

XtratuM overview

Miguel Masmano, Ismael Ripoll & Alfons Crespo.

Abstract

This white paper presents a new nano-kernel, called XtratuM¹. It is a nano-kernel designed to meet real-time requirements: fast, compact (small), portable, simple and predictable.

Introduction

XtratuM is an open source nanokernel (or better a picokernel) that goes beyond the classical ideas of nanokernels (www.xtratum.org). Basically, XtratuM is just two simple device drivers: interrupt and timer. Usually, operating systems are internally structured as a set of building blocks (see Figure 1): user interface layer, a set of drivers, memory manager, virtual filesystem infrastructure, network stack, etc. In this scenario, XtratuM can be considered as a small part of the lowest operating system layers.

The practical consequence of controlling interrupt and timer hardware is that XtratuM has the full control of the system. That is, XtratuM operates as a supervisor system which is executed at the highest priority. XtratuM handles a "guest operating system" (also referred as *domain*) in a similar way than any operating system manages a process: the OS can create, suspend, kill, etc. each process.

XtratuM provides a simple and uniform API to use interrupts and timers regardless the real available hardware devices through virtualisation techniques. Therefore, XtratuM can execute several domains concurrently. There are other alternatives as ADEOS and Fiasco provides also similar functionalities.

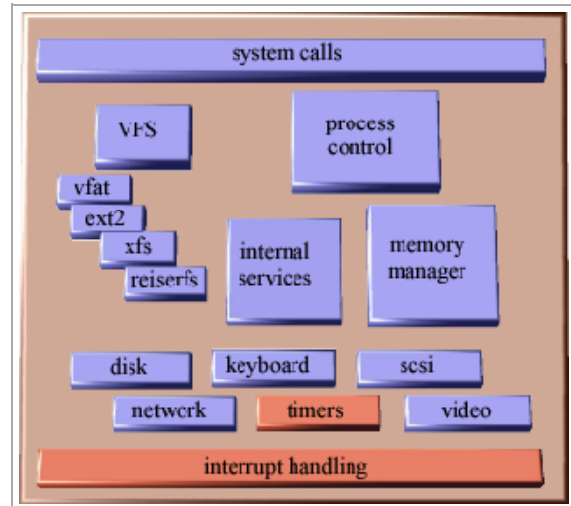


Figure 1: Classical operating system internal structure (simplified).

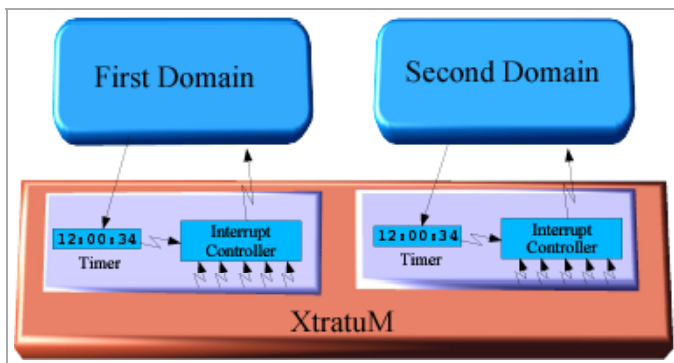


Figure 2: XtratuM shares hardware to different domains.

currently it is used in the ADEOS project. Since XtratuM takes over the interrupt and timer systems it is possible to implement any scheduling policy among domains. And going one step further, since it is possible to intercept processor exceptions, it is also possible to detect domain errors and act accordingly. This mechanism can be used to have fully application redundancy, including operating system redundancy. For example, it will be possible to run two different operating systems (RTLinux/GPL and MaRTE OS), both running the same application but programmed with different language ("C" and Ada95); one operating system performs the control of the physical devices while the other is monitoring the activity; if the first system fails, then the second can take the control.

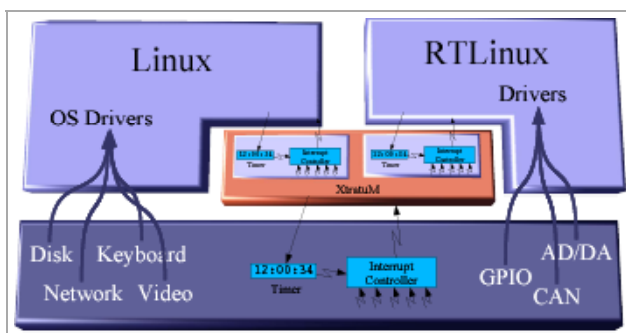


Figure 3: Running Linux and RTLinux/GPL over XtratuM.

Although it may seem that XtratuM is a complex and heavy layer of code that will delay (since all interrupts pass thru it) the execution of the guest OS, the overhead is quite low. In fact, the complexity of XtratuM is mostly related to the hardware programming rather than complex and costly interrupt processing. It has been designed to be fast and to have a bounded response time in order to be used in hard real-time systems.

