Tesina de Máster en Automática e Informática Industrial

Partitioned System with XtratuM on PowerPC

Author: **Rui Zhou**
Advisor: **Prof. Alfons Crespo i Lorente**

December 2009

Contents

Abstract

Nowadays, the diversity of embedded applications has been developed into a new stage with the availability of various new high-performance processors and low cost on-chip memory. As the result of these new advances in hardware, there is a great interest in enabling multiple applications to share a single processor and memory, which requires that each application must be in spatial and temporal isolation and protected from other applications in the same system. Meanwhile, this kind of isolation usually calls for the implementation at the level of safety/security-critical to guarantee the healthy running of the whole system.

To realize the idea of multiple applications sharing single resource, from the point view of software, one approach is to implement the partitioned system. This concept originates from safety-critical areas such as MILS (Multiple Independent Levels of Security) architecture and ARINC 653 (Avionics Application Software Standard Interface) standard. It has been utilized a lot in real-time applications and safety-critical systems, etc.

Aiming at the embedded applications, XtratuM is a real-time hypervisor originally built on x86 architecture. It is designed and implemented consulting the concept of partitioned system. As a hypervisor, XtratuM is located between the partitions and hardware. Partitions cannot directly access hardware but only via XtratuM. XtratuM enables partitions to execute simultaneously in spatial and tempal isolation without interfereing each other but sharing the same hardware. The main work in this thesis covers the implementation of XtratuM in PowerPC architecture. And some preliminary benchmarks have been carried out.

This thesis is organized as follows: Chapter 1 shows the ideas and similar ideas about partitioned system, Chapter 2 describes XtratuM hypervisor, Chapter 3 introduces PowerPC, Chapter 4 expresses the main ideas for porting XtratuM to PowerPC, Chapter 5 states the implementation in detail, Chapter 6 shows the benchmark results and Chapter 7 concludes the work with future development.

1. Introduction

Looking through the trend of technology development and application, embedded system has been developing in past years with a great leap. People's everyday life now has become indispensable with embedded applications, including cell phone, GPS, media player, game player, as well as e-book reader recently coming into vogue. The diversity of embedded applications has been developed into a new stage with the availability of various new high-performance processors and low cost on-chip memory. As the result of these new advances in hardware, there is a great interest in enabling multiple applications to share a single processor and memory. To facilitate such a model the execution time and memory space of each application must be isolated and protected from other applications in the same system [1]. At the same time, there is a todo list to implement systems to take advantage of rapidly increasing COTS (Commercial Off-The-Shelf) processing resource, including reducing system obsolescence by increasing software portability, enabling new software applications to be integrated into legacy systems, reducing life-cycle costs and increasing functionality for "new"systems, and enabling the integration of applications at multiple levels of criticality/security on the same processing resource, etc [2]. Consequently, the software implementation is another essential issue to achieve this goal. Kinds of real-time applications have been integrated with real-time systems for a long time and in a broad range. And the emergence and advance of ICT (Information and Communication Technologies) have introduced new ideas for real-time systems, which focus on dependability, low cost, and integration of different real-time applications [3]. This new trend is obviously suitable for the new features of hardware, and especially, the diverse requirements of embedded applications.

To realize the above idea of multiple applications sharing single resource, from the point view of operating system, one approach is to implement the partitioned system. Away from the federated and distributed implementation for real-time systems, partitioned system makes use of multiple COTS processor modules and software components in building comprehensive real-time systems. It allows the real-time applications to be merged into an integrated system [3]. The concepts or similar concepts of partitioned system can be referred to different materials.

1.1. MILS

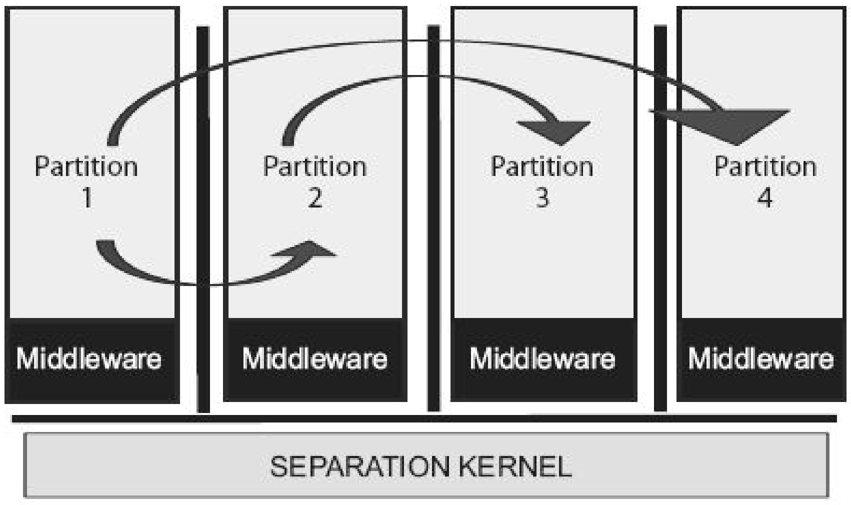


Figura 1: MILS architecture and information flows [4]

Considering partitioned system from the point of view of security, the MILS (Multiple Independent Levels of Security) architecture (Figure 1) has been proposed as a solution to meet the needs for critical information assurance. This concept originated in the early 1980s [5]. A separation kernel was proposed to divide memory into partitions using the hardware memory management unit and allow only carefully controlled communications between non-kernel partitions. This allows one partition to provide a service to another with minimal intervention from the kernel [5]. With this kind of separation, traditional operating system services that previously ran in privileged (i.e., supervisor) mode such as device drivers, file systems, network stacks, etc., now run in non-privileged (i.e., user) mode. The separation kernel provides very specific functionality, so the security policies that must be enforced at this level are relatively simple. The security requirements for the separation kernel can be categorized by four foundational security policies [6]:

- **Data Isolation.** Information in a partition is accessible only by that partition, and private data remains private.
- **Control of Information Flow.** Information flow from one partition to another is from an authenticated source to authenticated recipients; the source of information is authenticated to the recipient, and information goes only where intended.
- **Periods Processing.** The microprocessor and any networking equipment cannot be used as a covert channel to leak information to listening third parties.
- **Fault Isolation.** Damage is limited by preventing a failure in one partition from cascading to any other partition. Failures are detected, contained, and recovered locally.

As a componentized architecture based on a COTS separation kernel that enforces strict communication and partitioned process execution, MILS supports multiple levels of security communication, security policy composition, and modular design so that critical components are able to be evaluated at the highest levels to ensure secure and safe operation. In MILS, applications do their processing and enforce their own security policies in user-mode partitions. They can only access the memory that has been explicitly allocated for each partition, and can only communicate with each other through paths that have been configured when the system was generated. Unless explicitly authorized, application partitions cannot access hardware directly, which reduces the possibility of issuing errors by hardware. The control of information flow and data isolation guarantees the effectiveness of application-level security-policy enforcement. Information originates only from authorized sources, is delivered only to the intended recipients, and the source is authenticated to that recipient, so the application developer is empowered to build his or her own application satisfied with NEAT [6][7]:

- **Non-bypassable.** A component can not use another communication path, including lower level mechanisms to bypass the security monitor.
- **Evaluatable.** Any trusted component can be evaluated to the level of assurance required of that component. This means the components are modular, well designed, well specified, well implemented, small, low complexity, etc. They are small and simple enough to enable rigorous proof of correctness through mathematical verification.
- **Always Invoked.** Each and every access/message is checked by the appropriate security monitors/functions, i.e., a security monitor will not just check on a first access and then pass all subsequent accesses/messages through.
- **Tamperproof.** The system controls ”modify” rights to the security monitor code, configuration and data. Security functions and their data cannot be modified without authorization, either by subversive or poorly written code.

For MILS, non-NEAT security policy enforcement is not effective. Although other operating systems have offered some form of non-bypassability and tamperproof functionality, MILS provides NEAT for the first time in a COTS package that is formally modeled and mathematically verified at a high assurance level.

1.2. ARINC 653

Later, with the emergence of ARINC (Avionics Application Software Standard Interface) Specification 653, there is an API definition of APEX (APplication EXecutive) that supports space and time partitioning of applications. ARINC 653 is a software specification for space and time partitioning in safety-critical avionics real-time operating systems. As a part of ARINC 600-Series Standards for Digital Aircraft and Flight Simulators, it allows to host multiple applications of different software levels on the same hardware in the context of IMA (Integrated Modular Avionics) architecture [8]. Here each partition refers to a separate application and there is dedicated memory space for each partition thereby providing space partitioning (Figure 2). APEX provides a common logical environment that enables the execution of multiple independently developed applications to execute together on the same hardware. Spatial partitioning involves the strict segregation of application memory areas (containing code and/or data) so as to prevent faults occurring in one function from propagating to other functions, i.e. fault containment. Each partition owns a unique address space to store is code and data. Space partitioning means that partitions have separated address space and cannot read, write or execute data from other address spaces. This mechanism isolates partition memory and prevents any modification from other partitions. Also, it cannot access to other address spaces. Meanwhile, APEX provides a dedicated timeslice for each partition to support time partitioning. Temporal partitioning defines the decomposition of an application's runtime activity and its allocation to pre-determined time slots. Time isolation means that each partition has at least one timeslice to execute their threads and a partition cannot overrun its time budget. The flexibility of APEX together with the standardized definition of space and time partitioning has led to the proliferation of the use of ARINC 653 beyond IMA to include many other safety critical applications. Therefore, ARINC 653 systems are composed of such partitions that exactly are software partitions (Figure 3). Based on these partitions, ARINC 653 supports a multitasking environment in each partition through a well-defined set of process communication mechanisms and preemptive priority based scheduling [1][9][10].

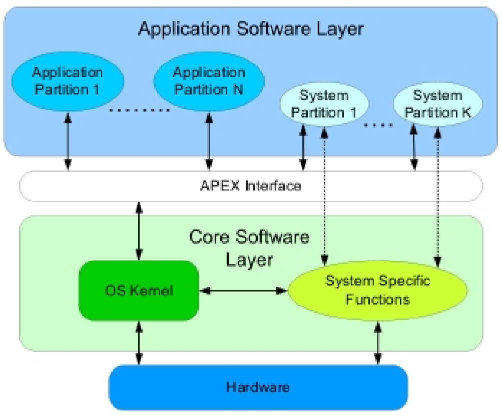


Figura 2: Standard ARINC 653 architecture [11]

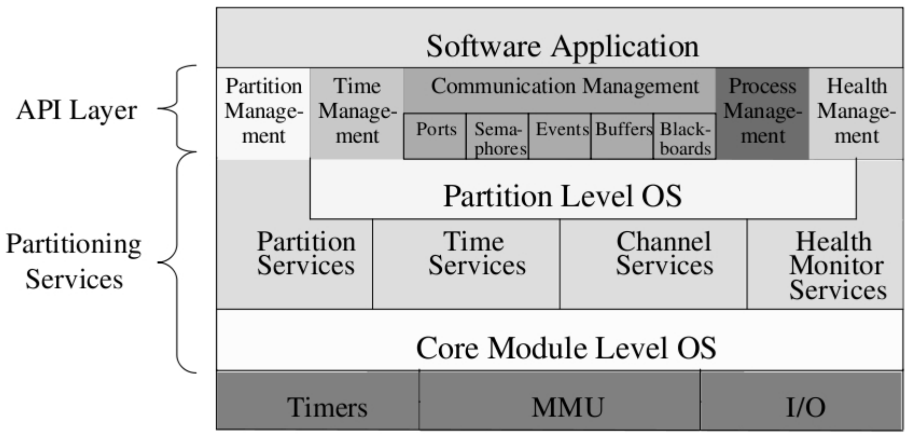


Figura 3: ARINC 653 defines software partitioning [2]

ARINC 653 allows software to be ported from one platform to another. To maximize the cost-benefits of re-use, it is necessary to reduce the verification and validation effort required when software is re-used. The concept of partitioning is utilized to address this issue. A partition is described as a program, composed by code, dates with a single, isolated memory address space [12]. The software within a particular partition can only address a pre-allocated area of memory, and can only execute within pre-determined time slices. The partitioning mechanism ensures that software errors in one partition do not propagate into other partitions. By employing the partitioning mechanism applications can be partitioned into loosely coupled components. It can be shown that failure of a particular component does not adversely affect the critical functions of the application, which has advantages both during initial design and when the component is re-used. As long as it is demonstrated that updated software can be located in a partition, meeting the time and space requirements, software can therefore also be ported to a different partition without repeating the verification and validation processes [13].

ARINC 653 proposes legal channels through which partitions can communicate. These legal channels are supervised by the kernel and no other channel can be used for inter-partition communication. This functionality ensures data isolation and prevents data propagation across partitions. Once a fault in a partition is raised, a dedicated program recovers it to keep the system in an error-free state. The fault can be caught at different levels (thread level, partition level or kernel level). ARINC653 defines three levels of faults: kernel (called module), partition and thread (called process). ARINC653 defines all possible faults, errors or exceptions that could happen in the system, for each of which, the system designer associates a recovering procedure. Therefore, an ARINC653-compliant system should ensure that resources are separated across partitions so a partition cannot use all resources, occurring faults can be recovered, and faults cannot be propagated outside a partition [10].

1.3. PikeOS

PikeOS (Figure 4) is a microkernel-based and para-virtualization real-time operating system presented by SYSGO AG. It is targeted at safety and security critical embedded systems. If several programs having different criticality levels are to coexist in one machine, the underlying OS must ensure that they remain independent. And resource partitioning is a widely accepted technique to achieve this. PikeOS provides a partitioned environment for multiple operating systems with different design goals, safety requirements, or security requirements to coexist in a single machine, which is based on the combination of resource partitioning and virtualization to provide a platform. PikeOS Separation Microkernel is fully compliant with the MILS separation kernel architecture. Also, it provides a built-in Health Monitoring Features which implements all features described in ARINC 653, especially the API of APEX. Its VM (Virtual Machine) environments are able to host entire operating systems, along with their applications. Since PikeOS uses para-virtualization, operating systems need to be adapted in order to run in one of its VMs. Application programs, however, can run unmodified. A number of different operating systems have been adapted to run in a PikeOS VM. Among them are Linux, several popular real-time operating system APIs, as well as Java and Ada runtime systems [14][15][16].

