Proteus, a Hybrid Virtualization Platform for Embedded Systems

Daniel Baldin and Timo Kerstan

Heinz-Nixdorf-Institute
University of Paderborn
33102 Paderborn, Germany
dbaldin@uni-paderborn.de,
timo.kerstan@uni-paderborn.de

Abstract. By the use of virtualization the security of a system can be significantly increased and performance can be improved by sharing hardware resources while reducing the overall costs of the whole system. Nowadays virtualization also finds approval within the field of embedded systems. However, the currently available virtualization platforms designed for embedded systems only support para-virtualization trying to provide reasonable performance and support realtime applications only by the use of dedicated resources. Our approach introduces a hybrid configurable hypervisor architecture designed to support real-time applications. We do not restrict the set of applications which can be run virtualized on top of our hypervisor to para-virtualized applications but also allow applications to run unmodified or even partly para-virtualized while using state of the art methodologies to obtain high performance.

Introduction

In the last decade, embedded systems tended to become increasingly complex systems while retaining the requirements of security and robustness. This inherent problem has usually been faced by the use of redundancy in which the whole hardware and software system is replicated multiple times to compensate a broken system. Additionally, embedded systems applications also often serve completely unrelated purposes which raises the need of being isolated from each other. Virtualization offers a solution to all of these problems, allowing the coexistence of multiple virtual machines on one hardware platform. By the use of virtual resources, the security of a system can be significantly increased [Heiser, 2008] [Chen and Noble, 2001] [Smith and Nair, 2005] and performance can be gained by sharing hardware resources while reducing the overall costs of the whole system [Chen and Noble, 2001]. The rapid increasing performance of embedded system hardware and the increasing availability of memory protection mechanisms inside embedded system processors make virtualization inside embedded systems more and more attractive.

A. Rettberg et al. (Eds.): IESS 2009, IFIP AICT 310, pp. 185–194, 2009.

1 Related Work

There already exist a few commercial virtualization platforms or hypervisors for a range of embedded processors with nearly all of them being proprietary systems. Trango [VmWare, 2009] and VirtualLogix [VirtualLogix, 2009] are two of those hypervisors allowing virtualization for a range of ARM and MIPS processors. Greenhills Integrity [Hills, 2009] and LynxSecure [LynuxWorks, 2009] are using virtualization to implement high security systems targeted for the military market and have been certified to fulfill the EAL7 standard. All of these available products only use para-virtualization trying to provide reasonable performance and support realtime applications only by the use of dedicated resources. Naturally, this limits the applicability of virtualization using these products to a subset of all possible scenarios. Especially, whenever there are applications that can not be para-virtualized since the source code is not available these applications can not be virtualized using the currently available virtualization products as this would require a binary analysis of the whole application which most often is not completely possible [Laune C. Harris, 2006] [Saumya Debray, 1998].

Our approach overcomes this problem by introducing a hybrid configurable hypervisor architecture designed and implemented for the PowerPC405 processor which allows the virtualization of unmodified applications as well as para-virtualized applications or a combination of both. Support for realtime applications was a major goal of our design which allows the integration of any kind of scheduling mechanism for virtual machines while being completely deterministic. Additionally, the high configurability allows the system to be optimized explicitly for the intended field of use.

2 Hybrid Architecture

The basic requirements which the design of this virtualization platform faces are:

- Security
- Real-time capabilities
- High performance
- Configurability
- Minimal demand on memory and energy consumption

In order to meet the requirements of high performance and configurability, the virtualization platform uses a hybrid architecture as depicted in figure 1. Therefore, it represents a hybrid hypervisor which uses full-virtualization as the basic virtualization technique with additional support for para-virtualization. Partially para-virtualized applications or self modifying applications are also supported by the use of a fallback feature which ensures the consistency while executing in and switching between both kind of states. The configurability of the hypervisor and the para-virtualization interface allows the system designer to create a system which is tailored for the particular field of application and saves as much memory as possible.

