RAT – Real-Time Audio Tools

Arvid Staub
Austria
arvid@gmx.at

Abstract

There is a strong tendency towards using modern Personal Computers for various tasks they are not designed for. This also includes audio and video processing. Audio and Video data is real-time data. It needs to be dispatched within predefined and fixed time windows to avoid perceivable lags and dropouts. Neglecting this very fact, today's software solutions are built on General Purpose Operating Systems that do not guarantee any timing behaviour.

Target of this project is creating an open-source versatile real-time capable data aquisition, routing and manipulation framework. The "Audio" part in the name refers to the concepts' suitability for audio recording.

The goal of the software design is to maximize reliability. No sample must be lost during data aquisition or manipulation. Of course, one can't eliminate the possibility of too slow hardware, disk failure, network packet or connection loss and similar events. In these cases, the software must report an error immediately.

This paper describes a way to use a hard real-time operating system for the tricky task of reliably recording and reproducing audio samples and video frames. It relies on a working implementation. This implementation uses the Linux operating system, extended with real-time capabilities by RTLinux/Free. It is directed towards software developers with Linux knowledge. The concept described relies on Linux kernel mechanisms, real-time specific programming and hardware management.

1 The Software Concept

Realtime **A**udio **T**ools split the various tasks needed to aquire and dispatch data by their timely importance. All critical processing is done in ISRs[1] or in real-time context. On the other hand, storage access is left to Kernel Space, user interaction to User Space.

The result of that is promising: It is no problem any more to record 10 channels of audio data while a disk benchmarking software is simultanously trying to measure the I/O throughput of the very same hard disk **rat** is writing to.

Almost all modules of **rat** reside inside the Linux Kernel. This allows them to share memory easily. And they do this excessively.

The user interface resides (as the name might suggest) in Userland. In this stage of development, the only existing interface is a non-graphical command-line tool, which has limited, but sufficient functionality.

A Message Queue is the communication interface between the **rat** -core (which does all the work) and Userland (which typically just sits and watches what is going on).

[1] Interrupt Service Routine

1.1 System Structure

Figure 1 shows the modular structure of **rat** . The central module which has total control over everything is the *router*. It provides registration hooks for devices and plugins, handles communication to Userland via a Message Queue and is responsible for routing the data streams.

The critical section is the transition between Real-Time Space and Kernel Space. Plugins are allowed to defer their data handling under certain cirumstances while still maintaining the clean split between the "Spaces" which is required to maintain system integrity. Without this controlled "breakage" of system barriers, it would not be possible to access Linux' I/O subsystem. The interface itself does not regulate how synchronisation with external resources should be done. Sections 4 (Plugins) and 4.2.1 (Linux File I/O) shows this in detail.

