PaRTiKle OS, a replacement for the core of RTLinux-GPL

S. Peiro, M. Masmano, I. Ripoll, and A. Crespo
Industrial Informatics and Real-Time Systems Group
Universidad Politecnica de Valencia, Spain
{speiro,mmasmano,iripoll,acrespo}@ai2.upv.es

Abstract

RTLinux-GPL[1] is an RTOS which uses a dual-kernel approach, that is, executing a real-time kernel (RTLinux-GPL itself) jointly with a general purpose OS (Linux). By using this approach, complex real-time application with both, hard and soft real-time requirements can be implemented easier.

However, the core of RTLinux-GPL is getting bloated and hard to maintain, with several out-of-date and not-working features. Moreover it's future is unclear regarding the adquisition of the RTLinux-GPL patent by Wind River. This facts forced us to consider either to reengage its code or to start a new kernel from scratch to replace it. Although the first option seems to be more suitable, since RTLinux-GPL is a mature and stable code, important design flaws (specifically, problems when a RTLinux-GPL application deals with signals and system calls) has headed us towards the second one.

In this paper we present this new core, called PaRTiKle, which is intended to be downward compatible with existing RTLinux-GPL applications.

1 Introduction

RTLinux-GPL is a hard Real-Time executive which uses an original approach to develop complex hard real-time applications in a fairly easy way: executing a hard RTOS (RTLinux itself) jointly with Linux in the same box. This approach enables splitting up a hard real-time application according to its criticality, the part with deadline requirements is run on the RTOS while the part with no "special" time requirements is executed on Linux.

RTLinux-GPL has been successfully used in a large amount of real-time applications such as:

- Satellite On-Board Data Processing.
- Real-time spacecraft simulation and hardware-in-the-loop testing.
- Real-time solution in petroleum industrys Cyber data acquisition simulation system.
- RTLinux Based Online Real Time Simulator of SPMSM Using the Block Pulse Approximation.
- etc.

However, several factors (among others: unmaintainability of the RTLinux-GPL's code, uncertainty about WindRiver's next step, etc.) have pushed us to implement a new real-time kernel, PaRTiKle.

In this new kernel we have meant to go a step further than in RTLinux-GPL, since PaRTiKle, unlike RTLinux-GPLhas been designed to be an RTOS on its own. That is, PaRTiKle does not have any knowledge either about virtualisation nor Linux but the virtualisation task is left to XtratuM [2, 3]. In fact, our current implementation can be executed in multiple execution environments: as a Linux regular application, on the top a bare machine and, eventually, as a XtratuM domain.

In this paper we describe the architecture of this new kernel, and compare it with RTLinux-GPL.

This paper is organised as follows: a brief overview of RTLinux-GPL and PaRTiKle is given in section 2 and 3 respectively. Section 4 compares the features of both kernels. Section 5 describes the process of configuring and compiling a PaRTiKle kernel jointly with an application. Section 6 describes the live-cd. Conclusions and future work can be found in the sections 7 and 8, respectively.

2 RTLinux-GPL brief overview

RTLinux-GPL [5] is a real-time operating system that uses a dual-kernel approach (figure 1). One of them is the Linux kernel, which provides all the

[1] RTLinux is a registered trademark of Wind River Systems, Inc.

features of a general purpose OS, whereas the other one is the RTLinux-GPL kernel, which fits hard real-time requirements. Applications are coded using the POSIX Threads (IEEE 1003.1b) API, and the application code is loaded into the kernel space by means of Linux Kernel Modules (LKM) infrastructure.

To provide real-time requirements RTLinux-GPL, takes over interrupts and executes the Linux kernel as the lowest priority task in the system. RTLinux-GPL executive itself is non-preemptible.

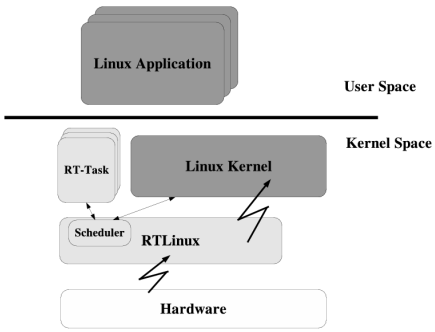

FIGURE 1: *RTLinux-GPL architecture*

3 PaRTiKle overview

PaRTiKle is a new embedded real-time operating system designed to be as compatible with the POSIX.51 [7] standard as possible.