The most exciting feature of XtratuM is the capability to share the same hardware among several operating systems running concurrently. It makes possible to run two or more operating systems at the same time in the same computer. This feature is not new, it was used in the IBM 360 in the seventies, and

In order to run several domains concurrently, each domain has to be adapted (ported) to the XtratuM infrastructure. In particular, the boot sequence of each operating system has to be removed because XtratuM will take care of it; and the interrupt and timer drivers have to be replaced by calls to the XtratuM API.

As many other good things, XtratuM has been developed using Linux. That is, Linux is the development host, and also the target host. Linux kernel modules is one of the better development platforms that can be used when you have to deal with low level hardware programming. Linux provides a running environment and a very easy method to run supervisor (ring level zero) code. XtratuM has been developed in a running Linux, therefore the porting of Linux to XtratuM was the first task done during its development. The porting was done in a way that permits to convert a regular running Linux to a XtratuM domain and back. Since the aim of XtratuM is **simplicity**,

was done in a way that permits to convert a regular running Linux to a XtratuM domain and back. Since the aim of XtratuM is **simplicity**,

Linux has been ported with the minimum possible changes (the patch is only 13Kb).

XtratuM has been developed as a part of the OCERA project². In OCERA we used RTLinux/GPL as the starting point for our developments. Therefore we had worked with RTLinux/GPL intensively, and some of the developments directly related to RTLinux/GPL code has been integrated into the "official RTLinux/GPL" release (maintained by Nicholas Mc Guire). From the technical view, RTLinux/GPL is a good RTOS (see the RTLinux vs. RTAI comparative), but a small part of the RTLinux/GPL code is covered by a software patent (U.S. Patent No. 5,995,745). A nice³ consequence of running RTLinux/GPL over XtratuM is that all the patented code is removed, therefore the XtratuM+RTLinux/GPL is patent free.

Applications examples

Next are some examples of how XtratuM can be used by an operating system:

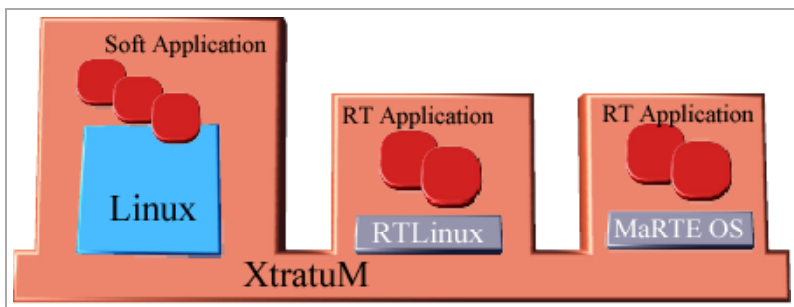


Figure 4: XtratuM running redundant systems.

Figure 4 represents a system where Linux is the background operating system; RTLinux/GPL playing the role of master real-time operating system and running the controlling application; and MaRTE OS also running the same application (but coded by a different developers group and using a different programming language Ada) but the application does not effectively send the actions to the hardware but compares its our results with those generated by the RTLinux/GPL domain. In case of a mismatch in the actions computed by both applications, or if a domain raises an exception, XtratuM can

stop the buggy domain. Note that in order to know which is the faulting domain it might be possible to need a third domain.

XtratuM, when compiled with booting code (which will be developed soon) can be used to run the real-time operating system in several hardware processors with minor code changes. We are currently working on the ARM (Xscale) porting of XtratuM.

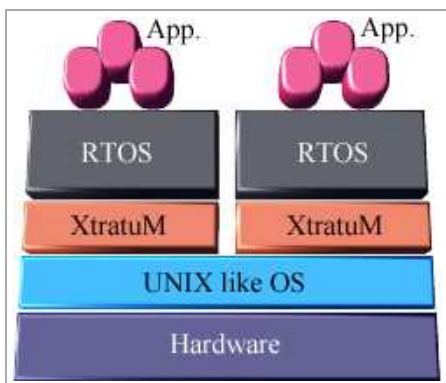


Figure 6: Using XtratuM to test the operating system, or application.

Another practical use of XtratuM framework is to run your RTOS on top of a Linux system as a regular Linux process. The idea is to compile the guest operating system as a normal ELF executable and then run it the same way as `ls` or `bash` do. In this scenario, XtratuM used the POSIX signals and timers facilities provided by the host operating system as if they were interrupts and timers devices.

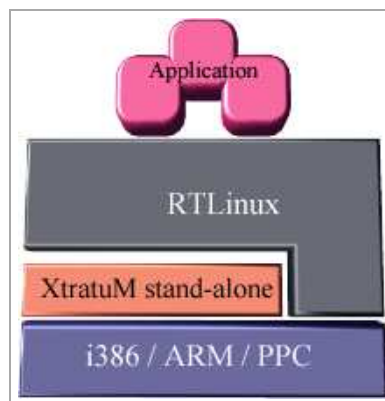


Figure 5: Stand alone system.