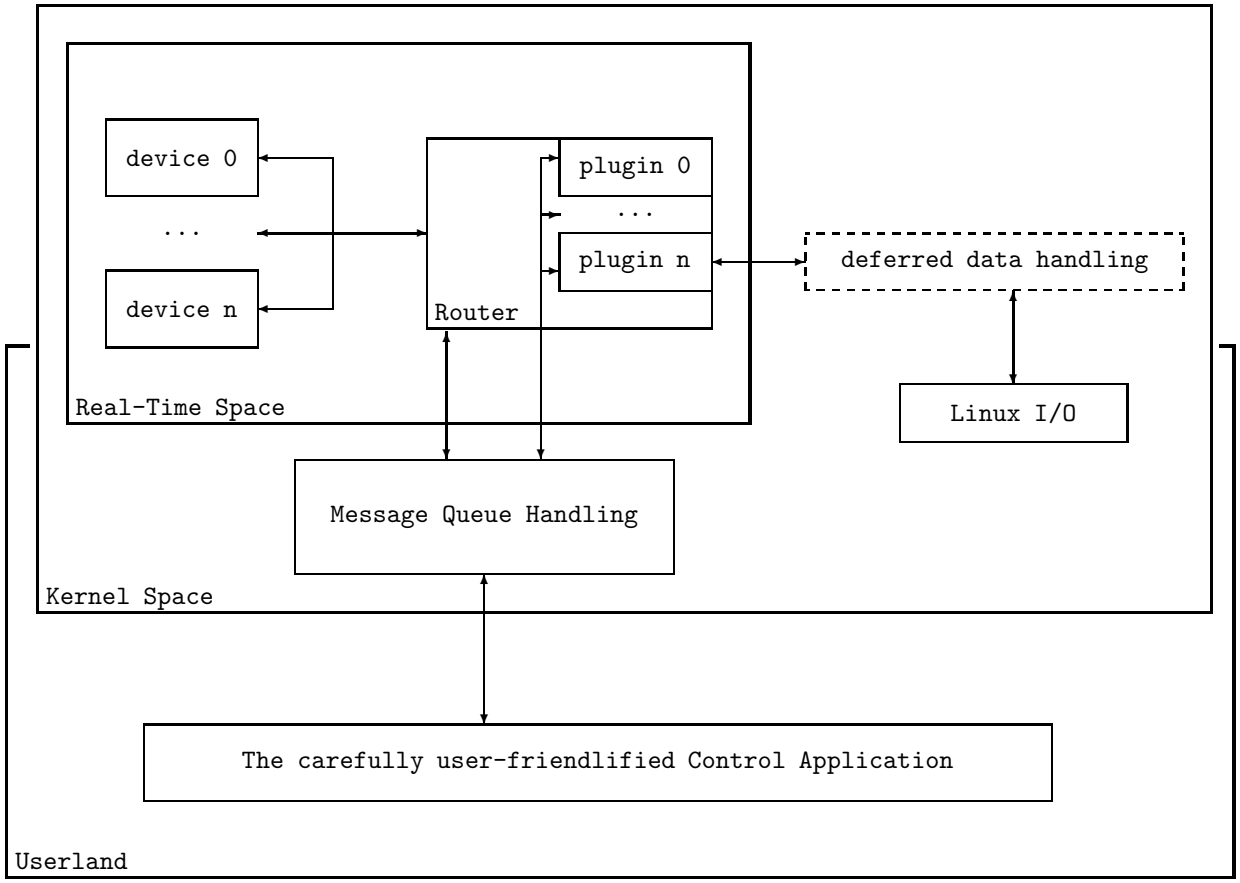

FIGURE 1: *System Overview*

1.2 Software Interfaces

To keep track of all known devices and plugins, all of those need to use a cleanly defined interface to register their capabilities at the router. There are currently two defined interfaces:

- UDAI - **U**nified **D**ata **A**quisition **I**nterface for hardware drivers
- LATIF - **L**ikely **A**dvanced **T**ool **I**nter**f**ace for router plugins

These interfaces hide the real hardware and processing details from the router. By using the abstraction, any data processing step can be implemented as a plugin and assigned to data channels at runtime. Also, the router does not have to know any details about the underlying hardware that is used to aquire or dispatch the data.

Plugins can also act as data sources or sinks. This allows to use them as interface to the Linux I/O subsystem or networking layer.

1.3 Framework and Helper Functions

Some widely used functionality has been implemented in helper modules. These are:

- Lock-Free ring buffers that are used to pass data between the modules
- Message Queue implementation for RT-Space, Kernel Space and Userland
- some minor macros for various purposes

2 The Router

The `rat-router` is the core which manages the operation of the system. The kernel module `rat-router.o` needs to be loaded before any hardware driver or plugin module can be inserted into the Linux Kernel.

2.1 Design and Implementation Details

The `rat-router` is an event-driven data dispatcher. It needs to be triggered by the underlying device

drivers. Whenever a device triggers the router (using the UDAI's callback function), it will mark the routing thread for execution and immediately return to the caller. This behavior ensures real-time and ISR safety.

