Linux Kernel Hacking Free Course 2003

UNIVERSITA' DEGLI STUDI DI ROMA TorVergata Università

D. P. Bovet, M. Cesati, R. Gioiosa

Asymmetric Kernels for MultiProcessor Systems

SMP systems

Multiprocessor systems (MP) are a natural extension of Uniprocessor systems (UP).

All processors in the system:

- Execute instructions concurrently
- Share the physical memory and access it through a common physical address space
- Can handle interrupts coming from I/O devices

More on SMP systems (1)

According to Intel's definition, two main properties:

I. *Memory symmetric*: Memory is symmetric when all processors share the same physical memory and access that memory using a common address space

II. *I/O symmetric*: I/O is symmetric when any processor can access the whole I/O sub-system (including I/O ports and interrupt controller) and any processor can receive interrupts from any source.

More on SMP systems (2)

Looking into more details, we might argue that SMP systems do not exist because computers feature other types of processors besides the CPUs

DMA controllers are quite different from the CPUs, although they must be scheduled by the kernel like the CPUs

Needless to say, the definition of SMP system focuses on the interrelationships between CPUs and does not consider other types of processors

Kernels for SMP systems

Two types of kernels for SMP systems:

- Symmetric kernels (SMP kernels)
- Asymmetric kernels (ASMP kernels)

SMP kernels

Most widespread kernels for SMP systems are symmetric. There are many reasons:

- Simplicity: all CPUs are treated in the same way
- Scalability: the same kernel works for any number of CPUs
- Performance: if you don't know anything about the workload, the best way to exploit the CPUs is to share the load evenly among them

The Linux SMP kernel (1)

Linux SMP handles processes in a symmetric way:

a) Processes don't care about the processor on which they are in progress.

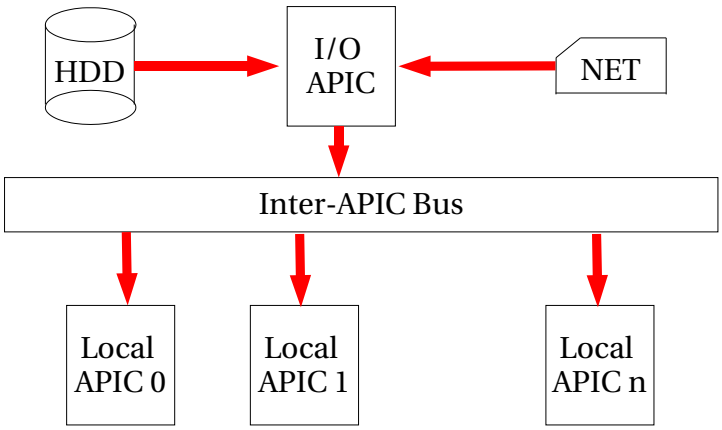
b) There is only one copy of the kernel code in memory.

c) There is only one copy of the code for each process in the system.

d) When the scheduler selects a new process to run on a CPU, it will give a small preference to the processes that were running on that CPU during their last time quantum (try to keep the cache hot)

The Linux SMP kernel (2)

Interrupts are handled symmetrically:



Linux SMP advantages

The current Linux SMP kernel:

- Is simple because it is a natural evolution of the uniprocessor kernel (however, a lot of race conditions that cannot occur on UP systems must be taken care of)
- Is scalable
- Saves system resources (like memory)

Linux SMP disadvantages

- Process migration invalidates hardware cache

Linux 2.4 SMP is not suitable for Real-Time applications because:

- If a RT process becomes runnable while the system is in kernel mode it cannot be resumed (*Non preemptible Kernel*)
- RT process may use resources that are also used by conventional processes (*Priority inversion*)
- RT process may require a kernel service that is granted by a lower-priority process (*Hidden scheduling*)

Implementing a Linux ASMP kernel

We decided to implement an ASMP kernel for Linux to test the following ideas:

- Using a CPU to perform specific functions of the kernel might yield a better use of the CPU's cache and thus better performances with respect to a SMP kernel
- Real-time applications could be run as processes running on a dedicated CPU (see later)
- Worloads like those of web servers might be handled more efficiently by associating some kinds of interrupts with specific CPUs

ASMP (1)

- We must select some CPUs *(shielded CPUs)* to execute only *asymmetric functions*
- Only Real-Time processes can run on the shielded CPUs
- RT applications must not be blocked by interrupts that are not related to the applications
- Hardware cache of a shielded CPU must be always preserved, even when the corresponding RT application is sleeping
- Kernel should assign a shielded CPU to its RT application as soon as it wakes up

ASMP (2)

In an Asymmetric Multiprocessor System several shielded CPUs (*S-CPU*) execute specific kernel functions while other CPUs (*O-CPU*) execute the remaining functions.