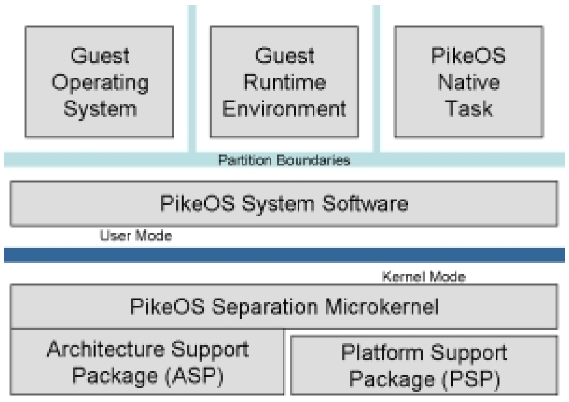

Guest Operating System
Guest Runtime Environment
PikeOS Native Task
Partition Boundaries
PikeOS System Software
User Mode
Kernel Mode
PikeOS Separation Microkernel
Architecture Support Package (ASP)
Platform Support Package (PSP)

Figura 4: PikeOS architecture [15]

Since each VM is guaranteed by partitions to have its own, separate set of resources, programs hosted by one VM are independent of those hosted by another. This allows for legacy (e.g. Linux) programs to coexist with safety-critical programs in one machine. Unlike other popular virtualization systems, PikeOS features not only separation of spatial resources, but also strictly separates temporal resources of its guest OSes. This allows for hard real-time systems to be virtualized, while still retaining their timing properties. Spatial and temporal resources are assigned statically to the corresponding individual VMs by PikeOS System Software. Together with PikeOS Separation Microkernel, this system software forms a minimal layer of globally trusted code, due to which, the system is suited for safety-critical projects requiring certification according to prevalent standards [16].

1.4. ADEOS

In another way, ADEOS (Adaptive Domain Environment for Operating Systems) presents the similar idea to partition as domain. ADEOS is a nanokernel HAL (Hardware Abstraction Layer) that operates between hardware and operating system. It is intended to run several kernels together, which makes it similar to virtualization technologies. It can be loaded as a Linux loadable kernel module to allow another OS to run along with it. In fact, it was developed in the context of RTAI (Real-Time Application Interface) to modularize it and to separate the HAL from the real-time kernel. It provides a flexible environment for sharing hardware resources among multiple operating systems, or among multiple instances of a single OS, thereby enabling multiple prioritized domains to exist simultaneously on the same hardware [17].

To enable multiple operating systems to run on the same system, there are two categories of existing solutions: one is simulation-based and provide a virtual environment for which to run additional operating systems, and another is to use a nano-kernel layer to enable hardware sharing. ADEOS addresses the requirements of both categories by providing a simple layer inserted beneath an unmodified running OS and thereafter provides the required primitives and mechanisms to allow multiple OSes to share the same hardware environment. ADEOS architecture (Figure 5) is generally designed to ensure equal and trusted access to the hardware, and it must take control of some hardware commands issued by the different OSes, but must not intrude too much on the different OSes' normal behavior. In ADEOS, each OS is encompassed in a domain over which it has total control. This domain includes a private address space and software abstractions such as processes, virtual memory, file systems, etc. As will be covered shortly, these resources do not have to be exclusive since OSes that recognize ADEOS and are able to interact with it may be able to share resources with or access the resources of other domains [18].

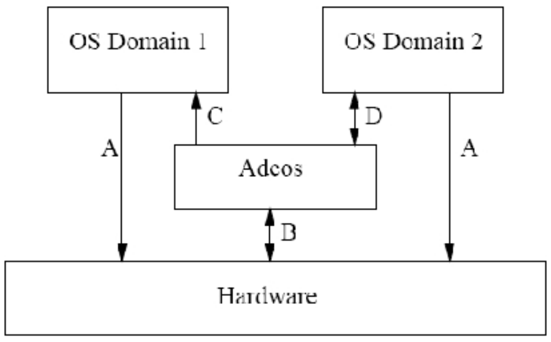


Figura 5: ADEOS architecture [18]

In ADEOS there are 4 broad categories of communication methods. Category A is a general usage, e.g. memory access, hardware access and traps, of the hardware made by the different domains, which functions as if ADEOS was not present. Category B involves ADEOS receiving control from the hardware because of a software or hardware interrupt, and all hardware commands issued by ADEOS to control the hardware. Category C involves invoking a domain's interrupt handler upon the occurrence of a trap while providing it the required information regarding the interrupt. Category D involves bidirectional communication between a domain and ADEOS. It is only possible when the domain is aware of the presence of ADEOS. This communication model provides the maximum usage of ADEOS' capabilities as it can be asked by one domain to grant complete or partial access to resources belonging to other domains or even to grant it priority over other domains during interrupt handling [18].

A domain in ADEOS mainly deals with operating system for sharing resources. While in partitioned system, the existence of partitions mainly focuses on safety-critical issues besides sharing resources. And a partition could not only be an operating system, but also some set of applications or processes. For the united description, "partition"will cover both domain and partition in the following statements in this thesis.

It is obvious that there is still other work related to the concepts or similar concepts of partitioned system, such as:

- Integrity-178B (Figure 6) is a RTOS (Real-Time Operating System) manufactured and marketed by Green Hills Software. As designed for real-time the Integrity-178B is POSIX and ARINC 653 compliant, it provides a portable and partitioned environment for safety-critical applications containing multiple programs with different levels of safety criticality on a single processor. The Integrity-178B safety is guaranteed and engineered accordingly with the DO-178B Level A standard [19].

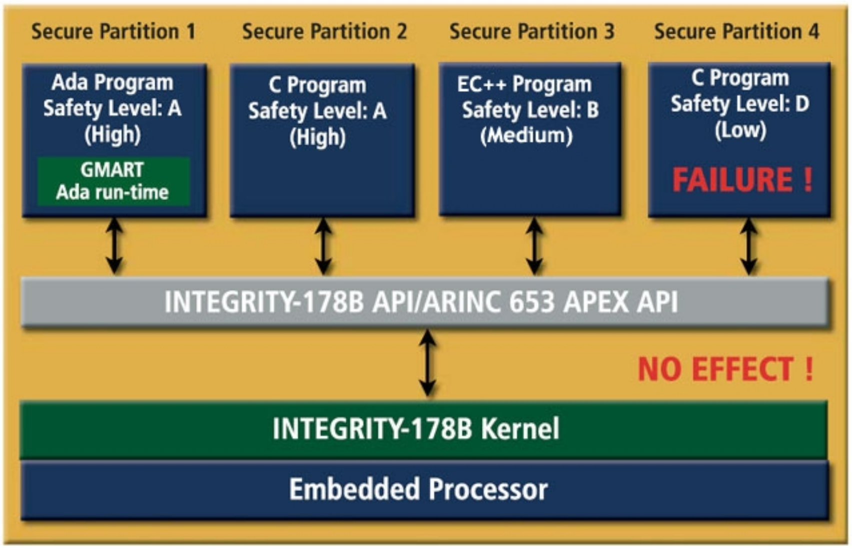

Secure Partition 1
Secure Partition 2
Secure Partition 3
Secure Partition 4
Ada Program Safety Level: A (High)
GMART Ada run-time
C Program Safety Level: A (High)
EC++ Program Safety Level: B (Medium)
C Program Safety Level: D (Low)
FAILURE !
INTEGRITY-178B API/ARINC 653 APEX API
NO EFFECT !
INTEGRITY-178B Kernel
Embedded Processor

Figura 6: Integrity-178B architecture [20]

- VxWorks 653 (Figure 7) is Wind River's robust operating system for controlling complex ARINC 653 IMA systems. It offers complete ARINC 653-1 compliance and DO-178B certification evidence, a range of language options - C, C++ or Ada is available for ARINC 653-1 system development with this product. Applications can be written through VxWorks, ARINC or POSIX APIs. The partition-level operating system varies for different certification levels and adaptation of legacy operating systems, enabling the OS to run existing code with little change. [19].

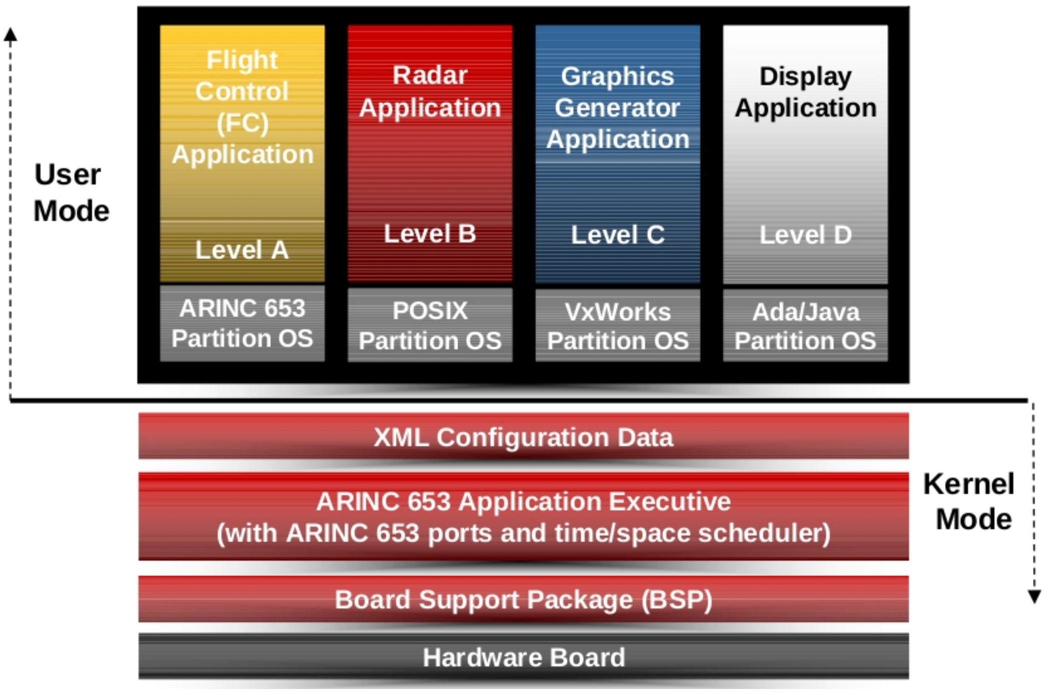


Figura 7: VxWorks 653 IMA architecture [21]

- In [3], the Strongly Partitioned Integrated Real-time Systems (SPIRIT) (Figure 8) is presented. It proposes a design approach for integrated real-time systems in which multiple real-time applications with different criticalities can be feasibly operated, while sharing computing and communication resources.

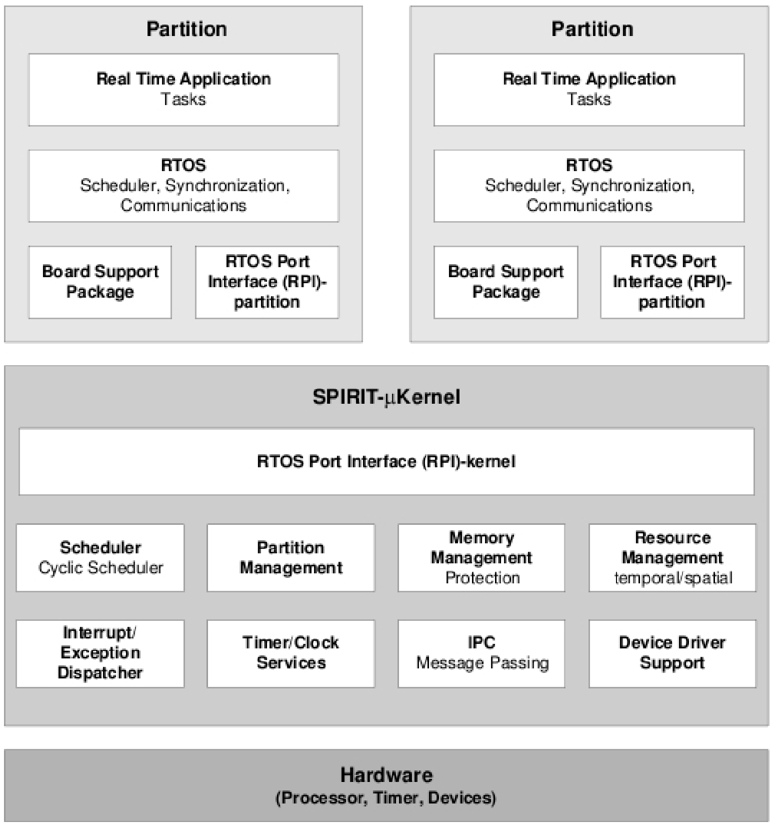


Figura 8: SPIRIT architecture [3]

- [22] presents some initial results which address the increasing gaps between the power of multicore/multiprocessor systems and the lack of development models and tools. One specific goal of this research is to support service contracts on multicore/multiprocessor platforms that support resource partitioning and therefore allow a compositional development of parallel real-time systems. The main focus is the schedulability analysis in partitioned system. Linux offers a low cost, rapid-prototyping capability for IMA applications, but moving these prototypes to a deployable ARINC 653 partitioned system presents many challenges to a developer.
- [23] discusses the steps needed to migrate an OpenGL-based graphic application from a Linux prototype to a deployable ARINC 653 partitioned system platform. It makes use of Wind River's VxWorks 653 Operating Systems with Seaweed Software's Certifiable OpenGL product.

All of above existing work shows that partitioned system has been widely deployed, especially in the areas requiring safety/security-critical to prevent mutual influence between different critical applications. The work presented in this thesis is based on XtratuM, which also focuses on this point, with paying more attention to the areas of embedded applications and providing an entire open-source solution in smaller size, lighter weight and more convenient usage.

2. Overview of XtratuM

2.1. Virtualization and Hypervisor

In computer science, virtualization is an research area existing for some decades years and now a popular topic focused by various hardware/software companies, research institutions, developers and users. It is also a terminology closely related and similar to partitioned system.