The routing thread will execute the tree-style routing chains. Every input channel has it's own routing tree which itself consists of so-called work steps. Each work step references a plugin instance (See Section 4 (Plugins) for details) and a arbitrary number of chained work steps.

A routing tree could look like depicted in Fig/ 2.

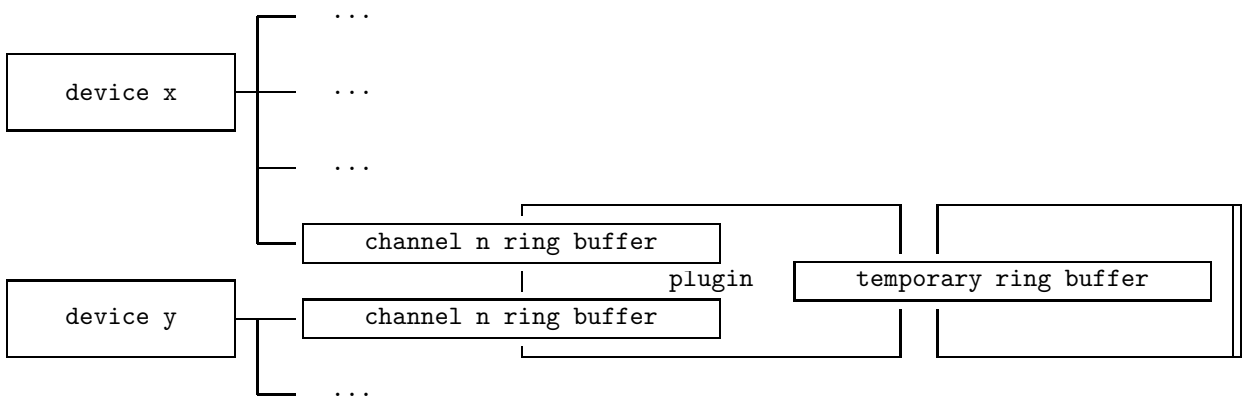

FIGURE 2: *Routing tree*

The "plugin" mentioned in the diagram should rather read as "plugin instance". Every plugin must be capable of creating any number of instances of itself. These instances must have their own runtime data and therefore be completely independent.

The last plugin in a routing chain must reference a special plugin which acts as a data source or data sink. It must therefore either have no input channels or no output channels.

Routing trees and all temporary buffers are maintained by the `rat-router`.

Note: Although theoretically possible, the current API does not allow a routing tree to span multiple devices (like shown in the diagram). By now, all chained channels have to be on the same device driver.

Plugins do not have to care about in which direction they are used. The router transparently handles the differences between input and output device routings.

2.2 Splitting Real-Time and Non-Real-Time operation

The router is entirely executed in Real-Time Space. Thus, it must not use any Linux Kernel synchronisation primitives, except some very special ones. Every function call that could possibly invoke the Linux Scheduler is inherently dangerous and must be avoided. Therefore, the router designed not to synchronize with Kernel Space or Userland. One exception to this rule is the Message Queue. It's handler is the only part of the router which is executed from a system call (Kernel Space).

2.3 Extendability

The router alone cannot do anything with the data. It is a mere dispatching and flow-control unit. Data handling itself must be implemented as plugins, which are Linux Kernel Modules themselves. They become a part of the router by registering their interfaces at load time. Some examples for plugins are:

- File System Access
- Compression
- Mixing
- Sample Rate Conversion
- Filtering
- Stream Mulitiplexing and Demultiplexing

The File System Access plugins for reading and writing regular files are already implemented and stable.

2.4 Things Still to be Done

The router is in an already-usable but still far from complete state. The current implementation can be considered as a prototype which shows how an implementation could be done. Main issues are:

- No data type, sample type or sample rate awareness

- Plugins are not yet allowed to use the message queue.
- large work step chains are untested, but should work
- A lot of cool plugins have to be written

3 Drivers

Hardware device drivers are more or less independent modules that can do whatever they want, as long as they stick to the semantics defined in the **U**nified **D**ata **A**quisition **I**nterface.

