RTOS, Spring 2015 – Lab #2: RTAI and Linux kernel modules

Paolo Torroni, paolo.torroni@unibo.it
Davide Chiaravalli, davide.chiaravalli@studio.unibo.it
[partially adapted from text by Paolo Mantegazza, www.rtai.org , Harco Kuppens, www.cs.ru.nl/lab/rtai, and other sources]

Objective: to understand kernel modules in Linux (RTAI) and learn how to list, load, and remove them

1. Background

Make sure you have read and understood the introduction to RTAI in Appendix.

2. Start up RTAI

A) Power up your station.
B) From "Select the operating system to boot" menu, select "Rtai."
C) Username/password: rtai
Notice: RTAI is not always available in the boot meu in Lab2.
Notice: after log off, data erased (no permanent private folders). It is advised that you bring your own USB stick where you can permanently save files.

3. Learn some more useful shell commands

A) uname -r
B) wget http://lia.deis.unibo.it/Courses/RTOS/input/naming
C) less naming | grep cat
D) dmesg
E) tail -f /var/log/kern.log
F) lsmod
Notice: remember that to quit an application you may often use q or CTRL+c

4. Compile and run a kernel module

A) Obtain hello.c and Kbuild from lia.deis.unibo.it/Courses/RTOS/src/2015-2/
B) Open hello.c and analyze code. Notice printk() instead of printf(). Why?
C) Open Kbuild and analyze code.

What is Kbuild? In the newer linux versions, kernel modules are built using the kernel build system 'kbuild', which uses a so-called 'Kbuild' makefile to make a module. The Kbuild file contains instructions about how to compile:
- where are Linux kernel source tree [/lib/modules/...]
- what target [modules:]
- what object files are needed [.c -> .o]
- which compiler to use [$(MAKE)]

Kbuild has a typical format:

```
# Kbuild file
KDIR ?= /lib/modules/`uname -r`/build
modules:
	make -C $(KDIR) M=`pwd` modules
obj-m := hello.o
```

Notice: after "modules:" there is new line and the next line must start with a TAB.
D) make -f Kbuild
E) sudo insmod hello.ko
F) lsmod
G) sudo rmmod hello
H) tail -f /var/log/kern.log

5. Load RTAI kernel modules and run latency testsuite

A) sudo insmod /usr/realtime/modules/rtai_hal.ko
B) sudo insmod /usr/realtime/modules/rtai_sched.ko
C) lsmod | grep rtai
D) cd /usr/realtime/testsuite
   cd kern/latency
   sudo ./run

6. Self assessment

- ☐ What information is displayed in each of three columns by **lsmod**?
- ☐ What shell command can be used to download a file from the Internet?
- ☐ How are kernel modules compiled?
- ☐ How are kernel object files used?
- ☐ Does loading a kernel module require special privileges?
- ☐ Where can I see the output produced by kernel modules?
- ☐ Does Linux schedule RT tasks?
- ☐ Does RTAI schedule user-space tasks?
- ☐ Can a RT task interact with a Linux task running in user-space?
- ☐ What happens when we load/remove the following module?

```
/*
 * hello.c
 */

#include <linux/kernel.h>
#include <linux/module.h>

int init_function(void);
void exit_function(void);

module_init(init_function);
module_exit(exit_function);

int init_function(void)
{
  printk("Hello World!\n");
  return 0;
}

void exit_function(void)
{
  printk("Goodbye World!\n");
  return;
}
```

- ☐ What does the folder /usr/realtime/modules contain?