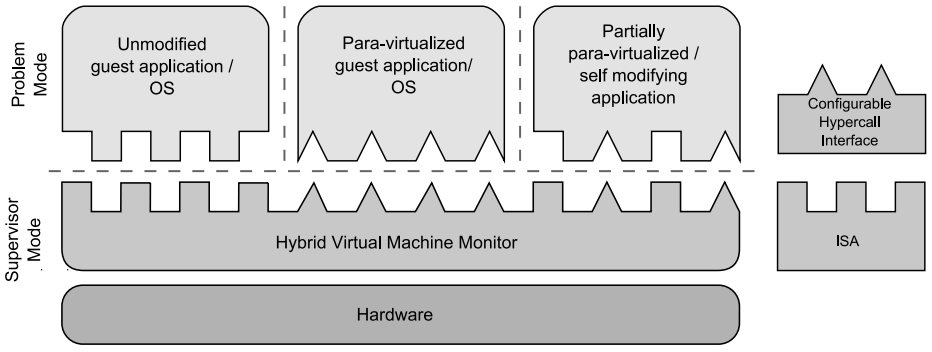

Fig. 1. The virtual machine monitor allows the virtualized execution of any kind of guest application. Left: a unmodified application. Middle: a completely para-virtualized application. Right: a partially modified application.

In general, the design is based on the multiple independent levels of security (MILS) architecture [Systems, 2008]. As illustrated in figure 2, only the minimal set of components which are needed to implement the secure partitioning, scheduling and communication between virtual machines are part of the hypervisor running in privileged mode. All other components called "Untrusted VMP Modules" are placed inside a separate partition and are executed in user mode as this is one of the fundamental concepts of the MILS architecture. The communication between partitions is controlled by the Inter Partition Communication Manager (IPCM) who, on request, creates shared memory tunnels between two partitions which can be used for virtual machines to ease the communication between each other. Especially this feature allows formerly physically spread systems which had to use real-time capable bus systems (e.g. a CAN-bus) for communication to enhance the security, robustness and performance of the information flow between each other if they are virtualized and placed on top of the same hypervisor.

The fundamental components of the whole system form the interrupt handler as seen in figure 2. Since the hypervisor executes the guest application in user mode, any interrupt occurring is first delegated to the hypervisor which then has to analyze the interrupt and forward it to the appropriate component or the virtual machine itself. Program interrupts raised amongst others by privileged instructions inside the guest application are forwarded to the emulation routine dispatcher which will determine the corresponding ISA emulation routine. Whenever the virtualization platform has been configured to support para-virtualization, applications running inside a virtual machine can use hypercalls to emulate privileged instructions, call the IPCM, use para-virtualized I/O drivers or call scheduler related functions. Especially the last component offers methods to set scheduler related parameters at runtime or the possibility to yield the cpu directly in order to allow system designers to incorporate special scheduling mechanisms.

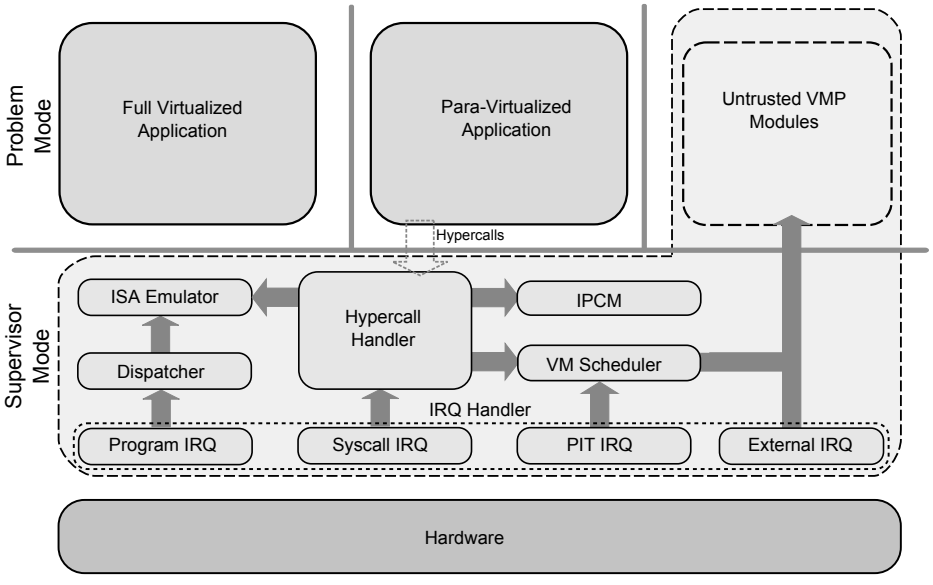

Fig. 2. Information and control flow of the components used inside the virtualization platform