3.1 The Unified Data Aquisition Interface

3.1.1 Overview

Data is always passed using the lock free ring buffers defined in `include/rat_buffer.h`.
The **UDAI** is based on callback functions for both partners, the router and the driver.

An input (capture) driver has to stuff some data into the defined ring buffers and then call the "data-ready" callback to signal the router that it should do something with the data.

An output (playback) driver requests additional data by calling the same callback function.

What really goes on inside a device driver is a completely different issue. The abstraction layer introduced in this section does not try to standardize hardware access mechanisms. It aims to be as device-independent as possible. Therefore, any type of device, be it a parallel port or a high-end data aquisition card, can be interfaced to the Router.

3.2 The NULL driver

The NULL device driver is a fake driver that can be used to test `rat-core` without having any supported hardware around. The module provides an arbitrary number of input and output channels which can be configured to consume or produce a fixed amount of data per time. This is implemented by simple real-time threads that fill or empty the associated ring buffers.

It also contains a very simple watermarking algorithm that can be used to verify the data integrity of the router. Whenever a data packet is sent to the NULL comsumer, the watermark is checked an an error message is issued on failure.

3.3 The Terratec driver

3.3.1 Hardware Overview

This driver is compatible with TerraTec's `Phase 88` and `EWS 88 MT` Audio Systems. The Phase 88 is the stripped-down and re-marketed successor of the EWS88MT. It lacks the AC97 codec, which was useless on the EWS88MT anyway. It might have made sense on the EWS88D which does not have any other analog outputs aside from this one, but on an 8-channel analog I/O system it is a complete waste of ... everything.

But still, these two cards contain some very nice chips:

- ICE1712 - (a.k.a. envy24) PCI host controller
- CS8405A - 96kHz S/PDIF Transmitter
- CS8413 - 96kHz S/PDIF Receiver

Both cards do not have on-board ADC/DACs. These are enclosed in a 5 1/4" breakout box, which contains:

4 AK4524s - 24bit stereo ADC/DACs up to 96kHz

A MIDI Interface - ... which I did not care about too much

The *envy24* is a versatile PCI audio controller. It can handle data streams up to 10 channels of 24bit each at 96kHz in full duplex and has a limited on-chip channel router. In fact, this router is so limited, that **rat** would not benefit from using it.

Sadly, the *envy24* does not have any on-card memory. It completely relies on DMA burst transfers that are issued just-in-time.

Newer PCI devices implement the *Minimum Grant* and *Maximum Latency* registers in the `PCI Configuration Space`. By doing so, they tell the PCI subsystem that their timing is critical within the given parameters.

The envy24 stupidly hardwires both values to 0. This would be all right on systems where no other PCI device has these values implemented.

Nowadays, almost all devices make use of in-time DMA and timing restrictions, and so the envy24 finds itself on the low-priority end of the PCI bus.

This can lead to problems, imagine the following situation:

> Our audio device streams audio data at 96kHz. So, what do we do with the data? We write it to a hard disk or a file server. Both of these operations require DMA and Interrupts themselves. The downside of that is, that most modern network card and harddisk controllers implement the timing constraint registers mentioned

above. So we can quickly get to the point where the audio card is not granted PCI bus access when it needs to transfer a burst of samples. We lose data. And there is nothing we can do about this. Moreover, we won't even notice, until we listen to the sound.

The driver was derived from an old version of the GPL'ed ALSA driver by completely rewriting buffer and interrupt management.

3.3.2 Theory of Operation

Figure 3 shows the basic operation of the **rat** with the ews88mt device driver.