Generally speaking, virtualization is a term used to provide abstraction of computer resources. It presents opportunities to reduce hardware costs and power consumption while enabling new platform-level capabilities. It allows a device to run multiple operating environments, and share the underlying processing cores, memory, and other hardware resources [24]. Virtualization can be categorized as hardware virtualization, which enables instruction-level support for virtualization, and software virtualization, which implements virtualization at the operating system or even application level. From the point of view of software virtualization, a hypervisor, also called VMM (Virtual Machine Monitor), is a software layer providing virtualization. The term hypervisor apparently originated in IBM CP-370 reimplementation of CP-67 for the System/370, released in 1972 as VM/370. Combined with hardware mechanisms or capabilities, a hypervisor now allows multiple operating systems to run on a single host computer simultaneously with spatial and temporal isolated. With the development of virtualization technology, a variety of hypervisors have emerged. And based on the mechanisms of implementation, hypervisors can be classified into [25][26]:

- **Unhosted Hypervisor.** This type of hypervisor runs directly on a given hardware platform (as an operating system control program). A guest operating system thus runs at the second level above the hardware. This type of hypervisor includes the classic IBM CP/CMS developed in the 1960s, IBM POWER Hypervisor (PR/SM), IBM System z Hypervisor (PR/SM), Microsoft Hyper-V, VMware ESX Server, Xen, L4 microkernels including OKL4 from Open Kernel Labs, Green Hills Software INTEGRITY Padded Cell, Sun Logical Domains Hypervisor, Real-Time Systems RTS-Hypervisor, LynxSecure from LynuxWorks, ScaleMP vSMP Foundation, Wind River hypervisor, VxWorks MILS Platform, Oracle VM Server, Parallels Server, and VirtualLogix VLX, etc. And KVM, which turns a complete Linux kernel into a hypervisor, also belongs to this type.
- **Hosted Hypersisor.** This type of hypervisor runs within an operating system environment. A "guest.ºperating system thus runs at the third level above the hardware, such as VMware Server, VMware Workstation, VMware Fusion, QEMU,

TenAsys eVM, Sun VirtualBox, Microsoft Virtual PC, Microsoft Virtual Server, and Virtual PC. etc.

As stated above, a key benefit and feature of virtualization is the ability to run multiple operating systems on a single physical system and share the underlying hardware resources - known as partitioning. The concept of virtualization originates from the server consolidation world where the ultimate goals are resource utilization, not time determinism and safety. Hence, it is common that the implementations have been focused on these goals and they cannot simply be reused for safety-critical systems in their current form. To adapt virtualization to the needs of safety-critical systems, the introduction and combination of partitioned system are the good matches [15].

2.2. XtratuM

In this thesis, the work presented is based on XtratuM, which is a hypervisor specially designed for real-time embedded systems. The name XtratuM derives from the Latin word stratum. In geology and related fields, it means "layer of rock or soil with internally consistent characteristics that distinguishes it from contiguous layers". In computer science, this term is used to designate a form of software hiding the programming complexity of the lower levels and supplying functionality for the upper levels. And in order to stress the tight relationship with Linux and the open-source movements, the "S"was replaced by "X". XtratuM would be the first layer of software (the one closest to the hardware), which provides a solid basis for the rest of the system. From this point of view, XtratuM can be defined as a layer which is directly inserted between the hardware and others OSes, easing the programming of these OSes as well as allowing to execute all of them in an isolated and concurrent way [27][28][29].

Many of today's embedded applications require deterministic real-time performance and visualization. And deployment of multiple operating systems on the same hardware is a logical step in embedded systems design, thus reducing total hardware costs while increasing reliability and system performance. Although hypervisor has been widely used in mainframe systems since the 1960s, it has not been used in the embedded devices until recent years [30]. Most of the recent advances of virtualization and hypervisors have been done in the desktop systems, and it is not as direct as it may seem to implement them for embedded systems. There are hypervisors for embedded systems, e.g. TRANGO, LynxSecure and OKL4, which are either commercial software or own commercial properties [29][31][32][33]. As a pure open source alternative, XtratuM is a hypervisor specially designed for real-time embedded systems. It is important to note that a hypervisor is an "enabling" technology, rather than a technology to solve problems. The hypervisor provides a framework to run several operating systems (or real-time executives) in a robust partitioned environment. XtratuM can be used to build a MILS architecture [28]. Meanwhile, for low-end CPUs without hardware virtualization support, such a software hypervisor is a helpful supplement for promoting hardware utilization. XtratuM aims to make use of para-virtualization. The para-virtualized operations are as close to the hardware as possible. Therefore, porting an operating system that has already worked on the native system is a simple task: replace some parts of the operating system HAL with the corresponding hypercalls. Here the term hypercall, or hypervisor call, referred to the para-virtualization interface, by which a guest operating system could access services directly from the higher-level control program - analogous to making a supervisor call to the same level operating system. The term supervisor refers to the operating system kernel, which runs in supervisor or privileged state. The para-virtualization technique consists in replacing the conflicting instructions explicitly by functions provided by the hypervisor. In this case, the partition code has to be aware of the limitations of the virtual environment and use the hypervisor services. Those services are provided thought a set of hypercalls. The hypervisor is still in charge of managing the hardware resources of the systems, and enforce the spatial and temporal isolation of the guests. Direct access to the native hardware is not allowed. The para-virtualization is the technique that better fits the requirements of embedded systems: faster, simpler, and smaller, and to para-virtualize the guest operating systems is not a problem once they are open source. Also, this technique does not require special processor features that may increase the cost of the product.

XtratuM was originally developed by Real-Time System Research Group of Universidad Politecnica de Valencia for x86 and ARM platforms. The preliminary implementation of XtratuM used the demonstration of ADEOS for reference. For the version 1.0 and before, XtratuM was designed/implemented avoiding starting from the scratch but from a previous existing kernel, which was Linux here as the root partition. This approach helped to avoid the management of the virtual memory or the great majority of the existing devices. It just took care of interrupts and timers of the system and supplied a scheduler to schedule the partitions. These first versions of XtratuM were initially designed as a substitution of the RTLinux HAL (Hardware Abstraction Layer) to meet temporal and spatial partitioning requirements and improve the initial RTLinux virtualization mechanism. The goal was to virtualize the essential hardware devices to execute several OSes concurrently, with at least one of these OSes should provide real-time capabilities. The other hardware devices (including booting) were left to the root partition, e.g. Linux in the implementations [28]. These first versions of XtratuM consist of [12]:

- A patch for the Linux kernel. This patch modifies the Linux kernel in two ways: replacing all the disabling/enabling interrupt instructions by calls to XtratuM and inserting several hooks in the Linux kernel code. These hooks will be used later to virtualize the interrupts and the hardware timers.
- The XtratuM nanokernel itself. This nanokernel is provided as a piece of software which has to be inserted inside Linux through the Linux kernel module mechanism. XtratuM is loaded into the system as a Linux kernel module (sharing the memory map with Linux). For partitions, XtratuM uses its own loader to create a specific memory map for each of them, enabling memory protection between the different ones.

These first versions were implemented in a similar way of hosted hypervisor because they ran within a Linux environment. XtratuM and the root partition still shared the same memory space. Nevertheless, the rest of the partitions were running in non-privileged mode and in different memory maps, providing partial space isolation. These first versions of XtratuM were initially developed to meet the requirements of a hard real-time system. The main goal was to guarantee the temporal constrains for the real-time partitions. And the core functionality was to abstract the essential devices to run a kernel and provide real-time capabilities: the timers, the interrupts and the memory. There are other characteristics including [27]:

- The first partition shall be a modified version of Linux
- Partition code has to be loaded dynamically
- There is not a strong memory isolation between partitions
- Linux is executed in processor supervisor mode
- Linux is responsible of booting the computer
- Fixed priority partition scheduling

The benefits of using XtratuM could be:

- Operating system developers do not have to deal with the complexities of timer and interrupt of hardware.
- Porting of a guest RTOS to a processor, supported by XtratuM, is almost immediate. As para-virtualization requires, the hardest part of a porting is to enable the guest OS compliant with XtratuM API. XtratuM will give a bootable platform ready to run code and more.
- A version of XtratuM supports multiple RTOS by sharing the hardware among them. Interrupts are delivered to the RTOS in daisy-chain fashion.
- These first versions of XtratuM have been designed to take advantage of running Linux jointly with partitions of RTOS, that is, it is possible to execute concurrently the guest OS and Linux (Figure 9). In this case, Linux is executed in background. This mechanism provides a convenient framework to develop new partitions, which would be loaded similar to Linux kernel module and exceptions are intercepted and reported by XtratuM. All first versions of XtratuM have the same API, so once to port or develop new partitions to XtratuM, all these benefits can be achieved by just recompiling the code.

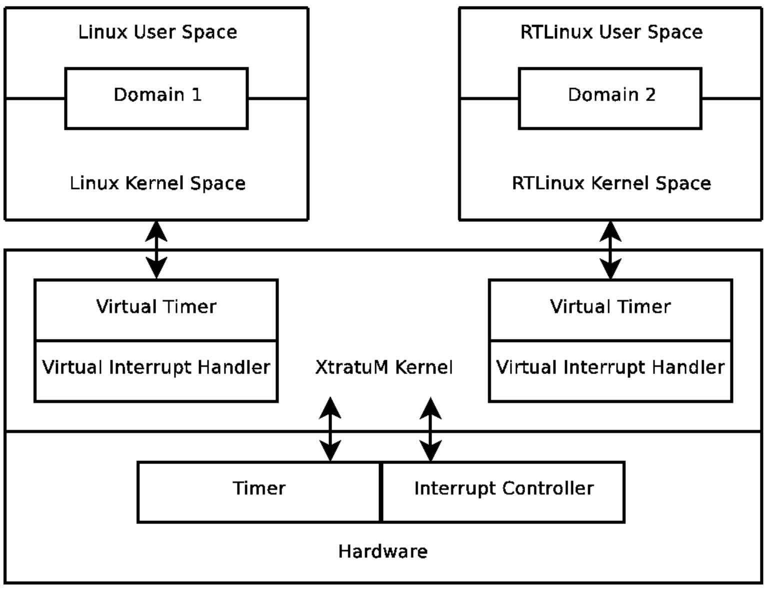


Figura 9: Preliminary XtratuM architecture [34]

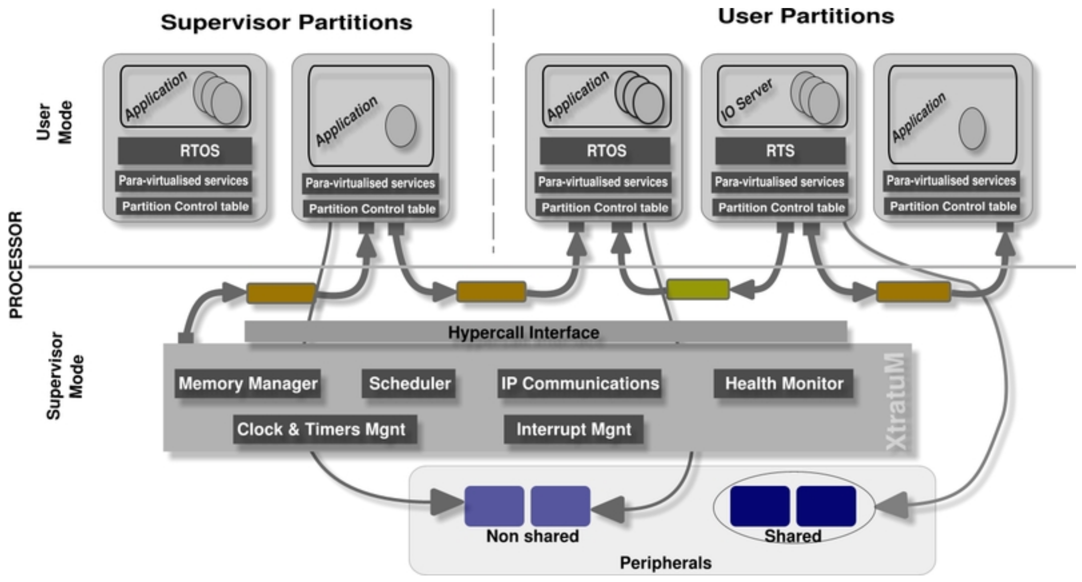


Figura 10: Current XtratuM architecture [29]

After the experience of first versions, XtratuM was redesigned to be independent of Linux and bootable. The result of this is XtratuM 2.x, an unhosted hypervisor which can be smartly configured (Figure 10). These new versions are being used to build a TSP-based (Time and Space Partitioning) solution for payload on-board software, highly generic and reusable, project named LVCUGEN. TSP based architecture has been identified as the best solution to ease and secure reuse, enabling a strong decoupling of the generic features to be developed, validated and maintained in mission specific data processing.

XtratuM 2.x hypervisor supports the x86 and LEON2 (SPARC v8) architectures, and operating systems including Linux as GPOS, and PartiKle and RTEMS as RTOS. The most relevant features of XtratuM 2.x can be concluded as [35]:

- Bare metal hypervisor
- Employs para-virtualisation techniques
- An hypervisor designed for embedded systems: some devices can be directly managed by a designated partition
- Strong temporal isolation: fixed cyclic scheduler
- Strong spatial isolation: all partitions are executed in processor user mode, and do not share memory
- Resource allocation via a configuration table
- System definition (partitions and allocated resources) defined via an XML configuration file
- Health monitoring support
- Supervisor and standard partitions
- Robust inter-partition communication mechanisms (ARINC 653 sampling and queuing ports)

In the case of embedded systems, particularly avionics systems, ARINC 653 standard defines a partitioning scheme. Although this standard was not designed to describe how a hypervisor must operate, some parts of the model are quite close to the functionality provided by a hypervisor. The XtratuM API and internal operations resemble the ARINC 653 standard, but XtratuM is not an ARINC 653 compliant system. ARINC 653 relies on the idea of a separation kernel defining the API, the operations of the partitions, and also how the threads or processes are managed inside each partition. It provides a complete definition of the system. However, XtratuM does not cover the details inside of partitions. For this reason, ARINC 653 is used as a reference in the design of XtratuM, but not the intention to implement XtratuM in an ARINC 653 compliant system [27].

3. Overview of PowerPC

3.1. POWER

POWER (Performance Optimization With Enhanced RISC) Architecture is a second generation RISC (Reduced Instruction Set Computer) architecture. IBM began delivery of RS/6000 products with POWER processors in February of 1990 [36].

