FSM: The RTLinux™ Company RTLinux FAQ

Maintained by: FSM Labs

July 26, 2001

Abstract

This document is in LaTeX syntax and is adopted from the pdflatex FAQ pages.

1. GENERAL QUESTIONS

1.1. Q: What is RTLinux?

A. FSMLabs RTLinux^{TM}(RealTime Linux) is a small realtime operating system that is used for systems where precise timing, down to a few microseconds or less, is needed. For example, RTLinux is used to run telescopes (at Kitts Peak), instruments (by NASA and many others), machine tools (NIST, Lang GmBh,...), high speed network switches (Huawei and Alcatel), and all sorts of other things where “usually fast enough” is not good enough.

1.2. Q: Is it hard to program?

A. RTLinux follows the POSIX 1003.13/PSE51 standard and so its API is pretty close to ordinary POSIX threads/signals. RTLinux “kernel” applications look like threads and signal handlers running on a tiny operating system close to the bare machine.

Realtime applications on RTLinux are almost always made up of two parts: a realtime kernel and the parts that do data logging, non-realtime networking, GUIs, data analysis or display, and anything else that does not need precise timing. This non-realtime part runs in either Linux or BSD UNIX and uses the ordinary programming interface of these systems. One of the big advantages of RTLinux is that realtime programmers can use a simple, very efficient, threads/signals environment for hard realtime software, use a regular operating system with many features for everything else, and glue the parts together. A typical application might use a standard database, a Perl-script, and a data analysis package — all driven by a realtime thread.

1.3. Q: What is hard real-time?

A. **Hard realtime** means that timing is not “kinda-sorta” or “typical” or “generally”. That is, if a robot arm needs to be sent a signal every 500 microseconds, it gets a signal **every** 500 microseconds (within some small, known, error) and not *typically* in 500 microseconds or every 500 microseconds unless something else is using the processor.

> When researching realtime operating systems, be sure to compare ”worst case” timings. It’s tempting to quote ”typical” times, but unless you only need your RT application to ”typically” meet timing requirements, this is not particularly useful.

1.4. Q: How does RTLinux work?

RTLinux uses a patented process to run a general purpose operating system like Linux or BSD Unix as its lowest priority thread and to make sure

that this general purpose operating system can always be preempted (interrupted) whenever a realtime operation needs to run. The basic mechanism that makes RTLinux work is protected under U.S. Patent 5,995,745 and licensed without fee to RTLinux users for commercial and non-commercial purposes as described below.

1.5. Q: Who is FSMLabs?

A: Finite State Machine Labs, Inc. is the company formed by the creators of RTLinux to give it commercial support and development. FSM offers RTLinux in both GPL and non-GPL versions, related realtime software, training and consulting.

1.6. Q: What platforms does RTLinux run on?

A: Almost every x86 (SMP and uniprocessor), plus PowerPC and Alpha. RTLinux/Pro also supports MIPS.

1.7. Q: How much memory is needed and how powerful a processor?

A: RTLinux still works on the i486. The MiniRTL release fits RTLinux, Linux and some applications on a single floppy disk and runs in 4Meg of memory.

1.8. Q: How well does it work?

A: Quite well. Worst-case times are about 15microseconds between the assertion of an interrupt and the starting of the realtime handler on a generic x86 PC, and better on the Alpha and PowerPC platforms. These times are close to the hardware limit, yet all of the power of Linux remains easily accessible to the realtime programmer.

1.9. Q: How can I get RTLinux?

A: There are three ways to get RTLinux:

FSMLabs sells a CD-ROM with RTLinux V3. The purchase price includes 30 days of email support to get you up and running.

If you wish to get RTLinux for free, go to www.fsmlabs.com or www.rtlinux.com and push the ”download” buttons.

RTLinux/Pro can be purchased from FSM by contacting business@fsmlabs.com.