PaRTiKle has been designed bearing the following ideas in mind:

- being as portable, configurable and maintainable as possible.
- support for multiple execution environments, allowing, thus, to execute the same application code (without any modification) to be executed under different environments (so far): in a bare machine, a Linux regular process and as a hypervisor domain.
- support for multiple programming languages, currently PaRTiKle supports Ada, C, C++, Java (the current support of this last language is only supported when GCC compiler version 3.4 is used).

3.1 Architecture

Figure 2 shows the architecture of PaRTiKle, as can be seen, the kernel can be split up in two well-differentiated and isolated parts, the kernel context (on the bottom) and the application context (on the top). Both parts interact via the PaRTiKle's system calls (entry point). The way this mechanism is implemented is architecture dependent. For example, in the x86 stand-alone version, it is implemented through segmentation, each context is held in a different segment. A system call is performed through a change of segment (`lcall` assembler instruction). An attempt from the application to access the kernel memory space triggers a processor exception which winds up the execution of the whole application.

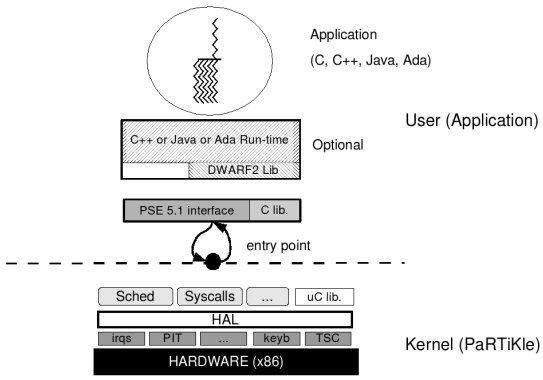

FIGURE 2: *PaRTiKle architecture*

The PaRTiKle kernel has been designed and implemented as a set of hardware-independent subsystems and a set of drivers. The drivers can only be accessed through the hardware abstraction layer (HAL). Briefly, these subsystems are:

- Scheduler: currently, the scheduler only implements a Rate-Monotonic (RM) scheduling policy, however, we expect to implement other policies in a close future.
- Physical memory management: this subsystem is in charge of managing the free available physical memory.
- Timing management: this subsystem is in charge of managing the hardware timer and clock, providing multiple virtual timers to the kernel and the application.
- Kernel libc: a minimal C library used only by the kernel. This library cannot be accessed by the application.
- System calls: as mentioned above, all kernel services are provided via a single entry point. Figure 3 shows how the system call mechanism is implemented, when an application invokes a system call, it is processed as a jump to Entry_point with the appropriate parameters. Once inside the kernel context, a call to the suitable system call is performed and the returned value is returned to the application.

On the other hand, in the application context, PaRTiKle provides a standard C library. And, op-

tionally, the correspondent language run-time support (currently, Ada, C++ and GCJ run-times are available, although the last one is out-of-date).

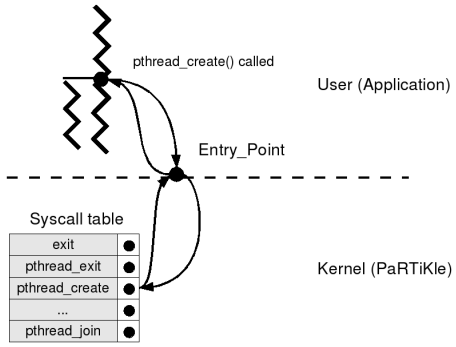

FIGURE 3: *PaRTiKle's system call mechanism*

3.2 Execution Environments

PaRTiKle has been designed to be run under several different execution environments. So far, three different execution environments are available, all of them for the x86 architecture: 1) on a bare machine, 2) as a Linux regular process and 3) as a domain of XtratuM [2], giving this last alternative the possibility of executing PaRTiKle jointly with another general purpose operating system (Linux so far), as RTLinux-GPL does.

1. On a bare machine: PaRTiKle is the only system executed in the system, it is in charge of managing the whole hardware. This environment is the best option for application with only hard-real time constraints, and small footprint.

2. As a Linux regular process: This environment is intented for testing purposes. The generated code is executed as a regular Linux process. PaRTiKle still has direct access to the hardware, however, real-time constraints are not guaranteed whatsoever.