3 Concepts

3.1 Configurability

Achieving only a small overhead as needed for the intended field of use was one of the major design goals of this system. This goal is met by the high flexibility offered through the possibility to configure a wide range of system components. Using the provided configuration files, the system designer is able to enable or disable features of the virtual machine monitor or specify which parts of the hypercall interface shall be supported. The principle workflow of the configuration is depicted in figure 3. The whole configuration workflow is based on the extensive usage of preprocessor statements. Based on the configuration files the preprocessor eliminates, adds or changes code segments inside the implementation files to create a source code which does not suffer from unneeded code parts any more. This is a very valuable feature if the platform is about to be deployed inside a very memory-limited environment.

Our virtualization platform allows the system designer to explicitly define whether there is the need for full-virtualization or para-virtualization or even both. It is even possible to configure the support for special parts of the host ISA as e.g. support for virtual memory. The hypercall interface may explicitly be configured by defining whether para-virtualized drivers are supported, inter partition communication is needed and which scheduler has to be used. Each of the components can then be configured as well to allow the platform to match the needs of the target system to a maximum amount.

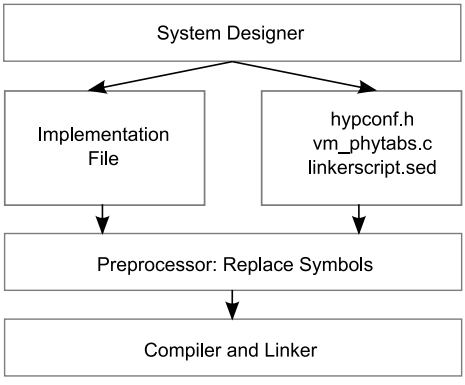

Fig. 3. Configuration flow of the virtualization platform

3.2 Innocuous Register File Mapping

Virtualizing the processor is one of the main tasks of the virtual machine monitor. Every virtual machine gets its own set of virtual registers stored inside the Register File as depicted in figure 4a. Whenever the virtual machine is activated, the registers are copied into the registers of the physical processor. During the execution the virtual machine has unrestricted access to the user space registers of the processor. Registers that need to be accessed by the use of privileged instructions will trap to the virtual machine monitor which then emulates the behaviour of that instruction on the virtual register inside the register file. However, there exist registers which can be accessed without having immediate influence on the state or behaviour of the virtual machine. An example is given by the Save Restore Registers (SRR) of the PowerPC Processor. The value of these registers will only be used whenever the processor executes the privileged "recover from interrupt" (rfi) instruction. In order to speed up the access to this kind of registers, Proteus uses a technique we call Innocuous Register File Mapping (IRFM) which allows virtual registers to be mapped to memory pages inside the virtual machines accessible memory area as illustrated in figure 4b. Virtual registers, which may be read directly without the need of any emulation, are mapped into a read-only memory page. Registers, that can be either read or written without immediate influence, are made accessible inside a readable and writable memory page whose address can be configured to conform to the systems memory map. Therefore, access to these registers is possible by using load and store commands instead of trapping to the virtual machine monitor which speeds up the virtualized execution of a program dramatically. A similar approach has been used inside the ia64 port of the Xen hypervisor [Barham et al., 2003] for desktop systems to speedup the information flow between the hypervisor and the virtual machines. However, the approach has not been applied completely to all possible registers of the ISA as this virtualization platform does.