POWER Architecture incorporated characteristics common to most other RISC architectures, such as [36]:

- Instructions were a 4-byte fixed length with consistent formats, permitting a simple instruction decoding mechanism.
- Load and store instructions provided all of the accesses to memory.
- The architecture provided a set of GPRs (General Purpose Registers) for fixed-point computation, including the computation of memory addresses.
- It provided a separate set of FPRs (Floating-point Registers) for floating-point computation.
- All computations retrieved source operands from one register set and placed results in the same register set.
- Most instructions performed one simple operation.

POWER Architecture was unique among the existing RISC architectures, as it was functionally partitioned, separating the functions of program flow control, fixed-point computation, and floating-point computation. The architecture's partitioning facilitated the implementation of superscalar designs, in which multiple functional units concurrently executed independent instructions [36].

3.2. PowerPC

Early in 1991, AIM (Apple-IBM-Motorola alliance) worked together to develop an architecture that would meet the needs of the alliance. Because it would have been impossible to develop a completely new architecture in time to satisfy the needs of their customers, the companies decided to use POWER Architecture as the starting point. They made changes to achieve a number of specific goals, including [36]:

- Permit a broad range of implementations, from low-cost controllers to high-performance processors
- Be sufficiently simple so as to permit the design of processors that have a very short cycle time
- Minimize effects that hinder the design of aggressive superscalar implementations
- Include multiprocessor features
- Define a 64-bit architecture that is a superset of the 32-bit architecture, providing application binary compatibility for 32-bit applications

Finally, the alliance reached a consensus on the definition of the PowerPC (Power Performance Computing, sometimes abbreviated as PPC) Architecture. As an evolving instruction set, PowerPC has since 2006 been renamed Power ISA (Instruction Set Architecture) but lives on as a legacy trademark for some implementations of Power Architecture based processors. This architecture achieves the goals previously listed, yet permits POWER customers to run their existing applications on new systems and to run new applications on their existing systems [36][37].

PowerPC retains a high level of compatibility with POWER; the architectures have remained close enough that the same programs and operating systems will run on both if some care is taken in preparation; newer chips in the POWER series implement the full PowerPC instruction set. Most POWER applications will benefit from the improved performance of new PowerPC processors. Starting with the basic POWER specification, the PowerPC added features as [36][37]:

- Support for operation in both big-endian and little-endian modes, except PowerPC 970, and PowerPC can switch from one mode to the other at run-time
- Single-precision forms of some floating point instructions, in addition to double-precision forms
- Additional floating point instructions at the behest of Apple
- A complete 64-bit specification that is backward compatible with the 32-bit mode
- A fused multiply-add
- A paged memory management architecture which is used extensively in server and PC systems
- Addition of a new memory management architecture called Book-E, replacing the conventional paged memory management architecture for embedded applications. Book-E is application software compatible with existing PowerPC implementations, but requires minor changes to the operating system

Some instructions present in the POWER instruction set were deemed too complex and were removed in the PowerPC architecture. Some of the removed instructions could be emulated by the operating system if necessary. The removed instructions are [37]:

- Conditional moves
- Load and store instructions for the quad-precision floating-point data type
- String instructions

PowerPC is designed along RISC principles, and allows for a superscalar implementation. Versions of the architecture design exist in both 32-bit and 64-bit implementations. It extends addressing and fixed-point computation to 64 bits, and supports dynamic switching between the 64-bit mode and the 32-bit mode. In 32-bit mode, a 64-bit PowerPC processor will execute application binaries compiled for the 32-bit subset architecture [36][37].

Originally intended for personal computers, PowerPC processors have since become popular embedded and high-performance processors. PowerPC was the cornerstone of AIM PReP and Common Hardware Reference Platform initiatives in the 1990s and while the architecture is well known for being used by Apple Macintosh lines from 1994 to 2006 before Apple's transition to Intel. Nowadays, its use in video game consoles, embedded applications and high-performance applications has far exceeded Apple's use [37].

3.3. PowerPC in Safety-critical

Operating systems that work on PowerPC are generally divided into those which are oriented towards the general-purpose PowerPC systems, and the embedded PowerPC systems. Those GPOSes cover Apple Macintosh, IBM AIX, IBM i5/OS, NetBSD, FreeBSD, OpenBSD and several Linux distributions, e.g. Debian, Fedora, Gentoo, and Yellow Dog, etc. For embedded systems, there are MontaVista, QNX, CISCO IOS, LynxOS, VxWorks, RTEMS, INTEGRITY and PikeOS etc [37]. As known from former sections, most of those embedded systems are compliant with safety-critical standards or applied into safety-critical areas. There are several cases that PowerPC is utilized in embedded applications for safety-critical goals, such as:

- LynxOS-178 RTOS is the first and only hard real-time DO-178B level A operating system to offer the interoperability benefits of POSIX with support for ARINC 653 APEX, and is also the first and only time- and space-partitioned, FAA-accepted (Federal Aviation Administration) RSC (Reusable Software Component). It is the only COTS solution supporting both Intel Pentium and PowerPC platforms [38].
- The Verisoft XT Avionics project, funded by BMBF (German Ministry of Education and Research), is to formally verify an existing operating system which has not been designed for deductive code verification. As functional correctness of the built-in operating system is a crucial requirement for the reliability of safety- and security-critical systems, one section of this project is to prove functional correctness of the microkernel in PikeOS. The verification target chosen for this part is the PikeOS version for the PowerPC processor family and the Freescale MPC5200 platform. Based on the model of the underlying PowerPC hardware, it is able to verify low-level functions of the PikeOS kernel. So far, a model of the PowerPC architecture and assembly language has been generated, with treatment of both entire assembly files and the relatively frequent inline assembly. Milestones that have been reached include the verification of the functional correctness of helper functions and the verification of first system calls of PikeOS kernel [39][40].
- Wind River VxWorks 653 Platform delivers the stringent IMA foundation A&D (Aerospace and Defense) companies need to address the safety and security requirements of mission-critical applications, as well as the portability and reusability requirements of noncritical applications. The platform delivers complete ARINC 653 Part 1, Supplement 2 conformance and expands the ARINC 653 XML configuration capabilities to enable a true multivendor platform that supports fully independent RTCA DO-297/EUROCAE ED-124 IMA supplier sourcing. The platform supports PowerPC, including 7xx/74xx/82xx/83xx/85xx/86xx [41].
- Lockheed Martin Aeronautics will use GNAT Pro, a robust and flexible Ada development environment, to develop Flight Management System Interface Manager and Radio Control software on the C-130J Super Hercules aircraft. The product is GNAT Pro High-Integrity Edition for PowerPC that runs VxWorks 653 [42].

- In the safety-critical area as aerospace, the Orion Project of NASA is going to be built with some of the most state of the art communications and control features that are out. These include a COTS PowerPC 750 FX Processor running the Green Hills INTEGRITY-178B real-time operating system [43].

According to above cases, in the applications and markets of embedded systems, real-time systems and safety-critical, PowerPC has been deployed a lot and playing an important role in hardware architectures for choosing. Therefore, in this thesis, considering the potential applied platforms of XtratuM, PowerPC is selected as the target for implementation.

4. Main PowerPC Drivers to Virtualize

The main work in thesis is summarized to port XtratuM 1.0 to PowerPC architecture. In what follows, the name XtratuM will be used to refer to the version 1.0 of XtratuM. To specify PowerPC as the target platform, it has been considered in the following aspects.

4.1. Processors

From the point of view of application, as a hypervisor aims at embedded applications, it will be beneficial for XtratuM to be implemented in different architectures of embedded systems. In contrast to the ubiquity of the x86 architecture in the PC world, the embedded world contains a wider variety of architectures. Support for virtualization requires memory protection and a distinction between user mode and privileged mode. As one of the most important and mainstream architectures, PowerPC is and will go on, playing a major role in IT (Information Technology). From former Apple Macintosh to current charming Wii/Xbox360/PlayStation3 and amazingly powerful Deep Blue and Blue Gene, PowerPC is a common feature of the vastly different spectrum of applications. Not like only two giants, Intel and AMD, dominating the x86 world, as the architecture is open for licensing by third parties, there are several manufacturers contributing to the PowerPC realm, including Motorola/Freescale, Xilinx, AMCC, CISCO etc. Those licensees can choose to license anything from a single predefined core, to a complete new family of Power Architecture products [44]. The open licensing mechanism makes PowerPC cover a wide range of family members from low-end 32-bit like 4xx series up to high-end multi-threaded 64-bit like CELL, which can satisfy different powerful and stable performance requirements of various types of embedded and real-time operating systems.

In embedded world, 32-bit and 64-bit PowerPC processors have been a favorite of embedded computer designers. To keep costs low on high-volume competitive products, the CPU core is usually bundled into a SOC (System-on-Chip) integrated circuit. SOCs contain the processor core, cache and the processor's local data on-chip, along with clocking, timers, memory (SDRAM), peripheral (network, serial I/O), and bus (PCI, PCI-X, ROM/Flash bus, I2C) controllers. IBM also offers an open bus architecture (called CoreConnect) to facilitate connection of the processor core to memory and peripherals in a SOC design. And a recent development is the BookE PowerPC Specification, implemented by both IBM and Freescale, which defines embedded extensions to the PowerPC programming model [45]. Hereby, the fundament of various PowerPC products keeps unified, which provides convenience for developers. And once XtratuM were implemented in one of the PowerPC processors, it could be not difficult to implement in others thanks to the similarity of hardware specifications. Some other kinds of hypervisors, e.g. Xen, QEMU, KVM, also cover some types of processors of PowerPC. So porting XtratuM to PowerPC will be a helpful expansion of the applicable area, and also promote embedded and real-time applicable abilities and safety-critical applications based on PowerPC. Also, the specifications of Power ISA keep evolving and virtualization and hypervisor functionalities have been included since v.2.04 finalized in 2007, and enhanced in v2.06 released in 2009. So XtratuM can be combined and updated with the hardware-level virtualization of PowerPC to achieve higher performance in the future after there is a prototype in PowerPC [44].

The work in this thesis was mainly developed in AMCC PowerPC 440EP (Yosemite), and also with AMCC PowerPC 440GR (Yellowstone) and DAVE PowerPC 405EP (PP-ChameleonEVB). Chosen as the main target platform in this work, 440EP is welcome in current embedded development and applications. For example, it has been selected by PIKA Technologies Inc. for its WARP Appliance, an embedded hardware platform that supports IP, voice and fax solutions [46]. Furthermore, for safety-critical, it is supported by LynxOS-178 Safety-critical RTOS [38]. So this processor could be a mainstream choice to make XtratuM enter the PowerPC realm. As mentioned in former sections, the timers, the interrupts and the memory are the essential parts to be processed in the implementation.

4.2. Timer

In the design and implementation of real-time applications, the timer is indispensable for setting or measuring the time sensitive items, such as budget, period, deadline and executing time etc. Hereby, there should be timers to obtain the absolute time such as a time stamp and to set the relative time such as a period of time offset.

In the x86 implementation, XtratuM refers to two types of timer: TSC (Time Stamp Counter) and PIT (Programmable Interval Timer). TSC is a 64-bit register present on all x86 processors since the Pentium. It counts the number of ticks since reset. When partitions are being scheduled to execute, XtratuM will read value of TSC to obtain the current time stamp. The Intel 8253/8254 PITs perform timing and counting functions. PIT is used as a 16-bit down counterinterrupting on terminal count. That is, a value is written to a register and PIT decrements this value until it gets to 0. At that moment, it activates an interrupt to the IRQ 0 input on the 8259 interrupt controller. XtratuM will set PIT with customized value to decrement from the time stamp returned by TSC [47][48].

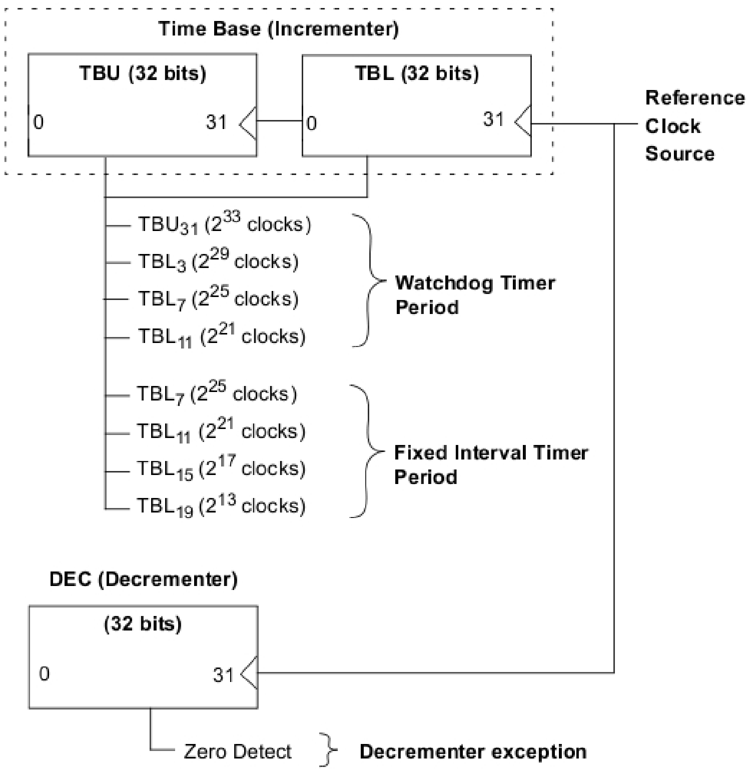


Figura 11: Timer facilities of PowerPC [49]

PowerPC 440EP provides four timer facilities (Figure 11): TB (Time Base), DEC (Decrementer), FIT (Fixed Interval Timer), and WT (Watchdog Timer). These facilities, which share the same source clock frequency, can support [49]:

- Time-of-day functions
- General software timing functions
- Peripherals requiring periodic service
- General system maintenance
- System error recovery

Considering the functionalities required by XtratuM, TB and DEC will be referred to. TB is a 64-bit register which increments once during each period of the source clock, and provides a time reference. Access to TB is via two SPRs (Special Purpose Register): TBU (Time Base Upper) SPR contains the high-order 32 bits of TB, while TBL (Time Base Lower) SPR contains the low-order 32 bits. The period of the 64-bit

TB is approximately 1462 years for a 400 MHz clock source. The TB value itself does not generate any exceptions, even when it wraps. For most applications, TB is set once at system reset and only read thereafter [49].