3. As a XtratuM domain: XtratuM is an hypervisor that provides hardware virtualisation and enables the execution of several kernels (or runtimes) concurrently. PaRTiKle can be built to be XtratuM aware and then loaded using the XtratuM domain's loader `xmctl`:

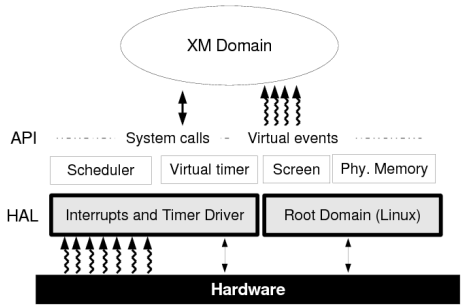

FIGURE 4: *PaRTiKle on XtratuM*

The figure 4 shows the functional blocks that compose XtratuM. The XtratuM's HAL is composed by an *interrupt* and a *timer* driver, plus two extra drivers *physical memory manager* and *screen* which do not deal with the hardware but with the root domain. On the top of these drivers we find the *virtual timer* and the *scheduler*.

The *virtual timer* multiplexes the hardware timer between the existing domains. The *scheduler*, at least in the first versions of XtratuM, implements a fixed priority policy. The interaction of XtratuM with the domains are performed via *system calls* and the virtual event dispatcher.

3.3 Language Runtime's support

Although PaRTiKle has been completely written in C and Assembly, it also supports the following languages' run-times:

- C++ using the `g++` compiler[2],
- Ada which uses `gnat` from GAP2005 [11].
- Java with the `gcj` compiler (currently out-of-date).

To provide a complete and advanced support for the exceptions handling mechanism (`try{ .. }catch` or `exception ... when`), the DWARF2 specification has been included.

Apart of the specific compiler needed for each language, PaRTiKle needs to be aware of the programming language in order to link correctly the application with the more appropriate runtime.

4 Comparative of RTLinux-GPL and PaRTiKle

There are some important diferences between both systems, they are discussed one by one and then summarised in table 1.

[2]Full support of all the C++ language features, but the standard library which is still incomplete.

Feature	RTLinux-GPL	PaRTiKle
Spatial isolation	X	√
Language support	Ada, Assembly, C, C++[3]	Ada, Assembly, C, C++, GCJ[4]
Driver support	Serial, mbuff, rtfifo, console, disk, CAN	Serial, mbuff, rtfifo, console, disk, RAM filesystem
User API	POSIX PSE51 (compatible)	POSIX PSE51 (compatible) + RTLinux extensions
Licence	Strict GPL	LGPL

TABLE 1: *Summary of the differences*

- **Spatial isolation**: There exists either a real or, at least, virtual isolation between the kernel space and the application space.
- **Language support**: The languages used to implement complex and robust real-time applications such as Ada and C++ are supported.
- **Driver support**: Support for common devices such as keyboard, console, serial, network cards, etc.
- **User API**: Application programming interface provided by the kernel, it can be standard such as POSIX, pSOS+ or a custom one.
- **Licence**: The legal licence covering the kernel and the implications of it on the end user application.

5 How to build a PaRTiKle application

In this section we explain how to build a naive `hello.c` application, from scratch. The application code is:

```
#include <stdio.h>

int main (int argc, char **argv) {
  printf ("Hello World!!!\n");
  return 0;
}
```

It is important to note that neither any non-standard C function has to be added in the code nor any non-standard C header has to be included.

The first step, supposing that we have already got PaRTiKle is to configure and compile it:

Configuration: The configuration interface of PaRTiKleis similar to the one used by Linux (in fact, the scripts have been taken from the Linux code). Execute the following command inside the partikle folder:

```
# make menuconfig
```

[3]Using the old sjlj mechanism.
[4]Using the new DWARF2 mechanism.

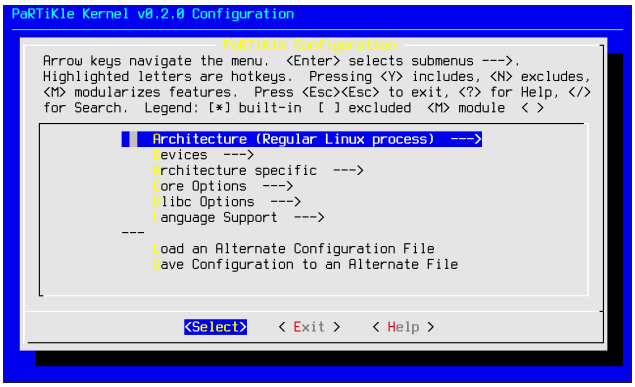

FIGURE 5: *PaRTiKle configuration menu*

The picture 5 shows a screenshot of the configuration menu. The configuration tool, among other things, permits to:

- Select the architecture and the execution environment, currently only the x86 architecture is available, nevertheless, support for ARM and PPC architectures is planned.
- Enable the different language run-time, by default only C is supported.
- Several core choises: stack default size, debug support, etc.