1.10. Q: Is there support?

A: Yes. FSM sells yearly support for both RTLinux/Pro and RTLinux/GPL. And purchase of the RTLinux v3.0 CD entitles you to 30 days of email support to

help you get your system up and running. In addition, we do everything from short term consulting to developing subsystems to longterm support. For a listing of FSMLabs products and services, see www.fsmlabs.com/fsmservices.pdf. For specific questions, email: support@fsmlabs.com. The mailing lists at www.rtlinux.org/mailinglists.html are quite active and a particularly good source of high quality free support.

1.11. Q: What is POSIX?

A: POSIX (the ”Portable Operating System Interface”) is a specification which dictates how operating systems should behave. Among other things, it specifies basic operations involving signaling and threads, making it easier for programmers to port their applications from one operating system to the other. Lots of documentation on using POSIX calls is readily available.

RTLinux API is based on a small system POSIX “profile” that is specified in POSIX standards 1003.13 PSE51. This profile is for a ”minimal realtime system” environment and it looks very much like a multithreaded single POSIX process. (On SMP systems, we have a realtime process per processor.) Of course, each realtime process may have a Linux (or BSD) thread.

1.12. Q: What’s the license? What about the patent?

A: FSM provides two versions of RTLinux: GPL and Profesional. FSM charges for RTLinux/Professional and RTLinux/Professional contains features and ports that are not in GPL RTLinux, however, FSM supports and develops both versions, both can be used for commercial and non-commercial projects, and source is available for both. (there is a GPL question below for more details).

GPL RTLinux is released under the GPL Version 2 and the Open RTLinux Patent License and can be used, modified, and redistributed under the terms of that license. If you modify RTLinux code, the new code is automatically governed by the GPL and **all** application software you write that uses the RTLinux RT method must also be under the GPL. The Open RTLinux Patent License can be found at www.fsmlabs.com/PATENT.html. There is no fee for using GPL RTLinux as long as things are properly labled and credited and you *scrupulously* follows the free license terms. If you violate or evade the terms of the GPL, our copyright or the Open RTLinux Patent License, you have invalidated the free licenses.

RTLinux/Professional is available with or without source and we charge a fee for binary distribution (customers may not redistribute source).

1.13. Do I need licenses?

A: No. Once you have accepted the Open RTLinux Patent License you are free to use RTLinux Open for commercial work: as long as you obey the terms of the license.

1.14. Q: What is FSM's attitude towards the GPL and "open" source?

FSM is in favor of the GPL where it is appropriate and against the GPL where it is not. We sell non-GPL code and we sell rights to incorporate non-GPL code into RTLinux while keeping it closed. On the other hand, we have a 6 year history of developing GPL RTLinux and are continuing development of GPL RTLinux, we sell support for GPL RTLinux, and we participate in development of GPL Linux. Our non-GPL products, such as RTLinux/Pro, are "open-source" in the sense that customers can get source, modify it, fix bugs, and use it as in-depth documentation. We think that for critical embedded applications access to source is of fundamental value.

FSM is a business and we don't have a point to prove about source code licenses. Our objectives are to make a profit by selling high quality products and services to customers. The fundamental values of the business have to do with providing reliable systems that can be used to make reliable applications and with treating customers and employees well. As for licenses, we don't get too excited about them.

2. THE COMPETITION

2.1. Q: What makes RTLinux different from other realtime operating systems?

A: The main advantage of RTLinux over earlier RTOS designs is that it allows programmers to write applications that combine the advantages of a lean, hard realtime operating system at hardware speeds with all the features of a general purpose operating system. RTLinux's emphasis is performance and ease-of-use, not features. RTLinux relies on the underlying Linux (or BSD) system to provide all soft realtime capabilities and features (GUI, soft realtime networking, plotting, disk access, etc.), while RTLinux itself guarantees the timing correctness of all hard realtime tasks.

Before RTLinux, RTOSs required programmers to accept a compromise: either use a simple RTOS without any features, or use an RTOS mixed in with all sorts of non-realtime services – and accept a significant loss in performance and reliability. The RTLinux method of running Linux as a thread avoids this choice. Hard realtime software can run in the realtime kernel *and* make use of services and programs running in the non-realtime thread – but realtime software is never delayed or slowed down by the non-realtime software. After RTLinux appeared, several other operating systems adopted the same method, but we don't think that anyone of these imitators has understood the importance of decoupled operation of RTLinux in terms of time-to-market, performance, and development costs.

