XtratuM for PowerPC

Zhou Rui[1,2], Wang Baojun[1], Arthur Siro[1], Nicholas McGuire[1], Zhou Qingguo[1]

1. Distributed and Embedded System Lab,
School of Information Science and Engineering,
Lanzhou University, P. R. China

2. School of Mathematics and Statistics,
Lanzhou University, P. R. China

Abstract

Xtratum is a nanokernel designed for providing domain support to execute concurrently several operating systems on a single computer. These domains are in the same hardware but running temporally and spatially isolated. As a very thin virtualization layer virtualizing the essential resources (interrupt, timer, memory, and CPU), XtratuM is suitable for embedded real-time systems, and at least one of those domains should provide real-time capabilities, which can be implemented by porting RTOSes onto XtratuM. The latest released XtratuM 1.0 supporting Linux kernel 2.6.17.4 is implemented on x86 architecture. In this paper, we mainly focus on our work of porting XtratuM to PowerPC. We started XtratuM for PowerPC since March 2006 and published some project fundamentals at the 8th Real-Time Linux Workshop in October 2006. For this time, we will present our implementation of XtratuM for PowerPC, especially the key components including timer, interrupt and memory management for multi-domain support based on PowerPC. PowerPC has a wide range of family members from low-end 32bit like 4xx series up to high-end multi-threaded 64bit like CELL, which can satisfy different powerful and stable performance requirements of various types of embedded and real-time operating systems. XtratuM for PowerPC will help to expand the applicable area of this free-software nanokernel, and also promote embedded and real-time applicable abilities based on PowerPC.

1 Introduction

It is difficult to build a nanokernel from scratch, which involves implementing boot code, new drivers, etc. ADEOS (Adaptive Domain Environment for Operating Systems) [1] project has introduced the concept of domain management to developers, which provides an extensible and adaptive environment used to enable the sharing of hardware resources among multiple operating systems or among multiple instances of the same operating system. This approach has shown that it can be greatly simplified by building the nanokernel around the infrastructure of other existing OS, which is root domain in ADEOS. Based on the concept of ADEOS, XtratuM is originally developed by Universidad Politecnica de Valencia in the framework of the OCERA (Open Components for Embedded Real-time Applications) project, and developed as a thin but important layer able to communicate directly with hardware, and to realize a minimum implementation of ADEOS [2]. As the result of various new developments in hardware, there is an 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 protected from other applications in the system. XtratuM provides a flexible environment for sharing hardware resources among multiple real-time operating systems. In this case, XtratuM enables multiple kernel components called domains or guest operating systems to run in separate address spaces (temporal and spatial partitioning) simultaneously on the same hardware. These domains are executed in a protected memory space independent of each other, but as paravirtualized OS, they all are able communicate with the XtratuM nanokernel directly. The advantages of this approach consist in dividing complex real-time ap-

plications in different parts according to their timing criticality and executing each part on the most suitable operating system. At the same time it allows to build composable systems as the individual domains do not influence each other (FIGURE 1)