Building the kernel The next step, once PaRTiKle has been configured, is to compile it, this is done executing the command `make` in the partikle folder:

```
# make
>> Detected PaRTiKle path: /usr/src/partikle/
>> Building PaRTiKle utils: done
>> Building PaRTiKle kernel [xm_i386]: done
>> Building PaRTiKle user libraries: done
```

The result of this compilation is two files: `partikle_core.o` which holds all the kernel related code and `ulibc.a` which holds the user code. If any language run-time support was selected during the

configuration stage, it is compiled during this stage and can be found in its correspondent directory.

Building the application Eventually, the last step is to compile the application code and then link it with PaRTiKle's object files. The result is a single object file which contains both the kernel and the application code.

PaRTiKle includes wrappers for the `gcc` and `ld` commands (`pgcc` and `ldkernel` respectively) to ease this step.

Therefore to compile our `hello.c` example the next command should be executed:

```
# pgcc -O2 -c -o hello.o hello.c
# ldkernel -o hello.prtk hello.o
```

Some additional examples can be found under the folder `user/examples`.

Running the resulting application depends on the selected execution environment, for example, in the case of PaRTiKle+XtratuM, the application must be loaded as a XtratuM domain:

```
# xmctl start
insmod /usr/src/xtratum/xm.ko
# xmctl load hello.prtk
Loading the domain "hello.prtk" ... Ok (Id: 1)
# xmctl unload hello.prtk
Unloading the domain "(1)" ... OK
```

Figure 6 summarises the whole process.

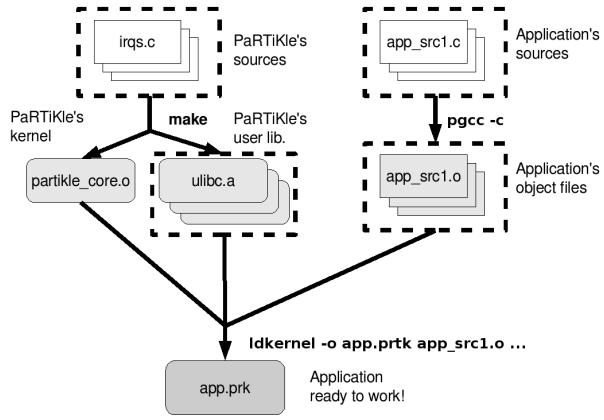

FIGURE 6: *PaRTiKle building*

6 Slax-rtl: *PaRTiKle live-cd*

To ease the utilisation and assessment of PaRTiKle+XtratuM, we have built a live-cd iso image with a complete development environment (see subsection 9 for further details).

These were the reasons that pushed us to select a SLAX distribution to create Slax-rtl:

- **Modularity**: the main reason to choose SLAX was its modularity since all the required changes for the filesystem respect an official SLAX iso image can be packed and compressed into a SquashFS file.
- **Easiness of modification**: modularity eases the organisation of changes, addittion of programs and customisation (remastering) the distributed iso; assuming that those changes are independent and do not interfere with the system.
- **Automated iso creation**: the construction of a SLAX iso, relies on the linux-live [12] package of utilities, which automate all the steps needed to build a complete working system, from kernel configuration and compilation, to the initial RAM filesystem creation.

6.1 Modifications in Slax-rtl

The modifications with respect to the official SLAX v5.1.8 have been:

Firstly, a XtratuM aware kernel and its kernel modules were configured and compiled. After this, an initial RAM filesystem was created to hold the minimum amount of modules and commands required to start up the SLAX's linux-live system. Then, the development tools and project source trees needed were packaged on compressed as SquashFS modules and included in the iso image:

- `Official_Development_module_5_1_4.mo`: the official development tools: `make/gcc/ld`
- `Subversion_1_4_0.mo`: subversion tools to update the project source trees,
- `rtlinux.mo`: the project source trees: `/usr/src/{linux,xtratum,partikle}`, with a compiled XtratuM's module: `xm.ko` ready to use.

At boot time the modules are mounted over the system's root directory: `/` by using the UnionFS file system, thus populating the filesystem with the slax-rtl changes.