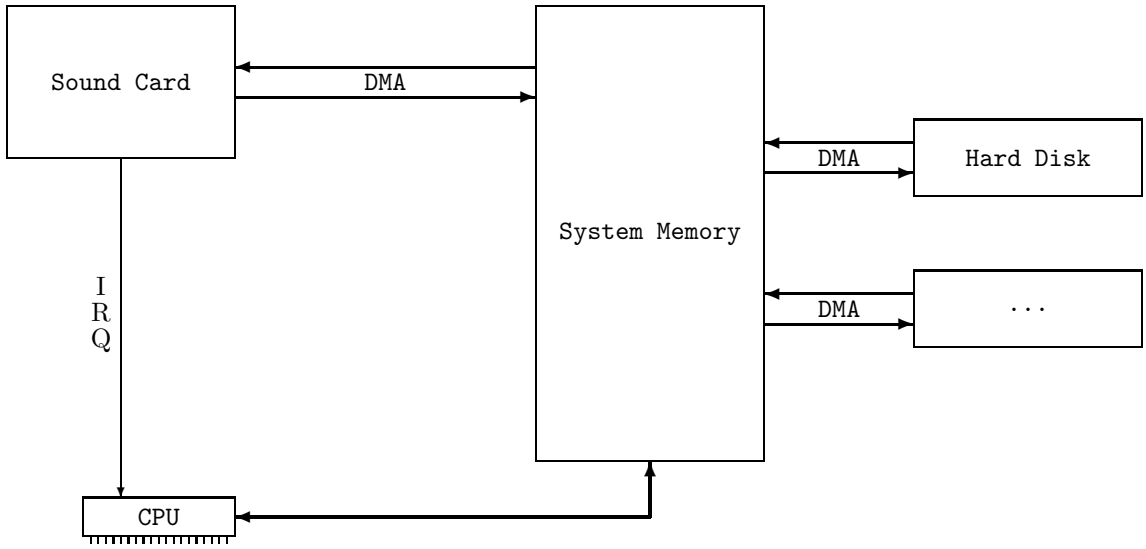

FIGURE 3: *Simplified Theory of Operation*

Audio samples are digitized by the ADCs, transferred via DMA by the envy24 controller, and then processed by the `rat-router` - That's the CPU's part in here. Much later on (in a computer's manner of speech), the audio stream is written to a file using the disk writer plugin. This also works the other way round for playback using the disk reader, of course.

3.3.3 Managing Direct Memory Access

Direct Memory Access (*DMA*) has become very easy to use in the 2.4 Linux Kernel series. There are also a lot of excellent books[2] about this issue out there, so I won't go into detail here.

The Kernel handles everything down to memory allocation and address translation, so everything left to do is to allocate some DMA-capable memory and stuff the required values into the device registers. These registers are described fairly well in the envy24 datasheet. Three values are required for a successful DMA operation:

- *Base Address* - tells the controller where to put the data in memory
- *Terminal Count* - tells the controller when to issue an interrupt request to the processor
- *Buffer Size* - tells the controller when to reset the running address pointer

One problem regarding the envy24's DMA transfer characteristics is the interleaved buffer structure. Each sample weights 4bytes (whereas 24bits are really used), and each burst transfers one sample per channel[3]. In the end, this leaves us with an interleaved buffer that has to be taken apart by the driver. This driver does buffer dissection and assembly in interrupt context, when servicing the device interrupt.

3.3.4 Interrupt Control and Handling

The Register `Terminal Count` decrements during the DMA transfers, and once it reaches zero an interrupt request is generated.

[2] For example, "Linux Device Drivers" by Alessandro Rubini and Jonathan Corbet

[3] The envy24 supports 8 analog and 2 digital channels of playback as well as 8 analog and 2 digital channels of recording + 2 "return path" channels, which come from the internal digital mixer. So we have 10 playback and 12 recording channels.

rat uses an RT-Linux ISR[4]. Otherwise the ISR would only be executed as an emulated interrupt, which do not have a suitable priority.

Everytime an interrupt occurs, the ISR queries the *Base Address* register to see how much data has already been transferred by the audio controller.