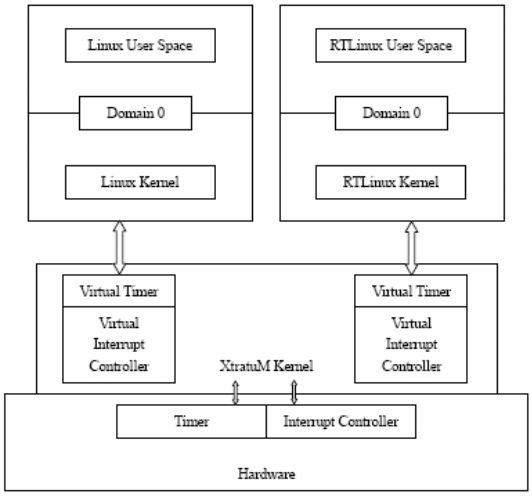

FIGURE 1: *XtratuM Architecture*

Interrupts and timers are the two key components directly affecting the real-time capabilities of an RTOS, if a low priority component like a general purpose OS could disable interrupts at the hardware level no guarantees could be given for a concurrently running RTOS. Thus in XtratuM only the nanokernel has control of the interrupt and timer hardware. XtratuM is mainly focusing on virtual timer, virtual interrupt as well as virtual memory management supporting for high-level domains. It also provides a simple and convenient API to access interrupt mechanisms and the timer devices. The small and simple API not only can be used directly to implement an application, but also has been designed to provide support for multiple domains. [3]
Our work to implement XtratuM for PowerPC has started in March 2006. We have published some project fundamentals at the 8th Real-Time Linux Workshop in October 2006. It is natural to port such a useful nano-kernel to other architectures besides x86 as especially for industrial application robust low-power platforms are mandatory and the PowerPC is one of these platforms. Nowadays, the development of embedded applications has come to a new level with the emergence of various highspeed processors and cheap on-chip memory. As a highly scalable processor family, the series of PowerPC are a typical representative of this development and chosen as our target architecture here. PowerPC has a wide range of family members from low-end 32bit like 405/440 series up to high-end multi-core 64bit like CELL BE, which can satisfy a wide range of different isolation and performance requirements for various types of embedded and real-time operating systems. XtratuM for PowerPC will help to expand the applicable area of this free-software nanokernel, and also promote embedded and real-time applicable abilities based on PowerPC. [4]

2 Implementation

The latest XtratuM 1.0 for x86 is based on Linux kernel 2.6, exactly 2.6.17.4. In this section, we will describe our current implementation of XtratuM for PowerPC in detail, especially virtual timer, virtual interrupt and virtual memory management. Our porting work is also derived from preliminary work reported at the 8th Real-Time Linux Workshop in October 2006 and being conducted in the broader context of DSLabs work on XtratuM extensions. For compatibility with the latest XtratuM 1.0 for x86, we also use Linux kernel 2.6 in our work, which is 2.6.19.2 for PowerPC provided by ELDK 4.1. Both AMCC PowerPC 405EP and 440EP boards are used as our target platform. Due to the API compatibility within the PowerPC family of processors, extending to other PowerPC CPUs should be fairly simple. During our work, it became obvious that we must do a lot about low level functions related to the hardware to make XtratuM/PPC work on PPC in a functionally compatible way with X86. For the high level functions, we tried to change them as little as possible from there current X86 centric implementation, but in certain cases some updating/modifications were necessary due to the change of the hardware architecture and the quite fundamental differences in the way X86 and PowerPC interact with the low level hardware - most notably the memory subsystem.

2.1 Timer

The current x86 implementation of XtratuM can support two types of timer: PIT (Programmable Interval Timer) and TSC (Time Stamp Counter). For time coordination of different domains, one way is that they can be sorted by virtual timer as a domain heap. For each domain, the virtual timer can be created based on the hardware timer got by timer handler. Also, timer interrupt generated by PIT will be taken over by XtratuM, which results in less latency than if it is handled by the individual domains themselves due to the excellent code locality of this extremely small nano-kernel and thus reduced cache/TLB sensitivity. XtratuM provides at least one virtual timer, and the exact number of

timers implemented by it depends on the available number of hardware timers. XtratuM implements a virtual timer for per domain and two system calls, get_time_sys and set_timer_sys, to interact with the virtual timer services. The domains should provide corresponding drivers to interact with these virtual facilities behaving like the regular drivers to the respective OS but invoking the XtratuM primitives instead of interacting with real hardware. The virtual timer implements the multiplexing of the hardware timer between the existing domains. To work with these virtual timers, XtratuM also provides a high-level API to deal with them.
As PowerPC is different from x86 in its hardware architecture, we had to take two different timer facilities - TB (Time Base) and PIT (Programmable Interval Timer) - into consideration. Basically, there is still similarity between these timers of PowerPC and x86 [5]. Both TSC and TB are counters returning a single-step increasing time stamp value. And they do not generate interrupts when they wrap, leaving detection to the driver/OS. For most applications, this kind of timer is set once at system startup and only read thereafter.

