defaulted to for our install. Rather than reinstall our entire RedHat 8.0 system, we found an ext3 kernel patch for 2.4.X kernels. If you have ext3 on your system and need to patch it, you can download it from http://www.zip.com.au/~akpm/linux/ext3/. The file you need to download is “ext3-2.4-0.9.17-2418p3.gz”. You need to patch your kernel with this just as we patched the kernel with the RTLinux patches earlier. Download the aforementioned file into /usr/src/rtlinux-3.2-pre1/patches. Now use these commands to patch the kernel:

```
# cd /usr/src/rtlinux-3.2-pre1/patches
# gunzip ext3-2.4-0.9.17-2418p3.gz
# cd ../linux
# patch –p1 < ../patches/ext3-2.4-0.9.17-2418p3
```

Now your kernel is patched with ext3 file system support. When configuring your kernel using one of the “make config” methods, make sure to enter the section named “File systems” and enable the option “Ext3 journaling file system support”. Now you should be able to read the ext3 file system when you boot the RTLinux kernel.

3.2 Configuration Tips

For your first attempt at configuration, make sure to apply these settings:

1. **Code maturity level options**: check YES

2. **Loadable module support**: check all of 1) Enable loadable module support, 2) set version info on all system module, and 3) kernel module loader.

3. **Processor type**: select your processor type. Refer /proc/cpuinfo" for your processor information.

4. **Network device support**: choose the proper device depending on your network device. If you don’t know what card you have, you can find out using the "System Settings/Network Configuration tool" in X mode.

5. Check **kernel hacking** and also check **Magic SysRq Key**.

6. For all others, keep the default for the first try. If the configuration fails, change the settings of suspicious options one at a time. Note that you can save your settings into a file and use it as the starting point next time so you don’t have to remember what options you’ve tried. For this, you can store your settings into a file named “myconfig”. Then, next time you want to change some configuration

options, you can copy the “myconfig" file to “.config". “.config" is the file that “make config", “make menuconfig”, and “make xconfig" use as the default. Also, note that “make mrproper" in the process of building the kernel will delete the “.config” file, so be sure to copy “myconfig” to “.config” after running “make mrproper”.

4. RTLinux Manual Page Installation – adapted from (1)

For developing RTLinux applications, we need to be familiarized with RTLinux system calls. Due to the large number of available system calls, it is nearly impossible to remember them all. The RTLinux developers were kind enough to provide RTLinux manual pages which provide information on the system calls and their syntax. In an event when the correct system call or exact syntax is not known, it is a good idea to refer to the manual pages for assistance.

To install the manual pages,

1. Download “rtldoc-3.0.tar.bz2" from the course website.

2. Copy it to /usr/src/rtlinux-3.2-pre1/doc.

3. Extract the documentation tarball

```
# cd /usr/src/rtlinux-3.2-pre1/doc
# tar xjvf rtldoc-3.0.tar.bz2
```

This should create the directory /usr/src/rtlinux-3.2-pre1/doc/rtldoc-3.0

4. Add the directory path to the MANUAL PAGE search path.

- Method 1: Add the following line to /etc/man.config:

```
MANPATH /usr/src/rtlinux-3.2-pre1/doc/rtldoc-3.0/man
```

This method makes the RTLinux man pages available to all users on the system.

- Method 2: Add the following line to .bashrc in your home directory:

```
export MANPATH=$MANPATH:/usr/src/rtlinux-3.2-pre1/doc/rtldoc-3.0/man
```

This method makes the man pages available only to you.

5. Try “man rtf_create" or “man 3 rtf_create" to test that your manual page paths are correct.

References

1. Chang-Gun Lee. RTLinux installation manual. http://www.ece.osu.edu/~cglee/EE694Z/rtlinux/rtlinuxInstallation2005Sp.pdf
2. Kwan Lowe. The Kernel Rebuild Guide. http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html