2.2. Q: What's the difference between RTAI and RTLinux?

A: RTAI is a variant of RTLinux that has gone off on its own path. GPL encourages such divergence and the Open RTLinux Patent License allows use of the RTLinux basic process in any GPL code without a fee – as long as the license terms are followed.

Though many differences are purely cosmetic, there is a fundamental difference between the two systems in attitude towards features. The RTLinux developers have worked in OS development for many years and are convinced that the realtime side of an RTOS should be as small and modular as possible and we spend a lot of time considering interactions between components and those "small" costs of features that turn fast software into slow software over a few years. We are very conservative about adding new features or systems, and we work very hard to make sure that any additional functionality we add remains optional: Users who don't need a specific function don't pay any performance price. RTAI seems to be much more feature friendly. We are inclined to say, "We're not sure this won't cause a problem three years down the line, so let's leave it out," and the RTAI folks are more likely to say "This looks useful now." As they say on the Internet: your mileage may vary.

Two other differences: RTLinux follows the POSIX API, RTAI has a blend of

the first RTLinux “V1” API, POSIX, and things that are original to RTAI; RTLinux runs in production versions on multiple architectures; RTAI does not.

2.3. Q: What’s the relationship between FSM and Lineo?

A: Lineo sells RTLinux/GPL based services and products and so we hope they advance the product, contribute to free source, and educate people about the virtues of the RTLinux model.

2.4. Q: What version of Realtime Linux is provided on the Lineo CD?

A: Lineo says it has some earlier version of RTLinux and RTAI on its CD.

2.5. Q: What’s the relationship between FSM and MontaVista?

A: FSM and MontaVista have a formal partnership that allows MontaVista to sell “Hard-Hat” Linux with RTLinux as an enhancement, supported by FSM. MontaVista also has a GPL “realtime” version of Linux that they are developing in-house. We see this as an interesting research project and believe that any improvements in Linux “low latency” operation will only make RTLinux more useful.

3. HELP DESK QUESTIONS

3.1. Q: How do I get started with RTLinux?

A: The INSTALL file in the distribution contains installation instructions. GettingStarted.txt is a good starting point on the API.

3.2. Q: What is miniRTL?

A. MiniRTL is a tiny implementation of RTLinux which fits on a 1.44MB floppy and runs on i486 machines. It is targeted especially towards industry-standard PC-104 boards. This is a minimally-sized standalone, bootable, networked realtime Linux system that includes the following features:

- Linux Kernel 2.2.13
- RTLinux v.2.0
- glibc 2.0.7
- Full network support via ethernet, slip *or* plip (it's only a floppy...)
- inetd/telnet/tftp access
- ssh/scp access for added security
- sunrpc support as loadable modules (portmap NFS)
- mail (outgoing mail only...)
- mini_httpd , a full blown httpd with cgi-bin support

More information about miniRTL can be found at www.rtlinux.org/minirtl.html.

3.3. Q: What is RTiC-Lab?

A: RTic-Lab is a graphical front end for hard realtime control via RTLinux. It gives the controls engineer access to:

- Plant states,
- Plant I/O,
- Controller states,
- Controller parameters (scalar or matrix), and
- Hard realtime environment for plant modeling.

Run time data can be saved/displayed into:

- stdout,
- disk,
- RTiC-Scope, a standalone Oscilloscipe Emulator, or
- any standalone application written by the user. Communication between the user's program and RTiC-Lab is handled by RTiC-Lab's easy-to-use API.

RTiC-Lab can be obtained from www.rtic-lab.org. If you install RTLinux from the FSMLabs v3.0 CD, RTiC-Lab is placed on your hard disk at that time.

