Paravirtualizing Linux in a real-time hypervisor

Ewili 2012, Lorient, France

Vincent Legout and Matthieu Lemerre
CEA LIST
vincent.legout@cea.fr, matthieu.lemerre@cea.fr

7 june 2012

Plan

1. Motivation - Objective
2. Related Work
3. Implementation
   - Linux
   - Boot
   - Hypervisor
   - Userspace
4. Performance
5. Conclusion

Motivation - Objective

- Critical hard real-time system
  - Constraints must be met (could be life critical)
- Underutilized processor
  - Could run tasks with less constraints to fill the processor
  - Tasks with as much possibility as possible
  - Embedded processors are fast enough to use virtualization
  - Thus Linux (popularity, virtualization eased, hardware supported)

Objective

Cohabitation between critical real-time tasks and Linux VMs

Virtualization

- Host, Guest, Hypervisor
- Paravirtualization vs Full-virtualization
  - Full-virtualization : Complete simulation of hardware
  - Paravirtualization : Guest knows it is being virtualized
- Why paravirtualization ?
  - Advantage
    - Simplify the hypervisor
  - Disadvantage
    - Cannot run unmodified systems
- Hypervisor, responsible for :
  - Receiving hypercalls from the guest
  - Delivering interrupts to the guest
  - Managing guest memory

Plan

Related Work

- Real-Time Linux : Xenomai, RTAI
  - Enhance Linux to support real-time
  - No critical hard real-time
  - No virtualization
- Hypervisors : L4 (L4Linux), Mach (MkLinux)
  - Microkernels
  - Run paravirtualized Linux
  - Not dedicated to hard real-time
- Paravirtualize Linux : KVM, Xen, Lguest
  - Interface to ease Linux paravirtualization : paravirt_ops

Anaxagoros Microkernel

Objective

Mix critically hard real-time and non real-time tasks

- Resource security :
  - Resource partitioning
  - e.g. cannot steal processor time
- Traditional security :
  - Microkernel :
    - Small kernel base
    - Services running in user space (e.g. hardware drivers)
  - Exokernel (low level interface : e.g. pagination syscalls)
- Implementation for x86

Plan

1 Motivation - Objective

2 Related Work

3 Implementation
- Linux
- Boot
- Hypervisor
- Userspace

4 Performance

5 Conclusion

Linux

- Linux 2.6.36 (With no additional patch)
  - Later version may work but untested
- Minimal configuration
  - make allnoconfig
  - EMBEDDED, CONFIG_BLK_DEV_INITRD
- New configuration option : ANAXAGOROS_GUEST
- Implementation similar to Lguest
- Implementation relies on paravirt_ops
  - Set of functions pointers
  - Used by lguest, KVM, Xen, ...
  - Override privileged functions

How a paravirtualized Linux boots ?

1. Create a new Anaxagoros task
2. Load vmlinux into the address space (and RAM disk)
3. Create a *boot_params* structure (boot configuration)
   - *hdr.hardware_subarch* field
4. Switch to Linux task
5. Jump to Linux entry point
6. Linux goes to the initialization function *anaxagoros_entry*
7. Linux sets up the *paravirt_ops* functions
8. Linux performs the initialization hypercall (initialize pages)
9. Linux boots

Hypervisor

- Hypercalls
  - Software interrupt
  - New entry in Anaxagoros's interrupt vector table
  - 13 hypercalls
  - Up to four arguments for each hypercall
- Interruptions
  - Deliver interrupts to guests
  - Guests cannot use their own interrupt vector table
  - Virtual interrupt vector table for each guest (updated via hypercalls)
- Segmentation (Limited usage)
  - Keep Linux kernelspace and userspace in Anaxagoros's userspace
- Pagination
  - Two-level paging (No PAE)
  - Use hypercalls to update page tables
  - Low performance because setting a page table entry requires a hypercall

Userspace

- Load ramdisk in memory before booting Linux
- Busybox (built statically)
- Other applications (with klibc, glibc, . . .)
- Services from Anaxagoros (e.g. VGA, Keyboard)
- Network
  - Use Linux driver to access the network card
- Limitations
  - Clock : Do not deliver all timer interrupts.
    - For example when the guest is not active
  - I/O : No I/O virtualization
    - A network card cannot be used by two guests simultaneously (but a network card can be dedicated to one guest)

Plan

Performance - Hypervisor

- Compare native and virtualized Linux
  - Bochs simulator (first two lines)
  - Intel Xeon (3.06 GHz), 4 Go of DDR-SDRAM (last two lines)
- 4 lmbench tests & tcc self-compilation

	syscall	rand	ctx	fork	pipe	tcc
Native	71	138	0.74k	91k	4.51k	231M
Virt.	308	141	1.59k	1969k	6.25k	284M
Native	490	167	13.83k	774k	33k	458M
Virt.	2640	188	17.1k	2589k	38.9k	563M

- Good : rand, ctx, pipe, tcc
- Bad : syscall, fork

Performance - Real-Time

- Questions :
  - Does the virtualized Linux affect the real-time performance of the real-time task ?
- Experiment :
  - Run a real-time task for a fixed amount of time periodically.
  - This task is incrementing a counter (counter reset each time the task begins a new execution).
  - The counter accounts for the execution time.
  - Tasks with various workloads are run next to the real-time task.
- Result :
  - The counter value at the end of each execution should always be identical, no matter how active the other tasks are.

Performance - Real-Time

- Deviation between nop loop and Linux with Intel Xeon (200ms) :

average deviation	maximum deviation
5%	18%

- Linux effects on the real-time task are not negligible
- With Bochs (no cache), Linux has few effects on the execution of the real-time task :
  - Counter mean value : 3.3×10^6
  - Difference between minimum and maximum values : 74
- Software approach validated but need to deal with hardware

Conclusion

- Initial problem : Run side by side real-time tasks and Linux
- What has been done ?
  - Paravirtualize Linux on Anaxagoros
  - Tests to ensure that Linux can run next to real-time tasks
- Future works :
  - Virtualization performance (batch hypercalls)
  - Improve integration with Anaxagoros
  - Based on this experiment, improve Anaxagoros :
    - Support more hardware
    - Multicore systems

leti

LABORATOIRE D'ÉLECTRONIQUE
ET DE TECHNOLOGIES
DE L'INFORMATION

list

LABORATOIRE D'INTÉGRATION
DES SYSTÈMES
ET DES TECHNOLOGIES

Thanks !