Since the IRFM feature can only be used by para-virtualized applications, partially modified applications may still use privileged instructions to access the virtual registers. In order to ensure the consistency of the mapped registers and

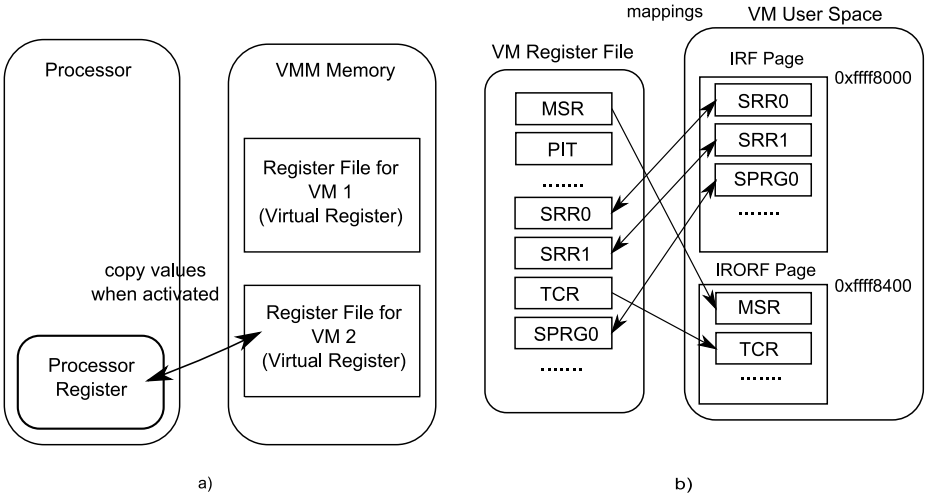

Fig. 4. a) Processor Virtualization by using Register Files. b) Innocuous Register File Mapping. Temporarily innocuous register are mapped into pages inside the VM memory space.

the virtual registers inside the register file, the virtual machine monitor updates the mapped registers whenever a value is written to the register file. This feature is called the Innocuous Register File Mapping Fallback Feature and may be enabled if the system designer is unsure whether all privileged instructions are para-virtualized or not. Especially for systems using third party libraries without access to the source code this is a very valuable feature.

3.3 Pre-virtualization Support

As some features like the IRFM are only accessible for para-virtualized applications, our virtualization platform supports a technique called Pre-Virtualization [LeVasseur et al., 2006] which is based on the principle of binary replacement. The technique is based on the idea to append "no operation" (nop) instructions to every privileged instruction inside the guests' applications source code in order to create space for a later replacement of these instructions by hypercall instructions to the virtual machine monitor. The overall process is illustrated exemplarily in figure 5. We developed a python script which first parses the source code of an application for privileged instructions. Whenever a privileged instruction is found, its address is stored inside a pre-virtualization table together with the binary code of the hypercall instructions. The code itself is only modified by appending nop instructions to the privileged instructions found. Therefore the intermediate pre-virtualized code can still be executed natively. At loadtime, the virtual machine monitor finally para-virtualizes the pre-virtualized applications by the use of the pre-virtualization table which is stored within the applications data area.

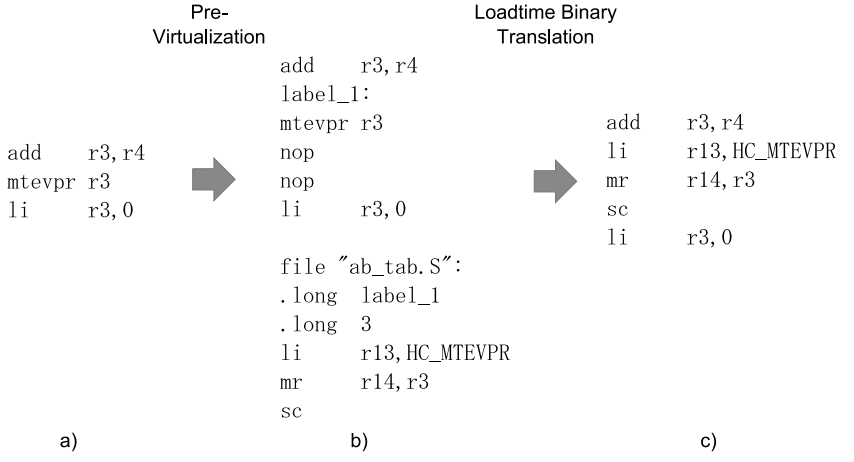

Fig. 5. Pre-Virtualization illustrated. a) Base source code. b) Pre-Virtualized source code and Pre-Virtualization table. c) The final para-virtualized code.