A comparison with Linux SMP:

- Memory management is the same (no need to change it)

- I/O management, process management, and scheduling are different

Process management (1)

Process management must change to satisfy several conditions:

1) Only RT application can run on the Shielded CPU

2) If the RT application sleeps, only the Swapper process can run on the *S-CPU*

3) A RT application can run forever on its CPU until it relinquishes it or until an interrupt is routed to the Shielded CPU

4) Other CPUs execute other processes with the usual Linux policies (time-sharing, round-robin priority,...)

Data structures for Process Management

The Linux 2.4 process descriptor contains several fields related to SMP systems:

1. `cpus_allowed`: bitmask indicating which CPUs can execute the process.

2. `cpus_runnable`: bitmask indicating the CPU that is now executing the process (`0xffffffff` if the process is not running)

The scheduler checks these fields when looking for a new process to run

Process management (2)

We can set the previous fields as `ksoftirqd_n` does:

1. For RT applications we set `cpus_allowed` to run process only on *S-CPU*s

2. For other processes, we set `cpus_allowed` to inhibit the process to run on a *S-CPU*

e.g.:

```
cpus_allowed = (1<<acpu);
```
for RT-application

```
cpus_allowed &= ~(1<<acpu);
```
for other application

Scheduler (1)

In order to improve the performance of an ASMP kernel, we need to change the scheduler as follows:

1. Two distinct runqueue lists: one for the *S-CPU* and one for the *O-CPU*. We do this by using `cpus_allowed` and `can_schedule()` macro

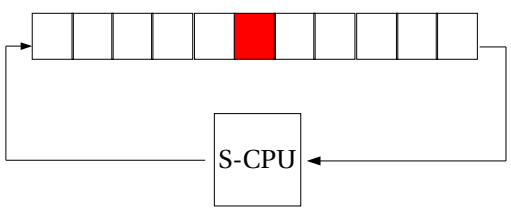
2. If the RT application is runnable, it must be selected as the next process to assign to a *S-CPU*

3. If the RT application is sleeping, the Swapper must be selected as next process, so that the hardware cache is not invalidated

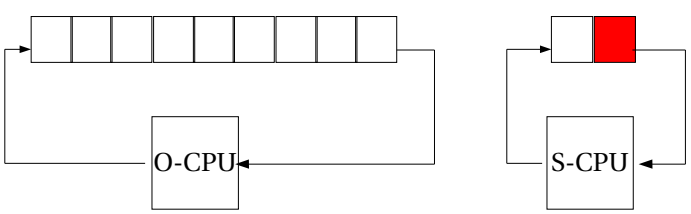
4. The RT application is never be blocked by other processes so it can run until it relinquishes the CPU!

Scheduler (2)

SMP Scheduler



ASMP Scheduler



Scheduler (3)

The *O-CPU* Scheduler algorithm doesn't change:

1. The scheduler scans the processes that are in the *O-CPU* runqueue and selects the best

2. The scheduler running on a *O-CPU* never scans the *S-CPU* runqueue

3. If the time quantum of a process has expired, the process must be replaced by some other one

4. A process running on an *O-CPU* can be interrupted by an interrupt or an exception, and then possibly replaced by a higher priority process (preemption)

Interrupts management

• If a *S-CPU* receives an interrupt signal from an I/O device, the RT applications running on it would be interrupted by the kernel to perform system functions, that's not good...

• In our implementation, the RT application can be interrupted only by interrupts coming from the corresponding RT device

• Interrupt coming from non-RT devices must be delivered to the *O-CPU*s

APIC (1)

How can we distribute non-RT interrupts to *O-CPU*s and RT interrupts to *S-CPU*s?

Intel chipsets use the *Advanced Interrupt Controller (APIC)* to delivery interrupts to CPUs

There are two types of APIC:

1. *I/O APIC* included in the motherboard chipset: receives interrupts from I/O devices and deliver them to the Local APIC of the CPUs.

2. *Local APIC* included into the microprocessor chip: receives interrupts from the I/O APIC and delivers them to the processor core.