3.4. Q: What about priority inversion/priority inheritance?

A: Any priority-based realtime system will have a problem with mutual exclusion. When a lower-priority task owns a resource that a higher priority task wants, mutual exclusion algorithms make the higher priority task wait – priorities are "inverted." The correct solution to this problem is to make clean use of mutual exclusion mechanisms, by making sure, for example, that operations on shared resources are simple and fast. The incorrect approach is to try to use semaphores to guard use of shared resources. This is a fundamentally incorrect design and can lead to a problem like this:

- The low priority task acquires a semaphore.
- The high priority task blocks waiting for the semaphore.
- A medium priority task runs, keeping the low priority task from getting a chance to release the semaphore.

Some people claim that semaphores using something called "priority inheritance" fix this problem. In priority inheritance, when a high-priority task blocks, it raises the priority of the task holding the semaphore, allowing it to complete. This is a dangerous method that we do not support in RTLinux. Our belief is that designers of realtime systems must understand exactly what shared resources exist, and that they should design access methods to these resources that are bounded in time. Consider what happens under priority inheritance if the low-priority process acquires the semaphore and waits for a second semaphore, or for an I/O event. Priority inheritance tries to cover up a design problem with a complicated, slow and failure-prone hack. We have yet to see a problem that could not be better solved another way. Ask us.

3.5. Q: Will my X-windows or Oracle or [you name it] Linux program work with RTLinux?

A: Yes. RTLinux runs a very slightly modified version of Linux as a subtask. There are no application-level visible differences – except that realtime devices are usable.

3.6. Q: I get errors when trying to build RTLinux. What's wrong?

A: Check that:

- The "linux" directory (or symbolic link) points to the Linux kernel source tree
- the Linux kernel is patched with the RTLinux patch. (To check this, see if `linux/include/asm-i386/rtlinux\_cli.h` is present.)
- You have performed both:

  `make config`

  `make dep`

  in the Linux kernel directory.

3.7. Q: How do I compile my own programs for RTLinux?

A: You need to put the following line into your `Makefile`: `include rtl.mk`. The rtl.mk file is created during compilation of RTLinux. It contains the paths to include files and compiler options.

You need to put the `rtl.mk` file into a place where `make` can find it, for example in the current directory.

Simple programs can be compiled with ¡userinput¿make -f rtl.mk rt_process.o¡/userinput¿

3.8. Q: Why does the RTLinux kernel crash?

A: There are several possible causes.

- You may be using the wrong kernel. Recent versions of "patch" fail SILENTLY if you use the wrong kernel version.
- Some distributions contain kernel sources with patches applied that might not be compatible with RTLinux. The kernels that come with RedHat generally will not work with the RTLinux patch. If unsure, try downloading the kernel sources from ftp.kernel.org, www.kernel.org, or any one of its mirrors.
- Advanced Power Management (APM) is enabled in your kernel. Recompile your kernel and disable APM – it interferes with the RTLinux kernel.
- Your BIOS has special power management features. Restart your computer, enter the BIOS and disable these features.
- Check that you have specified the correct CPU type. For example, if you have a Pentium, be sure to change the CPU type from the default 686 to P5.

- gcc 2.7.2.3 or egcs 1.1.2 (2.91) is recommended for 2.2.x kernel compilation. If your gcc is a different version, this may be a problem. RedHat systems are fine. On newer Debian systems, you will need to install the gcc272 package.

  For 2.4.x kernels, there is a known problem with RedHat 7.x systems. If you are using them, you need to change

  ```
  CC              = $(CROSS_COMPILE)gcc
  ```

  in the Linux kernel Makefile to

  ```
  CC              = kgcc
  ```

If nothing works, please report the problem to the mailing list (rtl@rtlinux.org).

3.9. Q: Is there a debugger for Real-Time threads?

A: Beginning with RTLinux v2.3, the debugger is included in the `debugger/` subdirectory of the RTLinux distribution.

3.10. Q: How can I prevent programming errors in my RT-programs from crashing the system?

A: See `debugger/README`. To intercept all hardware exceptions, simply load the debugger.

3.11. Q: Can I debug the initialization code with the RTLinux debugger?

A: Not at this time. You should move the initialization code to the beginning of one of your pthreads code.