DEC is a 32-bit privileged SPR that decrements at the same rate as TB increments, or in other word, at the same frequency as the CPU. When a non-zero value is written to the DEC, it begins to decrement with the next TB clock. A Decrementer exception is signaled when a decrement occurs on a DEC count of 1 [49].

Consequently, the functions to retrieve different time for real-time capabilities can be satisfied by these timer facilities of PowerPC. One issue should be noted that for PowerPC, thanks to the design and implementation of SOC, DEC is internal to the processor. This contrasts the x86 architecture where PIT is a peripheral chip and generates an external interrupt coming in from the interrupt controller.

4.3. Interrupt

Taking over interrupts on a computer is a synonymous of controlling the whole machine. Once XtratuM is started up, it is the only one who really controls hardware interrupts and, of course, the only one who is able to disable/enable real interrupts [12]. Interrupts and exception handling could be varied a lot in different processors, so XtratuM implementation strongly depends on the low level stuff. IDT (Interrupt Descriptor Table) is a data structure and software interface used by the x86 architecture to implement an interrupt vector table. It consists of 256 interrupt vectors and is the place where the addresses of interrupt handlers are stored. It is used by processor to determine the correct response to interrupts and exceptions. Use of the IDT is triggered by three types of events: hardware interrupts, software interrupts, and processor exceptions, which together are referred to as "interrupts"[50]. To handle an interrupt by way of an IRQ (Interrupt Request), kernel will access a jump table in entry.S, find out the corresponding call gate (descriptor), and finally the handler code [51] (Figure 12). In the x86 implementation, XtratuM replaces entries of IDT with XtratuM's own interrupt handler addresses. Once IDT has been modified in this way, interrupts are completely managed by XtratuM.

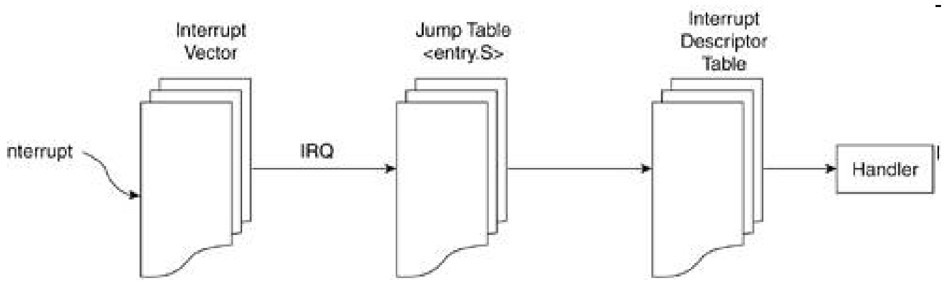


Figura 12: Interrupt flow of x86 [51]

In the terminology of PowerPC, an interrupt is the action in which the processor saves its old context, including MSR (Machine State Register) and next instruction address, and begins execution at a pre-determined interrupt handler address, with a modified MSR. Exceptions refer to the events that may cause the processor to take an interrupt, if the corresponding interrupt type is enabled. Exceptions may be generated by the execution of instructions, or by signals from devices external to the processor, the internal timer facilities, debug events, or error conditions, etc [49].

Note that in x86, the external interrupts come in processor from the interrupt controller, usually the family member of 8259 PICs (Programmable Interrupt Controllers), which acts as a multiplexer, combining multiple interrupt input sources into a single interrupt output to interrupt a single device [52]. XtratuM takes over the driver for this device in order to handle the external interrupts. In PowerPC, there are UIC (Universal Interrupt Controllers) providing all necessary control, status, and communication between the various interrupt sources and the processor core [49]. So it is necessary for XtratuM to manipulate UIC in a proper way.

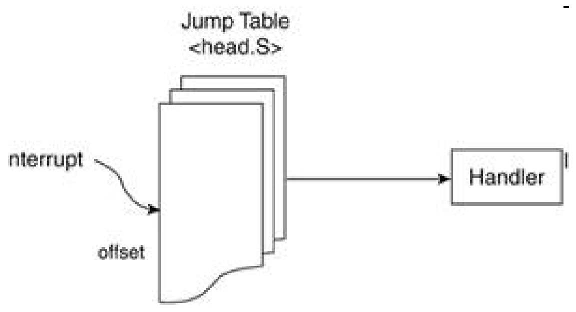


Figura 13: Interrupt flow of PPC [51]

In PowerPC, there is no data structure as IDT. IVORs (Interrupt Vector Offset Registers) are used to store the specific offsets of interrupt vectors in memory where the code to jump to the appropriate handlers [49]. The respective implementation of interrupt handlers is indexed by way of being fixed in memory in corresponding head.S of different cores, which also can be treated as a jump table [51] (Figure 13). Kernel will find out the appropriate handler according to the offset. To enable XtratuM to completely control the interrupts in PowerPC, those fixed offsets related with timer, memory, hypercall, and also that responding to external interrupts from hardware, must point to XtratuM's own interrupt handlers instead of the original Linux' ones.

4.4. Memory

Like interrupt/exception, memory management is also very hardware specific, even for the same architecture in different cores. In the x86 implementation, XtratuM is able to create a memory map per partition with memory isolation among different partitions.

XtratuM will manually update the corresponding entry in any process' PGD (Page Global Directory), as well as in the cached PGDs, to keep all of them in a consistent state. The reason for doing this manually in XtratuM is to reduce the number of page-faults, and also because some page faults could potentially occur in RT-context if this is not done, though it would not be needed for regular user-space tasks. Meanwhile, XtratuM also guarantees to disable real interrupts in order to avoid potentially harmful ones, when a trap handler executed for a triggered page fault [53]. At present, some features for inter-partition communication also have been implemented, such as XM/FIFO and XMSHM [54][55].

Though in the x86 implementation, XtratuM does not provide many features for memory management, in consideration of the PowerPC family owning various processors from 32-bit to 64-bit, it is better to try to cover as much as possible specific issues of memory management. An typical example is that PowerPC 405 could be transferred into real-mode to access physical addresses, such as virtual address translation will be disabled when interrupts being triggered, while 440 follows Book E specifications and has the address translation always on without physical address access. So the mechanism to handle memory management for PowerPC must be carefully treated and considered more in XtratuM.

5. Porting Implementation

Embedded system designers and users are becoming intensely interested in hypervisor for a variety of reasons, including security, reliability, licensing, legacy software support (especially related to multicore processors), and flexibility in resource provisioning. The most commercially prominent examples of embedded virtualization today are the Xbox360 and PlayStation3 game consoles, both of which ship with a hypervisor [56]. The original design of XtratuM aims to provide multiple partitions support, so there must be proper memory management to guarantee spatial isolation for each domain. Considering the real-time capabilities certain partitions need to offer, each partition must be in temporal isolation to enable proper timers for real-time services of its own. Meanwhile, virtualization prevents partitions accessing real hardware directly, so there must be mechanisms to handle interrupt requests for partitions, which also must keep real-time performance for related partitions. Although partitions are isolated, they must be synchronized at a macro level for the information of timer, interrupt and memory allocation to coordinate runtime status in the same hardware. Therefore, as stated in former sections, the implementation has to process three key functions: timer, interrupt and memory management. Furthermore, as these core functions are fundamental for partitions, to enable partitions to communicate with the hypervisor, basic hypercalls must be provided by XtratuM as the para-virtualization interface.

5.1. Hypercall

So far XtratuM implements the following fundamental hypercalls [53]:

- **int load_domain(char *dname, int prio, domain image t *img).** This function can be called only from the root partition's context. It is responsible for creating the partition's virtual memory map, loading it in this memory map, inserting it into the event daisy chain and starting execution. It function returns the partition identifier or -1 if an error occurs.
- **int unload_domain(int id).** This function can be called only from the root partition's context. Once executed, it unloads the partition identified by *id*. But the root partition can not be unloaded by it. It returns 0 on success or -1 if an error occurs.
- **void exit_domain(int value).** This function marks the caller partition as *terminated*. However it does not unload the partition. Every partition but the root partition can be unloaded only via the **unload_domain** system call. The *value* parameter allows the domain to report its finished status.
- **void suspend_domain(void).** This function marks the caller partition as *suspended*, yielding the processor to the next more priority partition.
- **void sync_events(void).** XtratuM does not implement a hypercall to enable/disable events but it maps a shared area of memory where partitions must write down the current status of events (only when events are enabled). Some of them are pending then partitions should call sync_events in order to execute the suitable event handlers.
- **void pass_event(int event).** This function permits to pass an event to the next partition in the event daisy chain.
- **int get_time (struct xmtimespec *t).** This function reads from the monotonic clock system, returning the current time in nanoseconds.
- **void set_timer (struct xmitimerval *t, struct xmitimerval *r).** This function programs the system timer to trigger a timer interrupt in t units of time. If the timer is programmed, then r will return the remaining time until the end of the current count.
- **unsigned long get_cpu_khz(void).** This function returns the frequency of the real CPU in Khz.
- **int write_scr(char *buffer, int len).** This function prints *len* characters of the string *buffer* on the screen through the root partition. It returns the number of printed characters.

Hypercalls are implemented in hypervisor level and not hardware sensitive, so in PowerPC, they are almost kept the same as in x86.

5.2. Timer

In real-time applications, scheduling is an indispensable issue that is closely related with timer. In any type of scheduling, the task must start from one moment, go on executing for a period of time and then switch to the next task. So the basic requirement for timer in XtratuM is to guarantee the start point and the available duration for the task. The start point should be a time stamp from which the duration will be measured. XtratuM makes use of two hypercalls (*get_time* and *set_timer*) to interact with the virtual timer services. To obtain a time stamp, in PowerPC, XtratuM must access TB to read out the latest value there. As mentioned before, the *get_time* hypercall is implemented for this function (Figure 14).

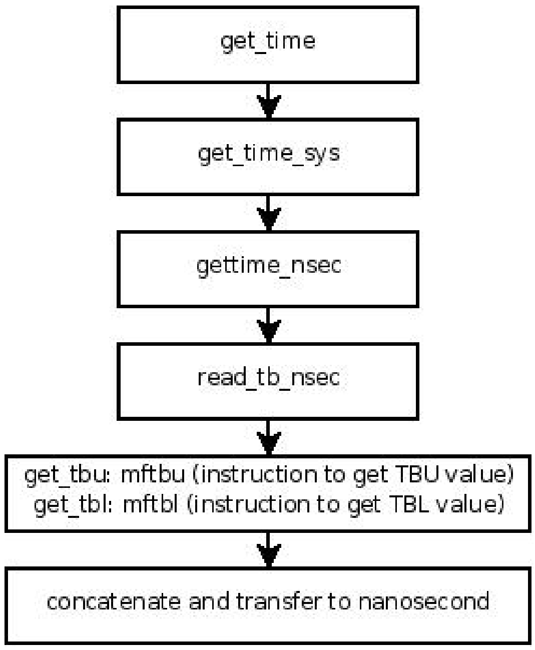

get_time
get_time_sys
gettime_nsec
read_tb_nsec
get_tbu: mftbu (instruction to get TBU value)
get_tbl: mftbl (instruction to get TBL value)
concatenate and transfer to nanosecond

Figura 14: Obtain the current value of TB as nanosecond

In the x86 implementation of XtratuM, to set the duration for a task, is performed as to assign the period for a partition. This period is one property specified by each partition self. Time coordination of different partitions is sorted by virtual timers as a partition heap. The virtual timer is created for each partition by XtratuM based on the hardware timer achieved by timer handler from kernel. For each partition, XtratuM provides at least one virtual timer, and the exact number of virtual timers depends on the number of available hardware timers. The partitions should interact with these virtual facilities behaving like the regular ones but invoking XtratuM primitives instead of interacting with real hardware. The virtual timer implements the multiplexing of the hardware timer between the existing partitions. Such a virtual timer in XtratuM will make use of DEC in PowerPC via the *set_timer* hypercall to count down the period. Hereby, when one partition is scheduled to execute, XtratuM must pass the specific period to DEC as the value to decrease. Unlike the series of complex operations on 8253/8254 chips in x86, as internal to processor, it is easy to set DEC with a privileged instruction *"mtdec"* (Figure 15).

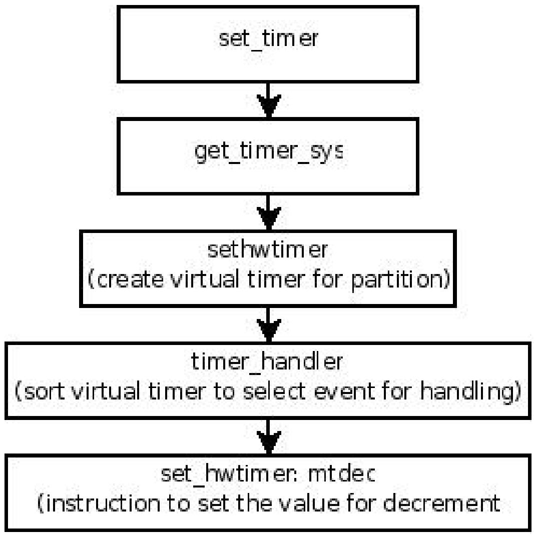
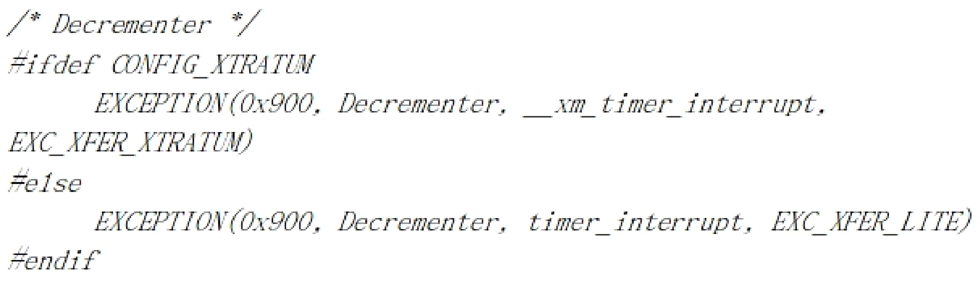

set_timer
get_timer_sys
sethwtimer
(create virtual timer for partition)
timer_handler
(sort virtual timer to select event for handling)
set_hwtimer: mtdec
(instruction to set the value for decrement

Figura 15: Set countdown value for DEC

5.3. Interrupt

XtratuM must be enabled to manage all hardware interrupts available in PowerPC. Once XtratuM interrupt virtualization is activated, all interrupts of any partition will pass through it, thus none of the partitions ever get direct access to the interrupt hardware. The mechanism is to insert hooks into Linux kernel to enable XtratuM to provide necessary interrupt handlers instead of Linux.