The PIT, available on both PowerPC and x86, is functionally a kind of timer than the TSC/TB, it provides timed events by interrupting periodically just like a system heart beat. The difference between the PIT on x86 and PPC is that the PIT on PowerPC causes an internal interrupt while PIT of x86 causes an external interrupt as would any other peripheral interrupt.[5][6]
For the low level timer related functionality, XtratuM should be able to initialize them, read them or get the current counter values of them, set them and shut them down. These functionalities must be implement by us in a PowerPC compatible way instead of x86 (as in the current XtratuM). Take as example the difference of the functionally equivalent TB and TSC as example here. Basically, the timer initialization is mainly to get the cpu frequency (calibration). For x86, there is a series of complex operations on Intel 8253/8254 Counter/Timer Chip (CTC) to get the value. However, for PowerPC, what XtratuM needs to do is using the variable "tb_ticks_per_jiffy" and macro "HZ" defined in Linux kernel source, just like:

```
static inline unsigned long long get_cpu_hz(void)
{
    return (unsigned long long)HZ * tb_ticks_per_jiffy;
}
```

To get the current value of TSC, XtratuM uses read_TSC(), which runs x86 assembly instruction "rdtsc" directly. While for TB, according to PowerPC mechanism, XtratuM should get values of Time Base Lower Register and Time Base Upper Register separately with PowerPC assembly instruction "mftb", and then concatenate them to return a 64-bit value, such as:

```
static hwtime_t read_tb (void)
{
    unsigned long lo, hi, hi2;
    unsigned long long tb;
    do
    {
    hi = get_tbu(); /* get_tbu() is defined to use "mftb" to get Time Base Upper Register value. */
    lo = get_tbl(); /* get_tbl() is defined to use "mftb" to get Time Base Lower Register value. */
    hi2 = get_tbu();
    }
    while (hi2 != hi);
    tb = ((unsigned long long) hi << 32) | lo; /* Get the 64-bit TB value. */
    return tb;
}
```

Based on above examples, when we try to implement the same low level functions on XtratuM for PowerPC, we can see the difference of the solutions compared with x86. While for high level functions, we have tried to change the mechanism as little as possible, but updating of some x86 specific, and thus non-portable code, was indispensable. For virtual timer, in x86, the timer interrupt is one of the common peripheral interrupts and it can be handled in the way as other external interrupts, so there is a general method to process them in XtratuM for x86. Once the timer interrupt happens, the interrupt handler will be called automatically via the IDT (interrupt descriptor table) without OS level intervention. However, as an internal interrupt on PowerPC, if the timer interrupt occurs, XtratuM should provide a specific method to handle it, e.g. when decremented overflows, time interrupt is triggered and XtratuM explicitly calls the interrupt handler such as timer_interrupt() to process it.

2.2 Interrupt

XtratuM manages all hardware interrupts available on the actual hardware architecture being used. Once XtratuM interrupt virtualization is activated, all interrupts of any domains will pass through it, thus none of the domains ever get direct access to the interrupt hardware. In the current XtratuM x86 implementation, these interrupts are controlled by XtratuM by intercepting all interrupts via the IDE, where all entries point to XtratuM only. XtratuM

is placed between the hardware and the domains requiring virtualization of the interrupts, thus preventing the domains from directly accessing interrupt related functions.
In PowerPC, entry of external interrupt is set at hardcoded physical address 0x500 [7] functionally similar to the IDT on x86. XtratuM will intercept external interrupt at this point and use our own handler "__xm_do_IRQ" instead of the original handler "do_IRQ", such as:

#ifdef CONFIG_XTRATUM

EXCEPTION(0x0500, HardwareInterrupt, __xm_do_IRQ, EXC_XFER_XTRATUM)
#else

EXCEPTION(0x0500, HardwareInterrupt, do_IRQ, EXC_XFER_LITE)
#endif

Once an interrupt is intercepted by XtratuM in PowerPC, __xm_do_IRQ() will call __do_IRQ() of XM_root_func, which can be initialized with xm_irq_handler() if the interrupt is for this domain and thus provides the same functionality as was present in Linux befor XtratuM was loaded. In xm_irq_handler(), it will get the IRQ number and the pending events, schedule the events compatible with the domains, and then begin the processing of virtual interrupt for high level domains (FIGURE 2) thus this simply provides a prioritization of hardware interrupts in software.
High level domains can generate various types of interrupts, even the same kind as is in use with a low-priority domain. These interrupt requests are collected and handled by XtratuM, so from the view of these domains they appear as hardware interrupts, though these interrupts are processed virtually by XtratuM instead of real hardware. We can describe the procedure like this: if one virtual interrupt from one domain is delivered, XtratuM will disable other virtual interrupts from the same domain, and then enable the real hardware interrupt, execute the event handler, disable the real hardware interrupt, and finally enable virtual interrupt for the domain (FIGURE 2). For low level operations such as enable and disable real hardware interrupt, x86 uses "sti" and "cli" instructions [8], while PowerPC should set EE (External Interrupt Enable) bit of MSR (Machine State Register) for these functions appropriately.

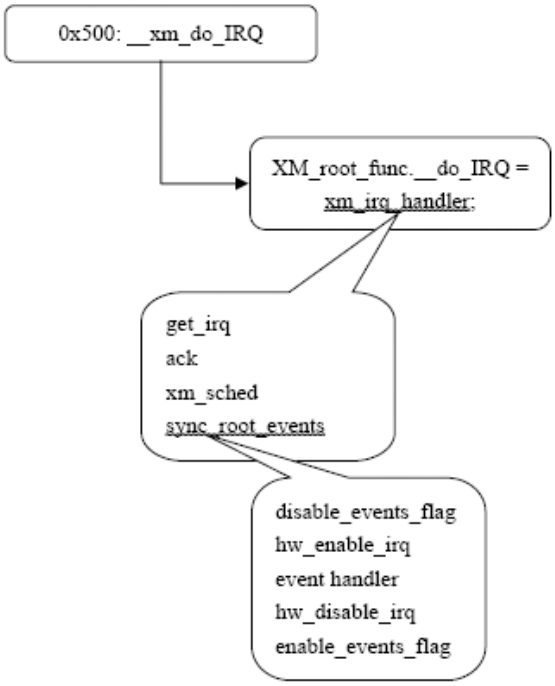

FIGURE 2: *Interrupt Processing*

Furthermore, for x86, XtratuM defines 32 as the maximum number of interrupts, and it seems only 16 of which are used. But for many types of PowerPC processors, e.g. PowerPC 440EP, they always have more than 32 interrupts, so we have defined the number to 64 here, though this clearly is a design flaw that needs fixing as no such hard-coded value will be the right one.

2.3 Memory Management

Though timer and interrupt are the two key issues for XtratuM to ensure real-time capabilities, for guaranteeing that multi domains can run well on XtratuM independently, a correct and efficient method of memory management is certainly indispensable. Notably this lack of address space separation is the key disadvantage of the currently existing hard real-time variants of Linux
For x86 implementation, XtratuM has implemented its own memory management and does paging manually instead of letting linux handle paging or switching automatically. In this way, XtratuM can assign isolated address space in user space for each domain and even prevent the Linux kernel from accessing the domains private memory, so it raises the security of XtratuM and allows providing redundant mechanism for the domains. Once one domain crashes, the effect on other domains can be reduced to a minimum, allowing to mask random hardware failures in

a domain as an other domains can still run well or even take over and go on the interrupted work of the crashed domain.
For PowerPC, basically XtratuM will follow the mechanisms as x86 to guarantee domains' stability and isolation. However, in consideration of the PowerPC family covering from 32-bit to 64-bit processors, XtratuM implements four paging levels, which is compatible with both 32-bit and 64-bit architectures, instead of two paging levels in x86 to provide a more general paging method [9]. Also, as different PowerPC cores may have their own features of memory management, we are trying to cover as much as possible specific issues in our implementation, though clearly the current focus is the 32-bit systems.
For low level functions of PowerPC memory management, i.e. updating page tables or the like, very hardware specific rules must be followed and software changes must be synchronized with the other instructions in execution and with automatic updates that may be made by the hardware (referenced and changed bit of Page Table Entry updates). These locking and atomicity issues are generally very nonportable and constitute the most difficult part of porting such a nano-kernel to a new platform. Updates to the page tables include the following operations [7]:

- Adding a PTE (PageTable Entry) when allocating memory
- Modifying a PTE (i.e. changing access permissions)
- Deleting a PTE

Consider the simplest page table case here to add a PTE. To create a PTE, maintain a consistent state, and ensure that a subsequent reference to the virtual address translated by the new entry will use the correct real address and associated attributes, XtratuM should do the following (in pseudocode) [7]:

PTE[RPN,AC,R,C,WIMG,N,PP] new values
eieio / order 1st update before 2nd */*
PTE[AVPN,SW,H,V] new values (V = 1)
ptesync / order updates before next page table search and before next data access */*

3 Status and Perspectives

Though it seems XtratuM for PowerPC makes slow progress, in recent months, the key functions of timer, interrupt and memory management have been basically implemented in running systems. We have made it boot successfully and run well on PowerPC 405EP and 440EP developing board, and multi domains can basically be stable on it though we don't see the current code state as stable yet. To make it perform as a real real-time nanokernel running on PowerPC, there still is a long way to go, such as for more efficient memory management, TLB (Translation Lookaside Buffer) is proposed to be addressed later. We have made some necessary module test and unit test in previous work, but for the entire project, a total benchmark has jet to be designed and implemented. And also for this project is closely related to hardware and kernel level, some tools such as BDI2000 and KGDB for finding bugs and debugging are also essential for quality guarantee. Though we feel that progress should have been faster it shows that it simply takes time to get a feeling for a new architecture and that our ways of thinking were too much distorted by our x86 work to quickly adjust to the PowerPC.
Another issue we can consider in the future is that currently, XtratuM for PowerPC has not yet supported IBM iSeries. But there may be some requirements for running redundant RTOS on such a powerful products, so we also should fill in this blank sometime. And also we hope to bring XtratuM to the latest PowerPC, the CELL Broadband Engine Architecture as one exciting architecture, especially for the safety critical domain.
Of course, now we can start to porting RTOS to PowerPC with the help of XtratuM, which can add the PowerPC to existing RTOS and expand performance requirement to broader hardware platforms.

4 Acknowledgments

At first, it is very grateful of Mr. Wang Baojun for his great contribution to XtratuM for PowerPC. His work makes all of us know more about XtratuM, PowerPC, Linux kernel, C programming language, debugging tools. Thanks a lot to Prof. Nicholas Mc Guire, Prof. Li Lian, Dr. Zhou Qingguo, Mr. Arthur Siro and all the other people of DSLab for there discussion on XtratuM and RTOS questions. Their help makes me know more and understand more about free-software and my work go on the right track.
This work was supported by National Natural Science Foundation of China through projects: Research on Computational Chemistry E-SCIENCE and its Applications (Grant no. 90612016) and Research on Job Scheduling in network computing (Grant no. 60473095). And this work is also supported by China National Technology Platform.

References

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

[2] *XtratuM*. http://www.xtratum.org

[3] Miguel Masmano, Ismael Ripoll, Alfons Crespo, Audrey Marchand, 2005, *Framework for Real-time Embedded Systems based on COntRacts: Nanokernels for multidomain support*, UPVLC

[4] Zhou Rui, Bai Shuwei, Nicholas McGuire, Li Lian, 2006, *Porting XtratuM to PowerPC*, RTLWS8

[5] International Business Machines Corporation, 2003, *PowerPC 405EP Embedded Processor User's Manual Preliminary*, International Business Machines Corporation

[6] Intel Corporation, *Intel 64 and IA-32 Architectures Software Developer's Manuals, part I, II and III*, Intel Corporation

[7] International Business Machines Corporation, 2005, *PowerPC Microprocessor Family: The Programming Environment Manual for 64-bit Processors*, International Business Machines Corporation

[8] Intel Corporation, 1987, *Intel 80386 PROGRAMMER'S REFERENCE MANUAL*, Intel Corporation

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