3.12. Q: Does the debug module support multi-module debug?

A: Yes, although the support is not complete as gdb has bugs with respect to symbol handling. You need to use add-symbol-file to load other modules's symbol table into gdb. The `.gdbinit` file in the `debugger/` directory contains some useful macros to do that:

```
modaddsym module.o    --- load symbols for module.o
modaddsched           --- load symbols for the RTLinux scheduler
```

The latter is very useful when using gdb's `backtrace` command. For macros to work, an rtl.mk file should be present in the current directory.

There is an annoying bug in add-symbol-file (gdb 4.18): data symbols do not use correct addresses. Examining code does work.

A (not very convenient) way to solve the problem with data symbols is to kill gdb, and start it on the module that generated the last exception.

3.13. Q: Can I use the RTLinux debugger to debug RTLinux modules running on a remote machine?

A: Yes. Install the `netcat` program on the remote machine. After an exception in a realtime thread has occured, you can forward the RT-FIFO used by the debugger to your local machine with the following command:

```
ssh remotehost -L 3000:remotehost:5000 'nc -l -p 5000 /dev/rtf10
/dev/rtf10'
```

In gdb, you can connect to the remote debugger with:

```
target remote localhost:3000
```

.

The `netcat` program is available, for example, in the Debian distribution.

3.14. Q: Why do modules fail to load (e.g., sh scripts/insrtl fails)?

A: You must have root access to do this (man "su").

3.15. Q: How do I get rid of a "couldn't find the kernel version the module was compiled for" message?

A: Use

```
#include <;rtl.h>;
```

in your Linux module.

3.16. Q: I'm seeing large delays in scheduling of RT threads

A: There are several possible causes:

- Advanced Power Management (APM) is enabled in your kernel. Recompile your kernel and disable APM – it interferes with the RTLinux kernel.
- The video hardware locks the PCI bus. This can be apparent if delays happen when dragging windows in X. Try using a different video card.
- Several people reported that PS/2 mouse can cause substantial delays; switching to serial mouse generally fixes it.
- The ISA bus devices may introduce delays as well. PCI-only systems are recommended.
- SDRAM memory timings are incorrect.

3.17. Q: Why do I get messages about unresolved symbols during insmod?

A: Try doing a full recompilation:

- `make mrproper` (in the Linux kernel directory)
- `make clean` (in the RTLinux top-level directory)
- Reboot, and your problem should go away. In general, recompiling from scratch tend to solve most problems.

Another problem may be that your modutils are too old. See the `Documentation/Changes` file in the Linux kernel directory, and upgrade your system software as required.

3.18. Q: Why do I get ”Error opening /dev/rtf0” messages?

A: You need to create RT-devices in /dev/. To do this, become root, cd into the top-level RTLinux directory and type:

```
make devices
```

3.19. Math symbols

```
h.o: unresolved symbol __isnan
h.o: unresolved symbol fputs
h.o: unresolved symbol stderr
h.o: unresolved symbol __assert_fail
```

A: You need to link in the C library. Add -lc after -lm in the linker command line.

3.20. Q: Can I use floating-point operations in realtime threads?

A: Currently, only on the x86 and PowerPC platforms, provided the CPU supports floating point. (For x86, this means 486 DX and up.) Kernel FP emulation will *not* work. To use floating-point in a thread, you should set the FP flag in the pthread attribute structure at the time of thread creation:

```
pthread_attr_setfp_np (&amp;attr, 1)
```

An alternative is to use the `pthread_setfp_np` function. It must be called in the thread before any FP-operations.

For a working example of FP operation in RT-threads, see `examples/fp`.

¡note¿ For PowerPC, you need to append ”`-mhard-float` ” to `CFLAGS` . See `examples/fp/Makefile`.

For functions like `sin()`, you will need to link in math library (-lm), e.g.

```
include rtl.mk
rt_process.o: rt_process.c
$(CC) ${CFLAGS} -c -o rt_process_tmp.o rt_process.c
ld -r -static rt_process_tmp.o -o rt_process.o -L/usr/lib -lm
rm -f rt_process_tmp.o
```

¡/note¿

3.21. Q: Can I use C++ in my RTLinux threads?

A: See `examples/cpp`.