As timer interrupt generated by DEC is internal for PowerPC, not like PIT generating external interrupt in x86, it should be handled explicitly by XtratuM. The offset of interrupt vector for DEC must point to the handler implemented by XtratuM instead of the original Linux one (Figure 16). From this vector, XtratuM will transfer its own timer interrupt handler (*timer_handler*) to respond the interrupt (Figure 17).



```
/* Decrementer */
#ifdef CONFIG_XTRATUM
	EXCEPTION(0x900, Decrementer, __xm_timer_interrupt,
EXC_XFER_XTRATUM)
#else
	EXCEPTION(0x900, Decrementer, timer_interrupt, EXC_XFER_LITE)
#endif
```

Figura 16: Interrupt vector for DEC in XtratuM

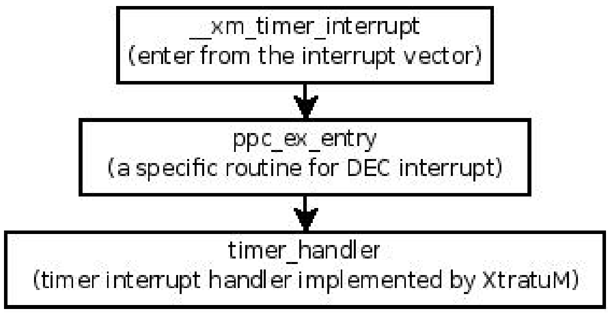

__xm_timer_interrupt
(enter from the interrupt vector)
ppc_ex_entry
(a specific routine for DEC interrupt)
timer_handler
(timer interrupt handler implemented by XtratuM)

Figura 17: Transfer timer interrupt handler from XtratuM

For the interrupt vectors related with memory, including DataStorage and InstructionStorage (Figure 18), the handlers point to the same function (*xm_do_page_fault*) provided by XtratuM responsible for these interrupts caused by paging issues (Figure 19).

```
#define DATA_STORAGE_EXCEPTION
        START_EXCEPTION(DataStorage)
        NORMAL_EXCEPTION_PROLOG;
        mfspr   r5,SPRN_ESR;            /* Grab the ESR and save it */
        stw     r5,_ESR(r11);
        mfspr   r4,SPRN_DEAR;           /* Grab the DEAR */
        EXC_XFER_EE_LITE(0x0300, handle_page_fault)

#define INSTRUCTION_STORAGE_EXCEPTION
        START_EXCEPTION(InstructionStorage)
        NORMAL_EXCEPTION_PROLOG;
        mfspr   r5,SPRN_ESR;            /* Grab the ESR and save it */
        stw     r5,_ESR(r11);
        mr      r4,r12;                 /* Pass SRR0 as arg2 */
        li      r5,0;                   /* Pass zero as arg3 */
        EXC_XFER_EE_LITE(0x0400, handle_page_fault)
```

Figura 18: Interrupt vectors for memory storage in XtratuM

```
handle_page_fault:
        stw   r4,_DAR(r1)
        addi  r3,r1,STACK_FRAME_OVERHEAD
#ifndef       CONFIG_XTRATUM
        bl    do_page_fault
#else
        bl    xm_do_page_fault
#endif
```

Figura 19: Transfer memory storage interrupt handler from XtratuM

For the interrupt vector DoSyscall processing system calls, the original handler is also taken place by the one (*xm_do_syscall*) provided by XtratuM. When partitions invoke the hypercalls of XtratuM, in the hypervisor level, this interrupt is triggered and responded by the handler (Figure 20).

```
#ifndef       CONFIG_XTRATUM
        EXC_XFER_EE_LITE(0x0c00, DoSyscall)
#else
        EXC_XFER_EE_LITE(0x0c00, xm_do_syscall)
#endif
```

Figura 20: Interrupt vectors for system call in XtratuM

XtratuM is also enabled to explicitly intercept external interrupt at the interrupt vector offset for external interrupt (Figure 21). XtratuM enables its own handler (*xm_irq_handler*) (Figure 22) to get the external interrupt number and the pending events, schedule the events compatible with the partitions, and then begin the processing of virtual interrupt for the partitions. XtratuM simply provides a prioritization of hardware interrupts in software. The partitions could generate various types of interrupts, even the same kind as is in use with a low-priority partition. From the point of view of partitions, the interrupts appear as hardware interrupts, though they are processed virtually by XtratuM instead of real hardware.

```
#ifdef CONFIG_XTRATUM
        EXCEPTION(0x0500, ExternalInput, __xm_do_IRQ, EXC_XFER_XTRATUM)
#else
        EXCEPTION(0x0500, ExternalInput, do_IRQ, EXC_XFER_LITE)
#endif
```

Figura 21: Interrupt vectors for external input in XtratuM

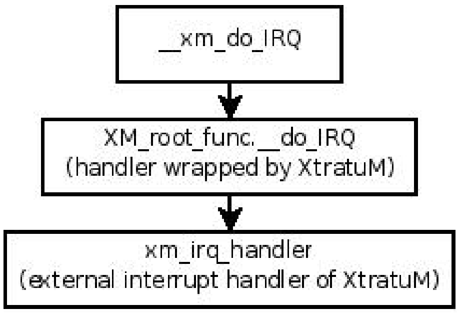

__xm_do_IRQ
XM_root_func.__do_IRQ
(handler wrapped by XtratuM)
xm_irq_handler
(external interrupt handler of XtratuM)

Figura 22: Transfer external interrupt handler from XtratuM

5.4. Memory

For memory management, the constraint imposed by XtratuM is that the memory map of the partitions, except the root partition, must be at least logically separated from the kernel map. XtratuM and the root partition, which are treated together as the kernel, share the same memory space as the whole memory range from 0x0 to 0xFFFFFFFF, while, the rest of the partitions are treated as running in unprivileged user space in different memory maps. Any try of the application of writing outside of its segment limits raises a processor exception which is caught by the kernel [53].

XtratuM provides an isolated virtual memory map per loaded partition. Basically, the partition itself will be always loaded in the range from 0x100000 to 0xBFFFFFFF, known as the partition memory range. And XtratuM will create a virtual memory map in kernel space with the partition's image. Besides mapping the partition, XtratuM also maps two additional things in the partition memory range: a shared page which will be used by the partition to install the event handlers to enable and disable an event, and an extra free area of memory to allow the partition to use it as a conventional memory heap, that is, an area of dynamic memory. In the x86 implementation, it is not XtratuM who decides the virtual map of a partition; this decision is taken by the system linker [53]. In PowerPC, in consideration of the various members of PowerPC family covering from 32-bit to 64-bit processors, XtratuM additionally implements four-level paging, which is compatible with both 32-bit and 64-bit architectures, instead of the original Linux two-level paging in x86 to provide a more general paging method [57] (Figure 23).

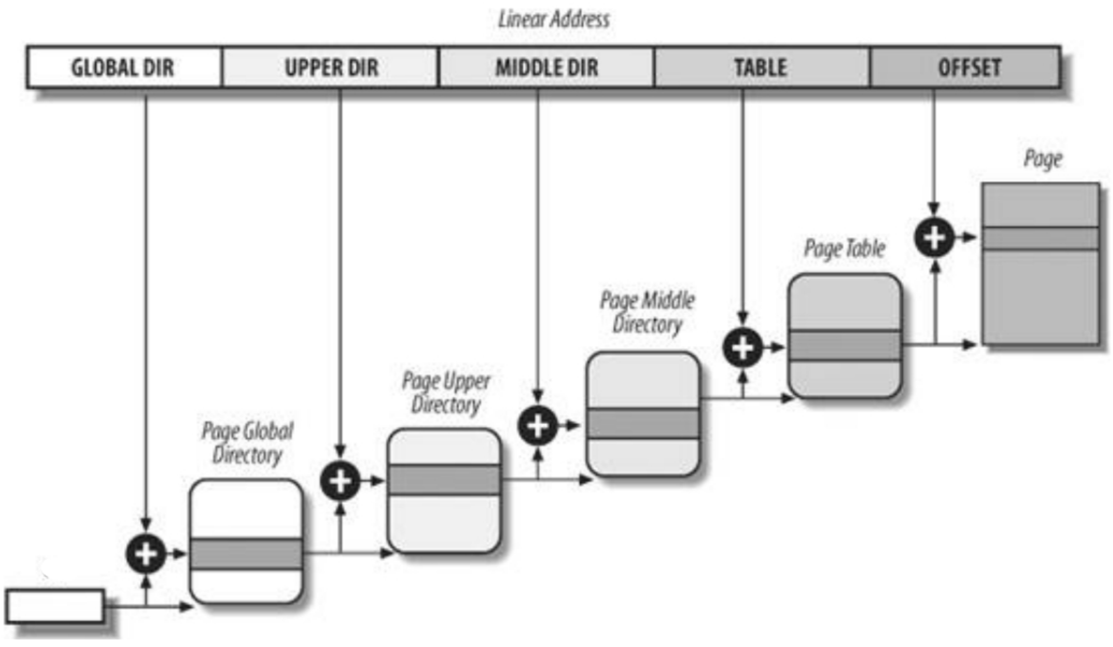


Figura 23: Four-level paging model [57]

5.5. Partition

There are also some necessary tools provided by XtratuM for partition usage, including a library and a loader. Also, there is a simple example of the partition to demonstrate the basic functions of XtratuM in PowerPC. They are almost kept the same implementation as in x86.

The library is to ease the building and the porting of new partitions to XtratuM. It basically provides a C-language interface to easily invoke XtratuM hypercalls as well as some extra functions not performed by XtratuM hypercalls, such as [53]:

- **int install_event_handler (int event, handler t h).** The function *h* is installed as the handler for the event *event*. This function returns 0 on success or an error if the parameter *event* is greater than the number of existing events.

- **void enable_event_flag (void).** This function sets the event flag, instructing XtratuM to deliver events to the partition.
- **void disable_event_flag (void).** When the event flag is unset, XtratuM blocks all the events to the partition.
- **int is_event_flag_enabled (void).** This function returns the current state of the event flag.
- **int mask_event (int event).** This functions allows to block a single event. On success, it returns 0. An error is returned when the parameter *event* is greater than the number of existing events.
- **int unmask_event (int event).** This function releases a blocked event. On success, it returns 0. As in the *mask_event* function, an error is returned when the parameter event is greater than the number of existing events.
- **void set_mask_events(bitmap t *new, bitmap t *old).** This function installs a new event mask, returning the previously one.

One important point in this library is the hypercall explaining syntax interface between the partition and kernel. The implementation of this interface is in the form of assembly, so it is totally different for PowerPC and x86.

The loader is in charge of processing the partition ELF file and creating a processed image. When the loader winds up this image, it calls the *load_domain* hypercall. One of the parameters of this hypercall is a pointer to the created image which will be used by XtratuM to finish the correct loading and to start its execution. When the loading finishes an identifier is returned. This identifier can be used later on to unload the partition. The only exception to this is the root partition, which can be never unloaded. In addition, the loader can also be used to unload any XtratuM partition. The only parameter to carry out this operation is the identifier of the partition to be removed from XtratuM [53].

To facilitate the partition load and unload, two commands are provided by the loader [53]:

- loader.xm: loads a new partition. The command parameters are:
  $ loader.xm <partition_image> [-prio <prio>] [-dname <partition_name>]
  The <partition_image> is the name of the ELF file to be loaded as a XtratuM partition, <prio> indicates the priority and <partition_name> is a string which will be show as the partition name.
  For example, the next invocation of loader.xm will load the file trivial.xmd creating a new partition with a priority of 30, which will be named "trivial".
  $ loader.xm trivial.xmd -prio 30 -dname trivial
  Once the command is executed, it will return the next message:
  >> Loading the partition "trivial.xmd (trivial) ... Ok (Id: 1)"
  The identifier returned (1) will be required later to unload the partition "trivial".

- unloader.xm: unloads an already existing partition. The command parameters are:
  $ unloader.xm -id <partition_id>
  The <partition_id> is the identifier returned by load.xm command when executed successfully.
  To unload the partition loaded in the previous load.xm example, it would be necessary to execute the unload.xm command as:
  $ unloader.xm -id 1

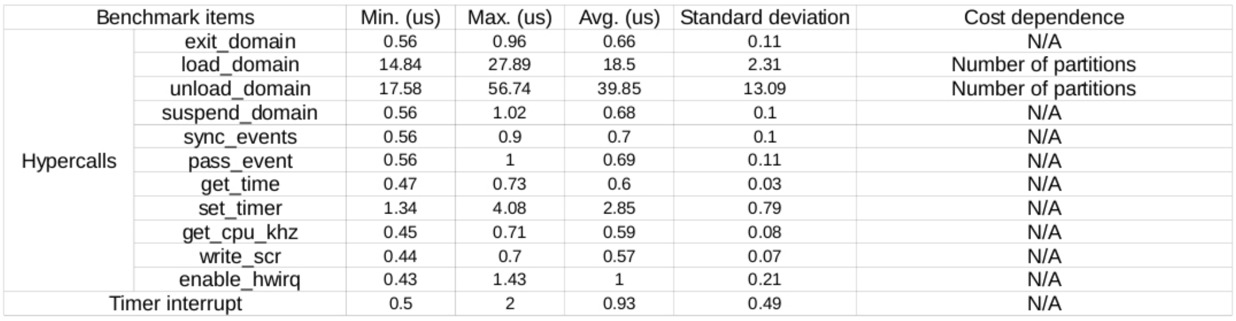
The basic example is a partition assigned with a customized priority and a period. After it is loaded once by the loader, there are one XtratuM partition and the root partition Linux existing. As Linux owns the lowest priority, within the time of one period, the XtratuM partition will be suspended at the end of execution, and XtratuM will switch to Linux. Until the next period coming, the XtratuM partition will be scheduled again for executing, and then Linux again. This kind of iteration is the implementation of FP (Fixed Priority) mechanism.