3.4 Programming Concept

A substantial part of the virtual machine monitor is comprised of the interrupt handler. Since a significant amount of time is spent inside these routines, most of the components called by the handlers and the handler themselves are directly written in assembly language. By using this methodology it was possible to create source code without the usual overhead induced by higher level languages like C oder C++ and EABI overheads like saving/restoring stack frames. Actually the implemented virtual machine monitor operates without using a stack as often as possible. The amount of registers used by the interrupt handler and emulation routines has been minimized as well to reduce the amount of time needed for a context switch between the virtual machine and the emulation software. Only if higher level activities are processed as e.g. calling the VM scheduler or triggering a driver, the whole context of the virtual machine needs to be saved. Additionally every operation of the virtual machine monitor has been implemented to behave deterministically in order to allow the calculation of an upper bound for the execution time of these operations.

4 Performance Figures

The performance overhead introduced by the virtualization software has been measured for a set of application scenarios. We used the real-time operating system ORCOS [University of Paderborn, 2009] running on a PowerPC405 processor to measure the overhead for two application scenarios. The first scenario used a simple periodic real-time task with the only purpose to count a variable to a specific value. The execution time was measured for an interval of two hundred repetitions to get a reasonable measurement. The overhead produced here was

Table 1. Execution in `ms` measured for a set of application scenarios and the corresponding percental lengthening of the execution time for the best case compared to the native execution time

	native	full-virt	para-virt	Overhead
ORCOS + CTask	10,7342	26,5085	17,1618	59,88%
ORCOS + FFTTask	5094,97	5123,382	5117,936	0,45%
SimpleFFT	294,564	294,564	294,564	0,00%

about 60% compared to the native execution time. The high overhead in this scenario can be easily explained by the fact that the amount of time spent inside the operating system compared to the amount of time spent inside the real-time task was really high. Thus the relative amount of privileged instructions used by the operating system which needed to be emulated was high as well leading to this kind of overhead.

A more realistic scenario was given by another real-time task which had to calculate a Fast Fourier Transformation for a set of input values for a fixed amount of repetitions. Since the relative amount of time spent for this calculation was much higher compared to the amount of emulation routines called the overhead reduced to less then one percent if the application was executed para-virtualized. It is also possible to see that running the unmodified application in full-virtualization mode is only slightly slower. The overhead increased by 0.56% in that case. Considering embedded control systems, which often consist of a comparable amount of calculations, this makes the usage of full-virtualization extremely interesting. Additionally, this observation gives reason to believe that the execution time of applications that could not be para-virtualized completely, since e.g. a part of the application has been linked against static libraries, will not suffer much by running inside our virtual machine.

Applications using only the user ISA, as given by the third example application SimpleFFT which calculated a Fast Fourier Transformation without using the support of an operating system, can be executed with native performance because no instruction needs to be emulated by the virtual machine monitor.

5 Footprint

In order to be deployed on small scale nodes, the memory overhead introduced by the virtual machine monitor needs to be as small as possible. Since Proteus is completely configurable, the amount of binary and memory footprint occupied by the virtual machine monitor strongly depends on the features used by the target system. The basis system, which already supports full-virtualization of the ISA and which is able to emulate a fairly complete set of privileged instructions needs about 4.4 Kb for the binary and about 2.5 Kb for the data structures whereas about 2.4 Kb of the data area, together with the whole text area, may be placed inside the ROM of the host platform. The biggest fraction of the data

Table 2. Binary and memory footprint of the virtualization platform in bytes for a set of configurable components for the PowerPC405 processor (32 Bit Architecture)

	.text	.data
Basis System	4348	2444
Para-Virtualization Support	252	144
IRFM Support	516	0
IRFM Fallback Support	572	0
Pre-Virtualization Support	192	0
TLB Virtualization Support	960	0
Driver Support	504	44
IPCM Support	460	4

area is used for dispatching purposes of the ISA emulation dispatcher. Adding Para-Virtualization support, including the whole IRFM feature set, increases the footprint by additional 1.5 Kb. The whole system (with all its features) has a footprint of under 11 Kb with 10 Kb being read-only data or executable code respectively which could be placed inside the ROM memory area of the platform.