3.22. Q: How can I use one-shot scheduling for my RT-threads?

A: You can use the `clock_nanosleep`. See `examples/misc/nanosleep.c`. You can use the absolute timeout of `hrt2ts(HRTIME_INFINITY)` to make the task sleep forever.

3.23. Q: How do I create an interrupt-driven task?

A: In the realtime task, use the following loop:

```
while (1) {
pthread_suspend_np(pthread_self());
... // handle the interrupt
}
```

In your initialization routine (such as `init_module()`), install an interrupt handler using `rtl_request_irq`. In the interrupt handler itself, call `pthread_wakeup_np` (`thread_id`) to wake the realtime thread up, where the `thread_id` denotes the identifier of the realtime task.

3.24. Q: How do I suspend a task?

A: Use the `pthread_suspend_np()` function.

3.25. Q: How can I make my RT-thread go to sleep?

A: Use `nanosleep()` or `usleep()` or `clock_nanosleep()`. An example is provided in `examples/misc/nanosleep.c`.

3.26. Q: How do I use shared memory in RTLinux?

A: Use `mbuff`. See `drivers/mbuff`.

3.27. Q: How do I access physical memory from RT-programs?

A: The `/dev/mem` interface is available through the `rtl_posixio` module. See the `/dev/mem` description in `GettingStarted.txt`

3.28. Q: Is it possible to create RT-threads from other RT-threads?

A: Yes, starting from RTLinux version 3.1. The thread stack should be pre-allocated and passed to `pthread_create()` via pthread attributes See `pthread_attr_setstackaddr`, An example can be found in `examples/misc/create_recursive.c`.

3.29. Q: Is it possible to share RT-Linux semaphores, mutexes, and other objects between modules?

A: Yes. By default, loaded modules export all non-static symbols to the global symbol table.

Module 1:

```
sem_t common_sem; /* define the semaphore  */
...
sem_wait(&common_sem);
```

Module 2:

```
extern sem_t common_sem; /* declare external semaphore */
...
sem_post(&common_sem);
```

Module that defines the semaphore (in this case, Module 1), has to be loaded first.

3.30. Q: How do Linux processes communicate with RTLinux threads?

A: RTfifos provide a device interface that can be read/written on the Linux side. (Data transfer is over 100Mbytes/second in a modern x86.) The simplest use of this interface is with a shell command:

```
cat < /dev/rtf1 > logfile
```

This is a workable data logging program. There is also a sophisticated shared memory utility, and since Linux is a ”thread,” you can send it signals via `pthread_kill()`.

3.31. Q: Why do I get messages about unresolved symbols in insmod?

A: You need to load the `rtl_posixio.o` module from the system directory. Alternatively, you can disable POSIX IO support during RTLinux configuration and recompile.

To save yourself the trouble of figuring out what modules need to be loaded, type `make modules_install` , and then:

`modprobe rtl_sched` `modprobe rtl_fifo`

This will load modules needed for running basic realtime programs. Refer to `modprobe(8)` .

3.32. Q: An RT-FIFO device seems to only provide one-way communication with the RT system. I need both reads and writes to work on the Linux side.

*A:*Yes. You can use bidirectional fifos. On the RT-side, this is a pair of ordinary uni-directional FIFOs. On the user side, one /dev/rtfXX device handles both reads and writes. Please see the `rtf_make_user_pair` manual page.

3.33. Q: Is there any way to flush a fifo ?

A: Use:

```
rtf_flush(unsigned int fifo);
```

3.34. Q: How do I print messages from RTL-programs?

A: Use

```
#include <;rtl_core.h>;
rtl_printf(const char *{\em format},...);
```

The arguments to this function are the same as to `printf(3)` . The messages from `rtl_printf` are directed to the kernel buffer and syslog. You can view them by running `dmesg` and/or viewing `/var/log/messages`.