6. Benchmark

With the current implementation of XtratuM in PowerPC, it is necessary to get a global and fair overview of the performance. So some initial benchmarks have been carried out. The technical criteria focused on include timer interrupt latency, hypercall latency and scheduling issues including context switch.

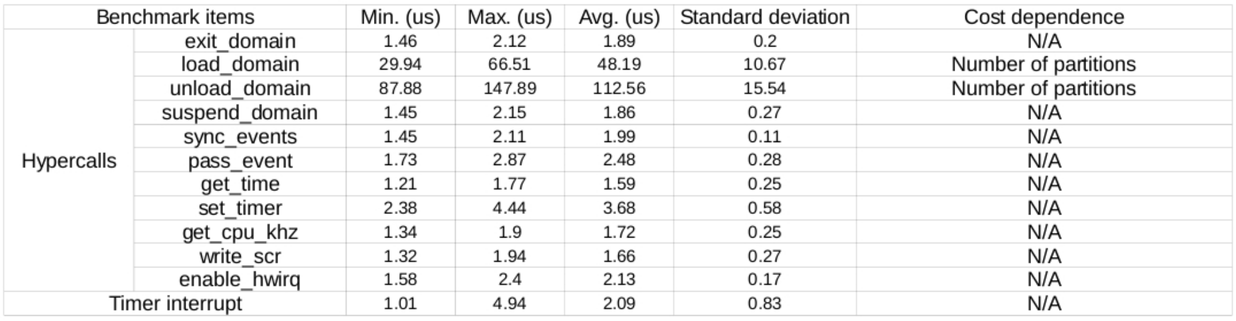
The benchmark environment includes:

- The implementation of XtratuM for PowerPC is based on XtratuM 1.0 and Linux kernel 2.6.19.2 from ELDK (Embedded Linux Development Kit) 4.1.
- The file system is provided by ELDK 4.2, and mounted over NFS (Network File Systems).
- All of these software components are cross-compiled for PowerPC 4xx processors.
- The cross-compiling environment is also from ELDK 4.1 (gcc version 4.0.0 (DENX ELDK 4.1 4.0.0)).
- The hardware parts include Yosemite AMCC PowerPC440EP Evaluation Board (AMCC PowerPC 440EP Processor at 533MHz, 256MB DRAM and 64MB FLASH), OCTOBUS BASIC developing board with DAVE PPChameleonEVB CPU module (AMCC PowerPC 405EP Processor at 133MHz, 32MB DRAM and 4MB FLASH), and BDI2000 debugger.



Benchmark items		Min. (us)	Max. (us)	Avg. (us)	Standard deviation	Cost dependence
Hypercalls	exit_domain	0.56	0.96	0.66	0.11	N/A
	load_domain	14.84	27.89	18.5	2.31	Number of partitions
	unload_domain	17.58	56.74	39.85	13.09	Number of partitions
	suspend_domain	0.56	1.02	0.68	0.1	N/A
	sync_events	0.56	0.9	0.7	0.1	N/A
	pass_event	0.56	1	0.69	0.11	N/A
	get_time	0.47	0.73	0.6	0.03	N/A
	set_timer	1.34	4.08	2.85	0.79	N/A
	get_cpu_khz	0.45	0.71	0.59	0.08	N/A
	write_scr	0.44	0.7	0.57	0.07	N/A
	enable_hwirq	0.43	1.43	1	0.21	N/A
Timer interrupt		0.5	2	0.93	0.49	N/A

Figura 24: 440EP benchmark summary



Benchmark items		Min. (us)	Max. (us)	Avg. (us)	Standard deviation	Cost dependence
Hypercalls	exit_domain	1.46	2.12	1.89	0.2	N/A
	load_domain	29.94	66.51	48.19	10.67	Number of partitions
	unload_domain	87.88	147.89	112.56	15.54	Number of partitions
	suspend_domain	1.45	2.15	1.86	0.27	N/A
	sync_events	1.45	2.11	1.99	0.11	N/A
	pass_event	1.73	2.87	2.48	0.28	N/A
	get_time	1.21	1.77	1.59	0.25	N/A
	set_timer	2.38	4.44	3.68	0.58	N/A
	get_cpu_khz	1.34	1.9	1.72	0.25	N/A
	write_scr	1.32	1.94	1.66	0.27	N/A
	enable_hwirq	1.58	2.4	2.13	0.17	N/A
Timer interrupt		1.01	4.94	2.09	0.83	N/A

Figura 25: 405EP benchmark summary

Each item for test is repeated for 100 times. Note that the loaded/unload domain is performed by the loader, which does not relate to real-time issues.

Interrupt latency is usually the time elapsed from assertion of hardware interrupt until then start of the interrupt handler execution. Timer interrupt is an important one for a real-time hypervisor. For timer interrupt latency, this period of time covers from the point when entering the interrupt vector (*__xm_timer_interrupt*) until the entry of timer interrupt handler (*timer_handler*) provided by XtratuM (Figure 18). As an internal interrupt in PowerPC, the handling process depends more on the processor, which makes it more efficient. From the results (Figure 24 and 25), for 405EP, timer interrupt latency is usually between 1us and 3us and the worst case is close to 5us; for 440EP, it is not more than 2us. This result is satisfactory as real-time performance. And the small standard deviation shows this latency does not spread out over a large range of values.

Hypercall latency shows the overhead on individual hypercall, as it measures solely the cost of communicating between XtratuM partition as the user space and kernel. For all hypercalls, the latency is calculated beginning from the hypercall function in partition to the entry of corresponding implementation in kernel. From the results (Figure 24 and 25), usually *exit_domain*, *suspend_domain*, *sync_events*, *pass_event*, *get_time*, *get_cpu_khz*, *write_scr* and *enable_hwirq* all hold latencies around or less than 3us for 405EP, and less than 2us for 440EP, which are good results for hypercalls. Though *set_timer* has a larger latency, it is still less than 5us for both 405 and 440, which is acceptable. The standard deviations of these items are all small values, which imply that these latencies are kept in a stable level. For *load_domain* and *unload_domain*, the latency is nearly linearly ascending with the number of existing partitions, which is the reason for high standard deviation. These two hypercalls are only invoked by Linux not XtratuM partitions to load/unload partitions, so the real-time performance is not compulsory for them.

As a hypervisor aims at real-time applications, scheduling is indispensable. The benchmark here is mainly going to test two issues: the one is whether the applied scheduling policy follows the design, and the other is the cost of context switch. This test is only carried out in PowerPC 440EP.

As mentioned before, the basic scheduling policy implemented is FP (Fixed Priority). For the test, one schedulable scenario is created as there are three partitions: P1, P2 and P3. They are loaded in the order of P1, P2 and P3 with different priorities and periods assigned (Figure 26).

	Priority	Period
P1	30	0.5s
P2	40	1s
P3	50	0.5s
P0 (Linux)	1023	

Figura 26: Partitions for scheduling test

Theoretically, the scheduling should be obtained as Figure 27.

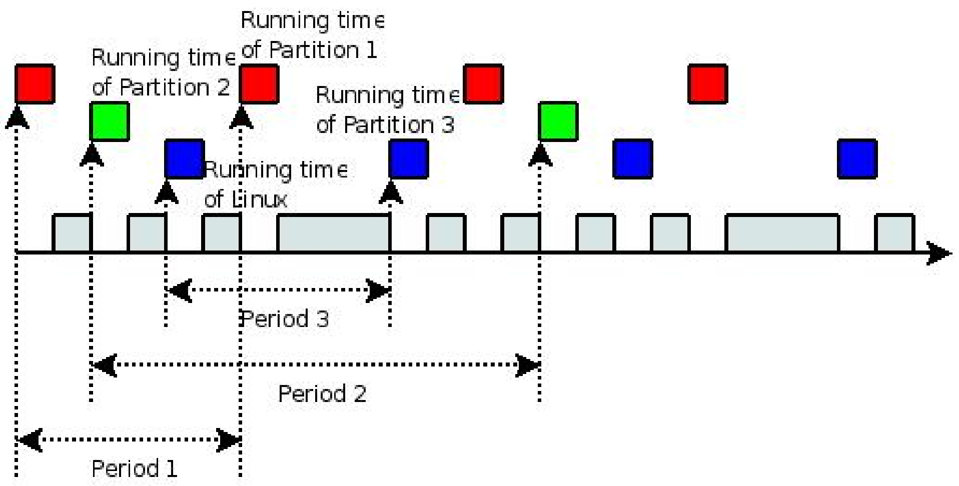
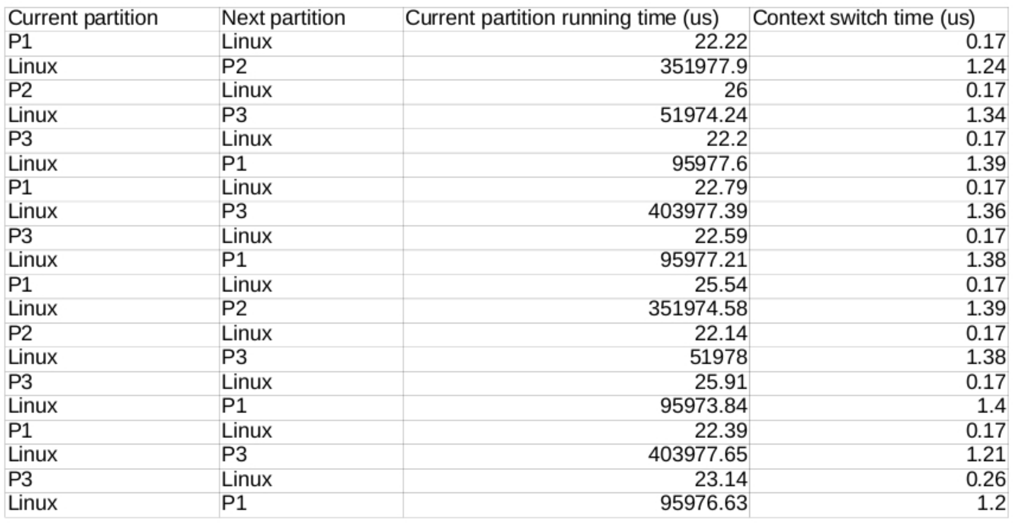

Running time of Partition 1
Running time of Partition 2
Running time of Partition 3
Running time of Linux
Period 3
Period 2
Period 1

Figura 27: FP scheduling model



Current partition	Next partition	Current partition running time (us)	Context switch time (us)
P1	Linux	22.22	0.17
Linux	P2	351977.9	1.24
P2	Linux	26	0.17
Linux	P3	51974.24	1.34
P3	Linux	22.2	0.17
Linux	P1	95977.6	1.39
P1	Linux	22.79	0.17
Linux	P3	403977.39	1.36
P3	Linux	22.59	0.17
Linux	P1	95977.21	1.38
P1	Linux	25.54	0.17
Linux	P2	351974.58	1.39
P2	Linux	22.14	0.17
Linux	P3	51978	1.38
P3	Linux	25.91	0.17
Linux	P1	95973.84	1.4
P1	Linux	22.39	0.17
Linux	P3	403977.65	1.21
P3	Linux	23.14	0.26
Linux	P1	95976.63	1.2

Figura 28: Scheduling test results

Then from the test (Figure 28), the order of executing partitions show that each partition is scheduled follows the desired model. So the basic scheduling policy of FP can work in this implementation.

One section is taken from the infinite scheduling loop. This section also shows the order of executing partitions, and the time arguments can be achieved. The partition running time is from the moment between two continuous context switches just finish, which means the current partition has just be updated. Therefore, the time cost of context switch is included in the partition running time. As the partition loaded in this test only acts as simple as print a string, it only takes around 22us for each partition execution. This value is much smaller than the period assigned, so after that, Linux will be invoked to run until the next partition is scheduled to execute. And the context switch results show that for switch from Linux to the partition, the time cost is around 1.1us, which seems to be a small value. While for switch from partition to Linux, it is even much faster as around 0.17us. Compared with partition running time, context switch only occupies less than 1 % overhead for each running partition. Meanwhile, as the partitions are assigned with different periods that are satisfied during execution, it expresses the temporal isolation by separate virtual timers for each partition by XtratuM.

Additionally, another "pure"test for context switch is done as with only one partition loaded. Because the partition is scheduled to execute and then suspend and then execute in a *while(1)* loop, the context switch will always happen between this loaded partition and Linux. The results show the same trend as the former test.

7. Conclusions and Future Work

The object of this thesis is to show the efforts to implemented an embedded-oriented hypervisor in real embedded hardware. As XtratuM refers to the idea of partitioned system, the implementation always covers this concept, especially for the key devices of an operating system: timer, interrupt and memory. With the different hardware specifications between PowerPC and x86, those key devices are manipulated properly to achieve the goal of spatial and temporal isolation of partitions. According to the preliminary benchmarks, different partitions could execute normally and be scheduled following the FP scheduling policy.

Based on this implementation, there can be more ideas about embedded applications, safety-critical applications and real-time capabilities developed and applied for PowerPC:

- At present, FP scheduling policy is deployed in the hypervisor level for XtratuM in PowerPC, which is just a basic scheduling policy. In order to evaluate the implementation in detail, and in consideration of the diversity of future applications, it is necessary to apply other scheduling policies in the hypervisor, such as EDF (Earliest Deadline First), FPS (Fixed Priority Server), to achieve more useful evaluation results that would be beneficial for theory research.
- Another issue is the communication between partitions. As ARINC 653 specifies the sampling port and queuing port for inter-partition communication services, and there have been other ways such as XM/FIFO and SHM implemented in x86 for XtratuM, it is feasible to apply these mechanisms and comparing the performance as one item of the following work.
- XtratuM is a hypervisor designed to support high-level partitions, but at present, there is only one trivial partition example show the basic functionalities. This is not enough for demonstration, so there is a need for the implementation of several partitions compliant with XtratuM in PowerPC. These partitions could be RTOSes, bare applications, or device drivers, etc. In the way of para-virtualization, all these potential partitions must be updated compliant with XtratuM. Currently, an ongoing project is enabling PaRTiKle, a embedded real-time operating system designed to be POSIX compliant, on top of XtratuM in PowerPC. This will enhance the real-time applications for XtratuM in PowerPC, and achieve more interesting results from research. Also, the partition of RTOS will introduce partition level scheduling issues, so the implementation of two-level scheduling is much more complex and practical in research area.
- Once there is support for RTOS by XtratuM in PowerPC, more interesting issues could be deployed. There has been a design of Redundant Real-Time Control System (RRTCS) based on XtratuM in PowerPC for distributed machine tool controlling (Figure 29). It covers the topics including real-time applications, safety-critical capabilities, control and device drivers, etc. This could be a wonderful theoretical and practical demonstration for both XtratuM and PowerPC.