It is a very restricted and unrealistic system that can only be used for testing and to speedup code development. It can also be used for teaching purposes.

Future guidelines

The main objective of XtratuM is to simplify operating system portability so that a XtratuM aware OS can be easily ported to any environment. For example, running the operating system as a stand-alone kernel or concurrently with other OSs or even on another processor architecture. Parallel to this main objective is the development of a framework which provides spatial and temporal isolation among domains.

Although we plan to add to XtratuM many new features, it is important to not that we are highly concerned with the requirements imposed by realtime applications. Our main research interests are around real-time issues (scheduling, control systems, Ada95, etc.). Therefore, any new feature added will be carefully considered and only included in the core XtratuM if: 1) it does not increment the timing overhead and the required memory space is very low; or 2) it greatly improves the functionality. Otherwise, new functionalities will be added as optional, and the end user will be able to select them at configuration time.

We believe that a new feature that is hard to explain, and the end user is not able to understand it in all its extent, is a bug. And so, it has to be removed.

Current status

Not all the feaures previously described are ready to be used... XtratuM is just a few months old!

ist of features (15 February 2005)

- Runs as a Linux module.
- Use of APIC and PIT timers.
- RTLinux/GPL ported to XtratuM infrastructure.
- MaRTE OS ported to XtratuM (experimental).
- Domain memory protection optional (experimental).
- Shared memory between domains (experimental).
- Framebuffer, mouse and keyboard emulation (prove of concept).
- Microwindows (now called Nano-X) ported to the frame buffer (prove of concept).

Right now it is hard to use, mostly due to lack of documentation (README, INSTALL, manual, etc.). We have been working on the code, writing prove of concept prototypes. We wanted to be sure that XtratuM will be able to have all the features we want. Once the code is released, it is harder to undertake large changes and modifications of the code and API.

There are lot of things to be done:

1. Define a clean and simple API.
2. Write documentation.
3. Implement domain scheduling policies.
4. Port XtratuM to other processors (ARM).
5. Port of more OS guests on top of XtratuM.

Who are we?

We are a research group (Real-Time Systems Group) that has been working with real-time kernels since 1996. Most of our research and developments have been done using RTLinux/GPL code (see RTPortal). Moreover we have been contributed, and will continue contributing, with the development of RTLinux/GPL. We have also experience with MaRTE OS, which is a RTOS mainly written in ADA95 which follows the POSIX standards.

Among other achievements, our research group has:

- Ported RTLinux/GPL to be run independently from Linux, that is, RTLinux/GPL is a complete bootable system.
- Ported the MaRTE OS kernel (mostly written in Ada95) to run jointly with Linux, using the RTLinux/GPL hardware virtualisation layer (it is GPL code covered if a FSM Labs, Patent).
- Ported MaRTE OS kernel to run on top of Linux as a regular process. Although it does not provide real time performance, this porting can be used for teaching and also during the first developments stages of real application (testing only the functionality of the application).
- Contributed with RTLinux/GPL adding POSIX timers, POSIX signals, POSIX barriers, POSIX message queues.
- One of the recent improvements of the POSIX standard is the tracing facility. This extension of the standard has also been implemented in RTLinux/GPL, which makes RTLinux/GPL one of the first RTOS that provides it.
- We have designed and implemented the first real-time memory allocator, called TLSF. It provides truly bounded and fast operation. Also the last release TLSF-1.4 has similar fragmentation as that caused by one of the best existing allocators: Douglas Lea's malloc.

Summary

RTLinux/GPL is a compact, mature and complete operating system developed to support POSIX real-time interface as native API. Since an operating system is nothing more than a library of functions (in fact RTLinux/GPL is linked with the user application as any conventional executable does), we can manage the operating system like a black box and work on the hardware dependent parts. XtratuM will add, in a transparent way, the portability and flexibility to any guest operating system, and in particular to real time operating systems.

Most of the claims done in this documents have been pre-alpha tested or we have enough experience to be 99% confident that it can, and will, be done. Sometimes it takes more time to prepare the installation scripts and write the documentation than the development of the code itself.

Footnotes:

¹ Stratum (WordNet dictionary):

- 1) one of several parallel layers of material arranged one on top of another (such as a layer of tissue or cells in an organism)
- 2) an abstract place usually conceived as having depth; "a good actor communicates on several levels"; "a simile has at least two layers of meaning"; "the mind functions on many strata simultaneously" [syn: level, layer]

² OCERA is IST 35102 European project.

³ As researchers that use the scientific method in our daily work (think > analyse > search other college results > compare > improve > publish improvements > back to think), we think that software patents are harmful to society in general and in particular to research, because it stops innovation and also gives all the exploitation rights to just a few people (companies) in the world. Software patents are the *excuse* that can use big companies to sue any other competitor (small new companies).