APIC (2)

APICs perform many operations:

1. Receive interrupt signals from I/O devices locally connected to the I/O APIC and deliver them to the processor core.

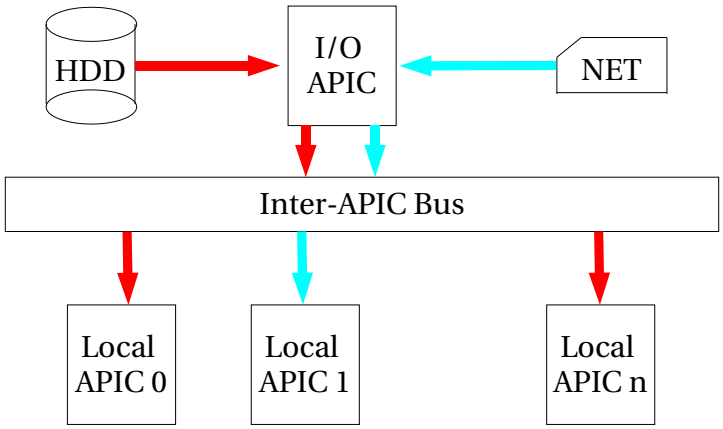
2. In SMP systems, send and receive IPIs (Inter-Processor Interrupts) to and from other processors.

Usually the I/O APIC sends interrupts in broadcast mode to all processors.

We have to re-program the I/O APIC in order to deliver only one type of interrupts (those coming from RT devices) to the *S-CPU*s!

APIC (3)

ASMP Interrupt management



How to program the I/O APIC (1)

Linux SMP provides some helper functions and data structures to program the I/O APIC:

1. For every IRQ line there is an IRQ descriptor in the `irq_desc` array

2. Every IRQ descriptor has an handler field that stores the Interrupt Controller descriptor (its content depends on the model of Interrupt Controller present in the system)

How to program the I/O APIC (2)

3. The Interrupt Controller descriptor stores the addresses of functions that can be used to handle the Interrupt Controller (*methods*)

4. We can use the `set_affinity` method to set the pairing between any IRQ line and any CPU

In an SMP system, there is an `smp_affinity` proc file for every IRQ line; this file can be written with a bitmask specifying which processors should handle the corresponding interrupt signals

There is a problem when hardware devices share the same interrupt line...

Preliminary tests (1)

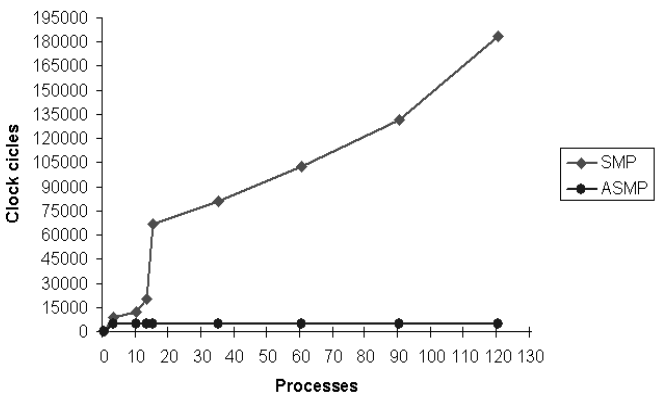
Now the time has come to test our ASMP system!

- Dual processor Intel Pentium III 500Mhz: one CPU is *S-CPU* and the other is *O-CPU*
- Network card on IRQ 12 is our RT device
- There is a RT application that runs on the *S-CPU*

We keep track of the time elapsed between the end of the network interrupt handler and the time the RT application wakes up.

Preliminary tests (2)

We tested the response time of our RT application under different system loads:



Conclusions (1)

Advantages:

- Response time and variance are short and constant
- Hardware cache of *S-CPU* is always hot
- RT application is never preempted by other processes or interrupted by unrelated interrupts
- A RT interrupt handler cannot be interrupted, except by another interrupt handler of the same kind

Conclusions (2)

Disadvantage:

- Computational resources are statically allocated, so the *S-CPU*s do not execute normal system functions (no load balancing)

Conclusions (3)

- Possible scenarios:
  - Real Time application that makes use of a device (e.g.: DVD player...)
  - System with unbalanced load or unbalanced interrupts (e.g.: web server, database engine, ...)
- Future work:
  - Use a 4-processor system to handle network and disk interrupts in asymmetric mode...

Haven't you got an SMP system? Don't worry, you can use UserMode Linux to simulate an SMP system!

References

R. Gioiosa, Master dissertation thesis in Computer Science (Ingegneria Informatica), Univ. of Rome "Tor Vergata" – October 23, 2002, Rome

S. Brosky and S. Rotol, Shielded Processors: Guaranteeing Sub-millisecond Response in Standard Linux, Fourth Real-Time Linux Workshop, December 6-7, 2002, Boston