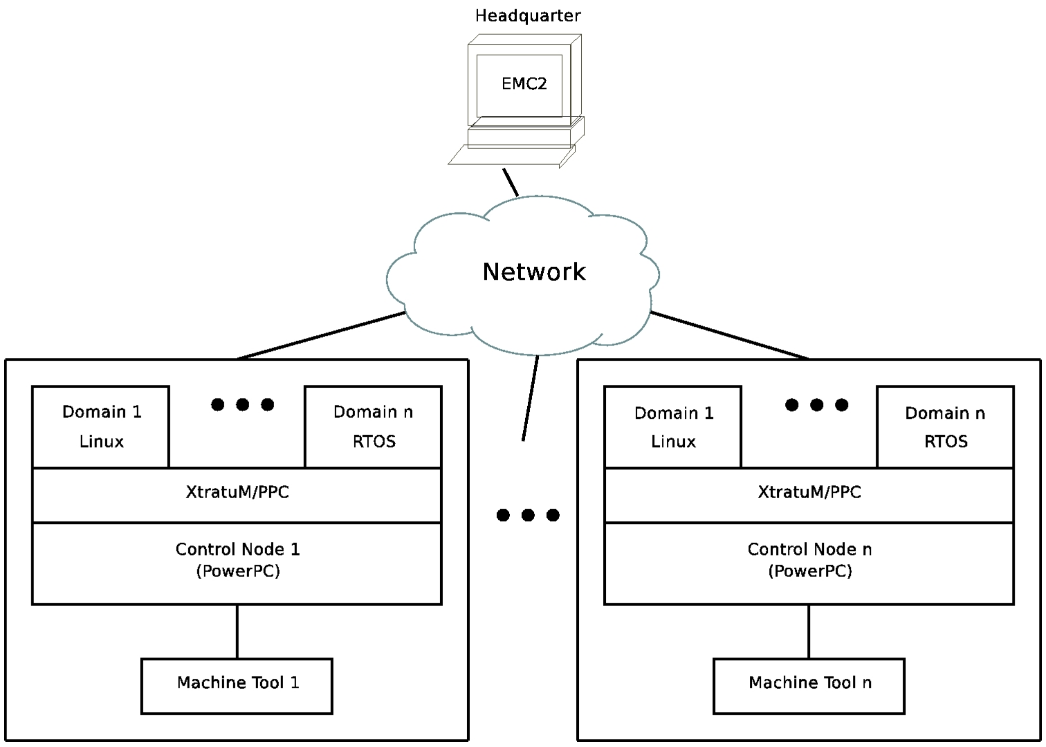


Figura 29: Redundant Real-Time Control System with XtratuM in PowerPC

- Furthermore, as XtratuM 2.x is totally changed compared to XtratuM 1.x, it is of course an important issue to see if it can be implemented in PowerPC.

Referencias

[1] Joyce L. Tokar, *Space & Time Partitioning with ARINC 653 and pragma profile*, IRTAW '03: Proceedings of the 12th international workshop on Real-time Ada, ACM, 2003, ISSN:1094-3641

[2] Tim Skutt, *Software Partitioning Technologies*, `http://www.dtic.mil/ndia/2001technology/skutt.pdf`, Smiths Aerospace, 2001

[3] Daeyoung Kim, *Strongly Partitioned System Architecture for Integration of Real-Time Applications*, Dissertation, University of Florida, 2001

[4] W. Mark Vanfleet and Jahn A. Luke, et al., *MILS: Architecture for High-Assurance Embedded Computing*, `http://www.stsc.hill.af.mil/crosstalk/2005/08/0508Vanfleet_etal.html`

[5] John Rushby, *Design and Veri cation of Secure Systems*, the 8th ACM Symposium on Operating System Principles, Pacific Grove, California, 14-16 December 1981. (ACM Operating Systems Review Vol. 15 No. 5 pp. 12-21)

[6] W. Scott Harrison and Nadine Hanebutte, et al., *The MILS Architecture for a Secure Global Information Grid*, `http://www.stsc.hill.af.mil/CrossTalk/2005/10/0510Harrisonetal.html`

[7] Multiple Independent Levels of Security from Wikipedia, `http://en.wikipedia.org/wiki/Multiple_Independent_Levels_of_Security`

[8] ARINC 653 from Wikipedia, `http://en.wikipedia.org/wiki/ARINC_653`

[9] A. Cook, *ARINC 653 - Challenges of the present and future*, Microprocessors and Microsystems, Volume 19, Issue 10, December 1995, Pages 575-579, Elsevier Science B.V., doi:10.1016/0141-9331(96)84158-5

[10] Julien Delange and Laurent Pautet, et al., *Validating Safety and Security Requirements for Partitioned Architectures*, Lecture Notes in Computer Science, Volume 5570/2009, Reliable Software Technologies - Ada-Europe 2009, Pages 30-43, Springer Berlin / Heidelberg, ISBN 978-3-642-01923-4

[11] J. Rufino and J. Craveiro, *Robust Partitioning and Composability in ARINC 653 Conformant Real-Time Operating Systems*, The 1st INTER-AC Research Network Plenary Workshop, Braga, Portugal, July 2008

[12] M. Masmano, I. Ripoll, and A. Crespo, *An overview of the XtratuM nanokernel*, Workshop on Operating Systems Platforms for Embedded Real-Time applications, Universidad Politecnica de Valencia, Spain, 2005

[13] A. Cook and K.J.R. Hunt, *ARINC 653 - Achieving software re-use*, Microprocessors and Microsystems,Volume 20, Issue 8, April 1997, Pages 479-483, Elsevier Science B.V., doi:10.1016/S0141-9331(97)01113-7

[14] SYSGO Product Datasheet of PikeOS, http://www.sysgo.com/fileadmin/user_upload/datasheets/PikeOS.pdf

[15] Rudolf Fuchsen, *Virtualization concepts for safety-critical systems*, Boards & Solutions - The European Embedded Computing Magazine, April 2008, Pages 36-37

[16] PikeOS from Wikipedia, http://en.wikipedia.org/wiki/PikeOS

[17] Adaptive Domain Environment for Operating Systems from Wikipedia, http://en.wikipedia.org/wiki/Adeos

[18] Karim Yaghmour, *Adaptive Domain Environment for Operating Systems*, Whitepaper, http://opersys.com/ftp/pub/Adeos/adeos.pdf

[19] Edgar Manuel Candido da Silva Pascoal, *AMOBA - ARINC653 Simulator for Modular Space Based Applications*, Dissertation, Universidade de Lisboa, 2008

[20] Green Hills Software Inc., *Safety Critical Products: INTEGRITY-178B RTOS*, http://www.ghs.com/products/safety_critical/integrity-do-178b.html

[21] Chip Downing, *Use of ARINC 653 and Integrated Modular Avionics to Reduce Space, Weight, and Power Consumption in Unmanned Vehicles*, Wind River Systems Inc., 2009

[22] Yang Chang and Robert Davis, et al., *Schedulability Analysis for a Real-time Multiprocessor System Based on Service Contracts and Resource Partitioning*, University of York, UK, http://www.cs.york.ac.uk/ftpdir/reports/2008/YCS/432/YCS-2008-432.pdf

[23] L. M. Kinnan, *Application migration from Linux prototype to deployable IMA platform using ARINC 653 and Open GL*, IEEE/AIAA 26th Digital Avionics Systems Conference, Oct 2007

[24] Fawzi Behmann, *Virtualization for embedded Power Architecture CPUs*, Electronics Products September 2009 Issue, Pages 32-33

[25] Rune Johan Andresen, *Virtual Machine Monitors*, http://openlab-mu-internal.web.cern.ch/openlab-mu-internal/Documents/2_Technical_Documents/Summer_Students/vmm.pdf

[26] Hypervisor from Wikipedia, http://en.wikipedia.org/wiki/Hypervisor

[27] Miguel Masmano, Ismael Ripoll, Alfons Crespo and Vicent Brocal, *XtratuM Hypervisor for LEON2 Volume 2: User Manual*, Industrial Informatics and Real-Time Systems Group, Universidad Politecnica de Valencia, September 2009

[28] XtratuM, http://www.xtratum.org

[29] XtratuM from Wikipedia, http://en.wikipedia.org/wiki/XtratuM

[30] Real-Time Systems GmbH, http://www.real-time-systems.com/real-time_hypervisor

[31] TRANGO, http://www.trango-vp.com

[32] LynxSecure, http://www.lynuxworks.com/virtualization/hypervisor.php

[33] Gernot Heiser, *Technology White Paper - Virtualization for Embedded Systems*, http://www.ok-labs.com/_assets/download_library/OK_Virtualization_WP.pdf, Open Kernel Labs, Inc., 2007

[34] Rui Zhou and Baojun Wang, et al., *XtratuM for PowerPC*, Proceeding of The Ninth Real-Time Linux Workshop, Pages 163-168, Linz, Austria, November 2007

[35] M. Masmano, I. Ripoll, and A. Crespo, et al. *XtratuM: a Hypervisor for Safety Critical Embedded Systems*, Proceeding of The Eleventh Real-Time Linux Workshop, Pages 263-272, Dresden, German, 2009

[36] International Business Machines Corporation, *Power Architecture: A High-Performance Architecture with a History*, http://www-03.ibm.com/systems/p/hardware/whitepapers/power/ppc_arch.html

[37] PowerPC from Wikipedia, http://en.wikipedia.org/wiki/PowerPC

[38] LynuxWorks Inc., *RTOS for Software Certification: LynxOS-178*, http://www.lynuxworks.com/rtos/rtos-178.php

[39] Christoph Baumann and Thorsten Bormer, *Verifying the PikeOS microkernel: First results in the Verisoft XT Avionics Project*, In Ralf Huuck, Gerwin Klein, and Bastian Schlich, editors, Doctoral Symposium on Systems Software Verification (DS SSV'09), number AIB-2009-14 in Aachener Informatik Berichte, pages 20-22. Department of Computer Science, RWTH Aachen, June 2009.

[40] Christoph Baumann and Bernhard Beckert, et al., *Formal verification of a microkernel used in dependable software systems*, In Bettina Buth, Gerd Rabe, and Till Seyfarth, editors, Computer Safety, Reliability, and Security (SAFECOMP 2009), volume 5775 of Lecture Notes in Computer Science, pages 187-200, Hamburg, Germany, 2009. Springer.

[41] Wind River, *Wind River VxWorks 653 Platform*, http://www.windriver.com/products/platforms/safety_critical_arinc_653

[42] Electronic Product News, *Lockheed selects AdaCore's development environment*,

http://www.epn-online.com/page/new118397/lockheed-selects-adacore-s-development-environment.html

[43] Sean Kalinich, *Networking in Space: Or The Longest Ethernet Cord Ever*, http://www.brightsideofnews.com/news/2009/6/3/networking-in-space-or-the-longest-ethernet-cord-ever.aspx

[44] Power Architecture from Wikipedia, http://en.wikipedia.org/wiki/Power_Architecture

[45] List of PowerPC processors from Wikipedia, http://en.wikipedia.org/wiki/List_of_PowerPC_processors

[46] PIKA Technologies Inc., *PIKA Technologies Selects AMCC*, http://www.pikatechnologies.com/english/View.asp?mp=803&x=978

[47] Time Stamp Counter from Wikipedia, http://en.wikipedia.org/wiki/Time_Stamp_Counter

[48] Intel 8253 from Wikipedia, http://en.wikipedia.org/wiki/Intel_8253

[49] Applied Micro Circuits Corporation, *PPC440 Processor User's Manual*, Revision 1.04, April 21, 2006

[50] Interrupt Descriptor Table from Wikipedia, http://en.wikipedia.org/wiki/Interrupt_descriptor_table

[51] Claudia Salzberg Rodriguez, Gordon Fischer and Steven Smolski, *The Linux Kernel Primer: A Top-Down Approach for x86 and PowerPC Architectures*, Prentice Hall PTR, September 2005, ISBN: 0-13-118163-7

[52] Intel 8259 from Wikipedia, http://en.wikipedia.org/wiki/Intel_8259

[53] Miguel Masmano, Ismael Ripoll and Alfons Crespo, et al, *Framework for Real-time Embedded Systems based on COntRacts: Nanokernels for multidomain support*, Industrial Informatics and Real-Time Systems Group, Universidad Politecnica de Valencia, 2005

[54] Shuwei Bai, Yiqiao Pu and Kairui She, et al., *XM-FIFO: Interdomain Communication for XtratuM*, Proceeding of The Ninth Real-Time Linux Workshop Proceeding, p151-162, Linz, Austria, November 2007

[55] Kairui She, *Share Memory in XtratuM/RTLinux*, Technical Report, Distributed and Embedded System Lab, Lanzhou University, P. R. China, 2007

[56] Kernel Based Virtual Machine, http://www.linux-kvm.org/page/PowerPC

[57] Daniel P. Bovet and Marco Cesati, *Understanding Linux Kernel, 3rd Edtion*, O'Reilly, 2005, ISBN: 0-596-00565-2

Acknowledgement

I would like to show respect and gratitude for my advisor Prof. Alfons Crespo i Lorente for providing me the opportunity, financial support and guidance for this work, and his help for my living and studying in Spain. His stimulating suggestions, sincere encouragement and straight-out attitude always help me in all the time of research.

I would like to thank all the committee members, all the colleagues in the laborary, and all the teachers and classmates in the master courses. They help me a lot during the study of master courses.

The colleagues in Distributed and Embedded System Lab of Lanzhou University, are always my good companions.

I really appreciate my parents for their long-term care and support for my life and study. They are my most valuable treasure in the world.