The contents of this register indicates where the last transfer has been made. By comparing this value to the "old" value from the last interrupt, the driver determines how much data was output or input since the last interrupt. Using this value, the buffer is interleaved or de-interleaved and the router's data-ready callback is called.

3.4 The Frame Grabber Driver

This driver is compatible with the BrookTree BT8(4,7)8[A] series of video decoder chips. These can be found on almost all Hauppauge WinTV cards and on some older Miro frame grabbers. It was originally written by Wayne E. van Loon Sr. and later ported to **rat** . The current version is intended a tech-demo only. It is not optimized in any way.

The BT8x8 chip has a very interesting mode of operation. It features an on-chip RISC engine, which is used to control the data flow to system memory. Apart from that, things are simple. There is an input mulitplexer, so theoretically 4 video sources can be selected via software. Unfortunately, mainstream TV cards only use a TV tuner and a single video input connected to the multiplexer.

Opposed to the envy24 described earlier, this PCI controller chip implements the PCI timing constraint registers *maximum latency* and *minimum grant*. It also incorporates a FIFO mechanism to survive bus stalls without data loss. In the very unlikely case of FIFO overflow and data loss, it can report this error to the host system, indicating that the PCI configuration and/or hardware has to be modified.

There are two types of DMAs involved when tranferring data from the frame grabber to memory.

Data **DMA** is controlled by the RISC program, which runs in the BT878 itself. The RISC program is located in main memory and it is also accessed via DMA. The driver has to set up a memory location where the RISC instructions are prepared. The instructions in this program specifies the memory location to which image data is written during grabbing.

The BT878 supports very flexible interrupt control. Every statement in the RISC program can be marked to generate an interrupt request. Additionally, numerous other conditions can be programmed to generate interrupts as well. These include video sync changes and error conditions. **rat** uses most of these interrupt sources to be informed about hardware status changes when they occur.

[4]Interrupt Service Routine

4 Plugins

4.1 The Plugin Interface

The Plugin Interface is called `LATIF`[5]. Every plugin must maintain a structure of general definitions, which contains the possibility to create and release instances of itself. These instances are managed by the `rat-router`.

In general, a plugin module has to provide the following features:

- A `rat_plugin` struct
- Mechanisms to create and destroy instances of itself
- Module Use Count Management

For most tasks, a plugin must also implement it's own runtime data structure.

4.1.1 Data Exchange

Payload data is always exchanged via `rat_buffer`s.

The `rat_buffer` is designed to be re-entrant and lock-free. It's *read* and *write* methods and safe to be preempted by complementary operations. They deliberately don't perform locking to enforce a clean system design and avoid priority inversions. This allows them to be used to pass data between the "Spaces", like from real-time space to kernel space and vice versa.

4.1.2 Constraints to Functionality of Plugins

The Plugin interface specifies a callback function, which the router executes whenever there is some data to process. The router will ensure that all needed input or output channels really contain data before the callback function is triggered.

The callback function **must** be RT-Safe.

This restriction makes it very difficult to use high level Linux Kernel resources (File I/O, network services and the lot). A trick can be used that allows plugins to leave the Real-Time Space. Linux Kernel Threads can be synchronized to Real-Time threads using Soft Interrupts (Tasklets). The Disk Reader and Disk Writer plugins use this method to pump data into kernel space and back again.

[5]Likely Advanced Tool Interface

4.2 Implemented Plugins

4.2.1 Linux File I/O

The `rat_reader` and `rat_writer` plugins are based on a common Linux VFS access module, which provides the File Access and Tasklet Synchronisation services.

Figure 4 shows how the escape from Real-Time Space is done.