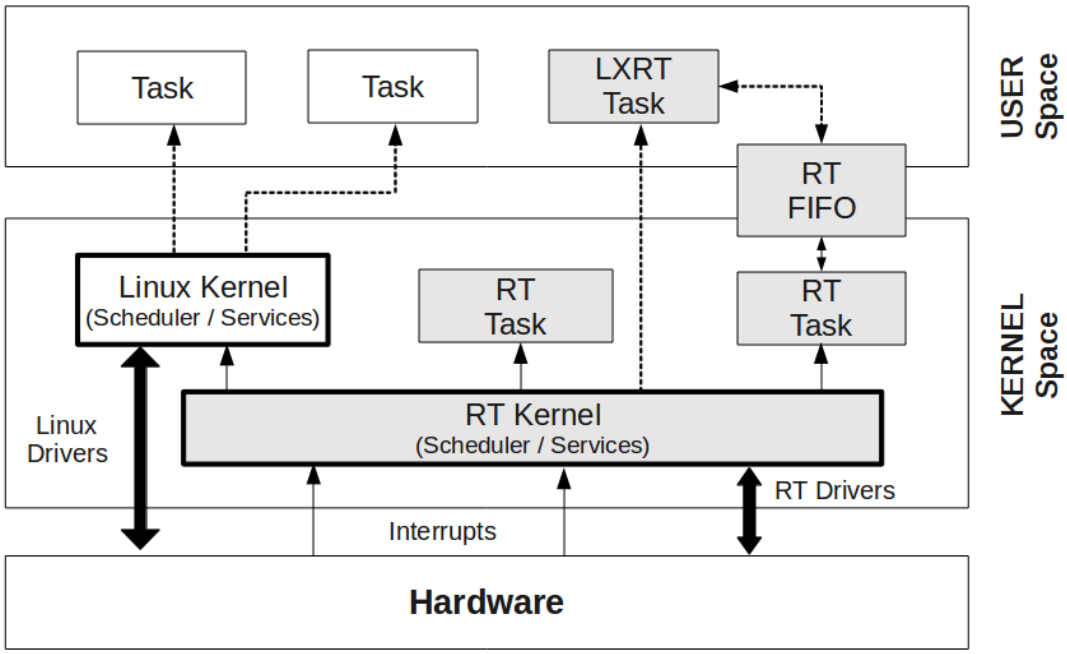
Appendix. About RTAI

RTAI is not a Real-Time Operating System. It is a **module** in dormant state ready to overtake Linux. RTAI adds RT features to Linux. To this end, RTAI's RT scheduler replaces the original Linux scheduler, and:

- intercepts the timer interrupt and external device interrupts,
- runs any real-time code associated with these, and
- runs any normal Linux processes in the time left over.

Essentially, for RTAI, Linux is a background task.

An **interrupt dispatcher** traps the peripherals interrupts and if necessary re-routes them to Linux.



Installing RTAI requires patching the Linux kernel (patches are available for certain versions of the Linux kernel, see rtai.org).

RTAI Linux tasks are "kernel modules," meaning they run as part of the privileged Linux kernel, similar to device drivers. Key RTAI modules are:

- **rtai_sched** (real time scheduler module)
- **rtai_fifos**: IPC
- **rtai_shm**: allows sharing memory among different real time tasks and Linux processes
- **rtai_lxrt**: implements services to make available any of the RTAI schedulers functions to Linux processes

→ see them in /usr/realtime/modules

Kernel modules, like device drivers, execute in a primitive environment, without direct access to many user-level Linux facilities like terminal or file I/O.

WARNING: Errors may crash the system (running in privileged mode)

Kernel modules are dynamically loaded using the **insmod** (insert module) shell command, and unloaded (stopped), using **rmmod** (remove module). These commands are only available to the root user (administrator)

→ use "sudo" to run a program with root privileges (e.g., sudo insmod …)

C programs are normally compiled into full executable programs, but kernel modules are compiled into object code. Compiling a kernel module does not produce a full-blown executable file, but **a loadable '.ko' (kernel object) file**.

In C, a program's "entry point" where execution begins is a function called 'main()'. For a kernel module, this entry point is declared inside module_init(), or else is called 'init_module()' by default. 'insmod' looks for module_init or init_module when loading the code.

A module's "exit point" is a function declared inside module_exit(), ('cleanup_module()' is default). This will be invoked when 'rmmod' removes the kernel module.