¡note¿ It is *unsafe* to use `printk()` from RTLinux threads and interrupt handlers.¡/note¿

3.35. Q: How do I print 64-bit time values ?

*A:*You can use

```
rtl_printf("%d%09d", n / 1000000000, n % 1000000000);
```

Alternatively, you can convert your 64-bit timevalue into a struct timespec using `struct timespec timespec_from_ns (long long t);`

3.36. Q: What are the time units in the old RTL versions 1.x?

A: They are the ticks of the `8254` PC timer. There are `1193180` ticks in a second. `rt_get_time()` returns the number of 8254 ticks since the `rtl_time` module was installed with `insmod(8)` .

3.37. Q: Why do the programs using the old RTLinux API (versions 1.x and older) fail to compile?

A: You need to enable `CONFIG_RTL_USE_V1_API` (`make config`) and recompile the system (`make clean; make`).

4. OPEN SOURCE GPL AND PATENTS

4.1. Is FSM a GPL company?

FSM is neither a GPL company nor a Linux company although we make extensive use of both, contribute to both, and are certainly in favor of both. We don't think that GPL is the answer to every problem and we make sure our paying customers who don't want to GPL their code can protect their own innovations. Of course, this applies to us as well, and we also sell a non-GPL version of RTLinux with many improvements: Customers can get the source for this system, but it is not freely redistributable.

It's worth pointing out that some of the biggest supporters of GPL software are in a wonderful position. You can pore through IBM's Linux for S390 code, copy it, improve it, market it, etc. and whatever you do, IBM still makes its money selling hardware, firmware, and the support that will come to the manufacturer. For IBM, or for Oracle, or SAP or Intel, or for anyone else owning a product that cannot be freely copied, a free product that expands the market is only good. Maybe even Microsoft will do an analysis of Office sales and discover the good side of GPL one day.

So we think GPL is a good thing, we see it as a permanent fixture in the market, and we plan to continue developing RTLinux GPL. But our business is to provide high quality realtime OS technology to paying customers and to make a profit. When that means releasing code under GPL, we will do so. When it means releasing code under a different license we will do that.

4.2. Is the Open Patent License GPL compatible?

Legally, we don't think there is a question. The GPL is a copyright license, it specifically restricts itself to copyright issues of copying, modification, and distribution. The RTLinux Open Patent License is a patent license, it licenses use of the patented process, an issue that has nothing to do with copying, modification, and distribtion.

Ethically, we think it's pretty straightforward too. The Open Patent License says that if you want to use the RTLinux process under GPL code, you can do so without paying a fee. If you don't like GPL for your code, then you need to either use unmodified RTLinux that FSM releases, or get a non-GPL license from FSM. If you like the GPL, you get to use our process under the GPL, if you don't like the GPL, then purchase a license (and our licenses are quite reasonable).

5. CREDITS

CREDITS

5.1. Q: Who created RTLinux?

A: RTLinux began life as a research project at New Mexico Tech in Socorro, New Mexico. The design and method came from Victor Yodaiken. RTLinux was implemented first, and much of its initial API came from Michael Barabanov.

RTLinux is developed and maintained by FSMLabs, Inc. The main kernel developers are Victor Yodaiken, Michael Barabanov and Cort Dougan, although many other people have contributed code or testing. Notable among these are:

- Jerry Epplin. The V1 Semaphores and IPC package.
- Andreas Franzen. Testing. Debian versions.
- Zdenek Kabelac. Testing.
- Jochen Kuepper. RT-com.
- Paolo Mantegazza. Help with floating point. Criticism. Competition.
- Jens Michaelsen. Drivers.
- Tomasz Motylewski. The king of shared memory.
- Patrick Mourot. Testing.
- David Olofson. Testing.
- Steve Rosenbluth. DAQ drivers.
- David Schleef. COMEDI – a system for generating DAQ drivers.
- Phil Wilshire. Testing.
- And the following students and graduates of New Mexico Tech: Axel Bernal, Oleg Subbotin, Ramesh Nalluri, Jan Deninger, Hua Mao and Jose Guilberto.
- Bill Krauss: Documentation.
- Edgar Hilton: Documentation, testing, rtic-lab.sourceforge.net.

If your name has been left off the list unfairly, please support@fsmlabs.com.