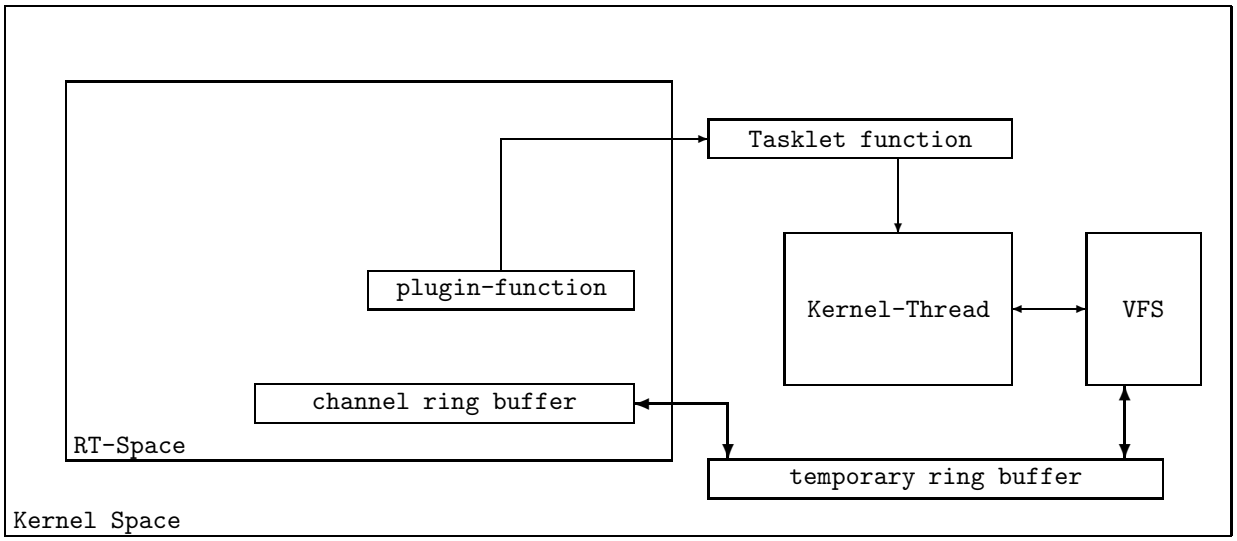

FIGURE 4: *Escape from Real-Time Space*

The plugin function is required to be RT-Safe. Thus, it cannot directly access the VFS, because this could put the calling process to sleep. The workaround uses a "buddy kernel thread" which is synchronized to the plugin function. Again, the plugin function can not directly interact with the kernel thread, because the Linux Kernel synchronisation primitives (like the wait queues used here) are likely to invoke the Linux Scheduler. To overcome this problem, a Tasklet is used to synchronize the Kernel Thread. The Tasklet can be scheduled for execution from real-time space.

The `rat_reader` and `rat_writer` plugins are used to read and write data from and to any mounted device. They copy the "real-time" data to or from a large temporary buffer, from where it can be lazily read or written.
This method is used to overcome the unpredictability of the Linux Kernel by simply buffering the data.

If the O_STREAMING kernel patch is present, this optimisation will also be used.

5 The Closing

5.1 A Look Ahead

The open system design makes **rat** suitable for a very wide field of applications. Everything from a simple (but highly reliable) audio player up to complex data aquisition and processing or even real-time control applications can be built on top of the **rat** -core.

The core system of **rat** is planned to become a live recording application. It will be used to record any number of audio and video channels simultanously.Future versions of **rat** will therefore take advantage of multiple processors.

6 List of Acronyms

`ADC` - Analog to Digital Converter
`ALSA` - Advanced Linux Sound Architecture
`BTTV` - BT 8x8 TeleVision card
`DAC` - Digital to Analog Converter
`DMA` - Direct Memory Access
`GNU` - GNU's Not Unix
`GPL` - GNU General Public License
`ISR` - Interrupt Service Routine
`LKM` - Linux Kernel Module
`MIDI` - Musical Instrument Digital Interface
`RAT` - Realtime Audio Tools
`RT` - Real Time
`S/PDIF` - Sony / Philips Digital InterFace
`VFS` - Virtual File System