7 Conclusions

In this paper we present an assessment of a new hard real-time kernel, PaRTiKle, which, when used jointly with XtratuM, provides all the benefits RTLinux-GPL but without, as far as we think, its drawbacks.

Besides, PaRTiKle can also be executed as a stand-alone kernel and as regular Linux process. Due to our previous experience adding language support in RTLinux-GPL [14, 16, 15], it has been quite straightforward implementing complete support of

the following languages in PaRTiKle: Ada, C++, Java.

Since, PaRTiKle has been designed as a replacement of RTLinux-GPL, it offers a compatibility layer with all POSIX non-portable function (such as `pthread_make_periodic_np`, `pthread_delete_np`, `pthread_wait_np`). Furthermore, the people from DSLab have already sent us an implementation of shared memory and RT-FIFOS for XtratuM [4].

8 Future work

Although, a lot of work has been done, the kernel is far from being finished. So far, our priorities are:

- Porting PaRTiKle to other architectures, ARM and PPC. Both of them, widely used for embedded systems.
- Implementing a low-level standard interface between the kernel and the drivers to ease the support of new hardware. The Real-Time Driver Model (RTDM) seems to be a solid candidate but we have still to compare it with other existing possibilities.
- Porting the COMEDI (Linux Control and measurement device interface) [13] drivers.
- Implementing some additional drivers depending on the new emerging necessities.
- Etc.

9 Project Information

PaRTiKle is being actively developed by the Real-Time Systems Group of the Universidad Politecnica de Valencia, Spain. Although PaRTiKle and XtratuM are different projects, both of them are tightly related, and they are being developed in parallel using the same subversion tree.

A wealth of information about PaRTiKle and XtratuM can be found at its respective web-sites:

- PaRTiKle: http://www.e-rtl.org/partikle
- XtratuM: http://www.xtratum.org

on them you can find information about tracking, downloads and development versions of the projects:

- tracker: https://www.gii.upv.es/trac/rtos/
- repos: https://www.gii.upv.es/svn/rtos/trunk/
- slax-rtl iso: http://www.xtratum.org/node/31

References

[1] Miguel Masmano, Ismael Ripoll, Alfons Crespo, Audrey Marchand, *Nanokernels for multidomain support* Deliverable: D-EP2 Frescor, Responsible: UPVLC

[2] M. Masmano; I. Ripoll; A. Crespo, *An overview of the XtratuM nanokernel*, Universidad Politcnica de Valencia, Spain

[3] Nicholas McGuire, *XtratuM Hardware Initialization*, Distributed & Embedded Systems Lab, Lanzhou University, China

[4] Shuwei Bai, Yiqiao Pu, Kairui She, Qingguo Zhou, Nicholas MC Guire, Lian Li, XM-FIFO: *Interdomain Communication for XtratuM* , Distributed & Embedded Systems Lab, Lanzhou University, China

[5] Michael Barabanov, *A Linux based Real-Time Operating System*, Master's Thesis, New Mexico Institute of Technology.

[6] Victor Yodaiken and Michael Barabanov, *RTLinux Version Two*, VJY Associates LLC

[7] POSIX Standarization Comitee, *PSE.51 Minimal Real-time System Profile*, POSIX

[8] Yoshinori K. Okuji. *GRand Unified Bootloader*, available at http://www.gnu.org/software/grub/

[9] *The EtherBoot Project*, available at http://www.etherboot.org/

[10] Yoshinori K. Okuji, Bryan Ford, Erich Stefan Boleyn, and Kunihiro Ishiguro. *The Multiboot Specification*, available at http://www.gnu.org/software/grub/

[11] *GAP2005 GNAT/GPL*, from AdaCore co. available at https://libre.adacore.com/

[12] Tomas Matejicek et al. *The SLAX Live CD*, available at http://www.slax.org/ http://www.linuxlive.org

[13] David Schleef et al. *Linux control and measurement device interface (COMEDI)*, available at http://www.comedi.org/

[14] M. Masmano, Jorge Real, Ismael Ripoll, and A. Crespo *Running Ada on Real-Time Linux*, 8th International Conference on Reliable Software Technologies - Ada-Europe 2003. Lecture Notes in Computer Science (Vol 2655).

[15] M. Masmano et al. *Ocera RTLinux CC*, available at http://www.ocera.org/download/components/WP5/rtlcc-0.1.html

[16] M. Masmano, Ismael Ripoll, Jorge Real, and A. Crespo *Early Experience with an Implementation of Java on RTLinux*, Sixth Real-Time Linux Workshop. 2004.