The sizes listed above do not include the required space for the register files of the virtual machines. Depending on how many virtual machines the hypervisor shall support, additional 304 bytes need to be reserved for every virtual machine for our target processor the PowerPC405. If the TLB needs to be virtualized as well, in order to allow the virtual machines to use virtual memory, the reserved bytes per virtual machine increases by 956.

6 Conclusion and Future Work

In this work we presented the first hybrid configurable virtualization platform which supports full-virtualization and para-virtualization as well as a mixture of both as needed. The high configurability allows the system designer to adapt the platform for its dedicated field of use. This feature allows the system to be very small and efficient as the maximum memory overhead induced by the virtual machine monitor is below 11Kb which lets Proteus compete with current State of the Art virtualization platforms like Trango or VirtualLogix.

The system combines state of the art features like the IRFM feature and Pre-Virtualization while being implemented directly in assembler to minimize the overhead. Our newly introduced IRFM Fallback feature ensures the consistency while switching between full and para-virtualization. It is designed and implemented to support real-time applications and offers an interface to add deterministic real-time capable hierarchical scheduling mechanisms as e.g. described in [Lipari and Bini, 2005] in the future.

We are currently working on further measurements demonstrating the real-time capabilities of the system. Additionally we are working on dynamic self-optimizing features that shall be integrated into the system in the future.

References

[Barham et al., 2003] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. ACM, New York (2003)

[Chen and Noble, 2001] Chen, P.M., Noble, B.D.: When virtual is better than real. In: HOTOS 2001: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, Washington, DC, USA, p. 133. IEEE Computer Society, Los Alamitos (2001)

[Heiser, 2008] Heiser, G.: The role of virtualization in embedded systems. In: IIES 2008: Proceedings of the 1st workshop on Isolation and integration in embedded systems, pp. 11–16. ACM, New York (2008)

[Hills, 2009] Hills, G.: Real-Time Operating Systems (RTOS), Embedded Development Tools, Optimizing Compilers, IDE tools, Debuggers - Green Hills Software (2009), http://www.ghs.com/

[Laune C. Harris, 2006] Laune, C., Harris, B.P.M.: Practical Analysis of Stripped Binary Code. Technical report, Computer Sciences Department, University of Wisconsin (2006)

[LeVasseur et al., 2006] LeVasseur, J., Uhlig, V., Chapman, M., Chubb, P., Leslie, B., Heiser, G.: Pre-virtualization: soft layering for virtual machines. Technical Report 2006-15, Fakultät für Informatik, Universität Karlsruhe, TH (2006)

[Lipari and Bini, 2005] Lipari, G., Bini, E.: A methodology for designing hierarchical scheduling systems. J. Embedded Comput. 1(2), 257–269 (2005)

[LynuxWorks, 2009] LynuxWorks, Embedded Hypervisor and Separation Kernel for Operating-system Virtualization: LynxSecure (2009), http://www.lynuxworks.com/virtualization/hypervisor.php

[Saumya Debray, 1998] Debray, S., Muth, R., Weippert, M.: Alias Analysis of Executable Code. Technical report, Department of Computer Science, University of Arizona (1998)

[Smith and Nair, 2005] Smith, J., Nair, R.: The architecture of virtual machines. Computer 38(5), 32–38 (2005)

[Systems, 2008] Systems, O.I.: MILS Technical Primer (2008), http://www.ois.com/Products/MILS-Technical-Primer.html

[University of Paderborn, 2009] University of Paderborn, ORCOS (2009), https://orcos.cs.uni-paderborn.de/orcos/

[VirtualLogix, 2009] VirtualLogix, VirtualLogix - Real-time Virtualization for Connected Devices: Products - VLX for Embedded Systems (2009), http://www.virtuallogix.com/

[VmWare, 2009] VmWare, TRANGO Virtual Prozessors: Scalable security for embedded devices (2009), http://www.trango-vp.com/