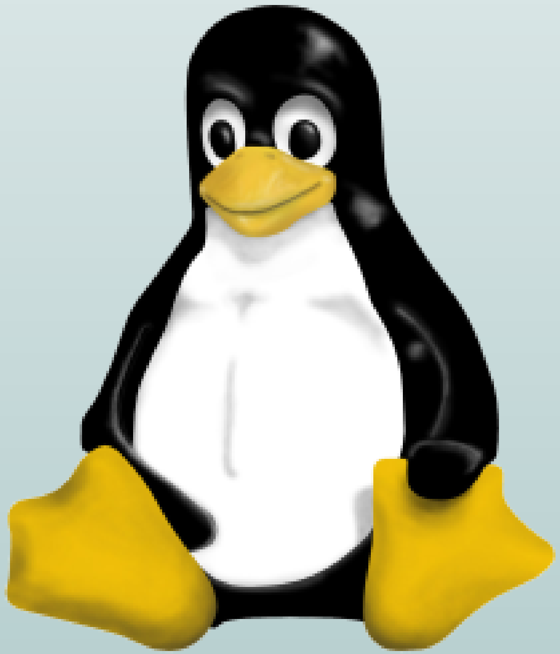
Linux Kernel Hacking Free Course, 3rd edition

R. Gioiosa
University of Rome “Tor Vergata”

Real time systems



March 29, 2006

Index

- Introduction
- The simplest real time system
- Micro-kernel RTOS
- Non-micro kernel RTOS

Real time classification

The term real time can have different meaning, depending on the audience and application. In the computer science literature real time systems are divided in two main categories:

Real time classification

The term real time can have different meaning, depending on the audience and application. In the computer science literature real time systems are divided in two main categories:

Soft real time systems are characterized by their ability to execute a task according to a desired time schedule on the average.

Real time classification

The term real time can have different meaning, depending on the audience and application. In the computer science literature real time systems are divided in two main categories:

Soft real time systems are characterized by their ability to execute a task according to a desired time schedule on the average.

Hard real time systems must always satisfy timing requirements, quality of service, latency constraints, etc.

Real time applications

A typical real time application spends most of its time waiting for external events, but:

Real time applications

A typical real time application spends most of its time waiting for external events, but:

- as soon as the event fires, the system must be ready to resume the real time application

Real time applications

A typical real time application spends most of its time waiting for external events, but:

- as soon as the event fires, the system must be ready to resume the real time application
- the real time application must have all the resources required to complete its task.

Real time applications

A typical real time application spends most of its time waiting for external events, but:

- as soon as the event fires, the system must be ready to resume the real time application
- the real time application must have all the resources required to complete its task.

Other non-critical processes may be running at the same time: a time-sharing system must reach a compromise between real time and non-real time applications.

Real time constraints

Real time constraints depend on the purpose of the system, in general the following goals should be reached:

Real time constraints

Real time constraints depend on the purpose of the system, in general the following goals should be reached:

- Response time from 100 μs to milliseconds

Real time constraints

Real time constraints depend on the purpose of the system, in general the following goals should be reached:

- Response time from 100 μs to milliseconds
- Latency from hundreds of microseconds to seconds

Real time constraints

Real time constraints depend on the purpose of the system, in general the following goals should be reached:

- Response time from 100 μs to milliseconds
- Latency from hundreds of microseconds to seconds
- Determinism always!

Real time constraints

Real time constraints depend on the purpose of the system, in general the following goals should be reached:

- Response time from 100 μs to milliseconds
- Latency from hundreds of microseconds to seconds
- Determinism always!

Having an hard real time system is mainly a matter of determinism!

Real time systems

A real time system is composed by:

Real time systems

A real time system is composed by:

1. deterministic hardware

Real time systems

A real time system is composed by:

1. deterministic hardware
2. a real time OS

Real time systems

A real time system is composed by:

1. deterministic hardware

2. a real time OS

Generally speaking hard real time constraints can be met with both dedicated CPU (*Digital Signal Processors*, DSP) and OS only.

Real time systems

A real time system is composed by:

1. deterministic hardware

2. a real time OS

Generally speaking hard real time constraints can be met with both dedicated CPU (*Digital Signal Processors*, DSP) and OS only.

RTOS must handle problems such as process scheduling, resources allocation, priority inversion, etc.

COTS hardware

Commercial Off-The-Shelf (COTS) processors are used for their lower cost and their widely availability.

COTS hardware

Commercial Off-The-Shelf (COTS) processors are used for their lower cost and their widely availability. Their performances tightly depend on:

- virtual memory and its related MMU
- caches (L1, L2, L3, TLB, etc)
- pipeline and speculative execution
- branch prediction
- intelligent bus arbitration (PCI, etc)

COTS hardware

Commercial Off-The-Shelf (COTS) processors are used for their lower cost and their widely availability. Their performances tightly depend on:

- virtual memory and its related MMU
- caches (L1, L2, L3, TLB, etc)
- pipeline and speculative execution
- branch prediction
- intelligent bus arbitration (PCI, etc)

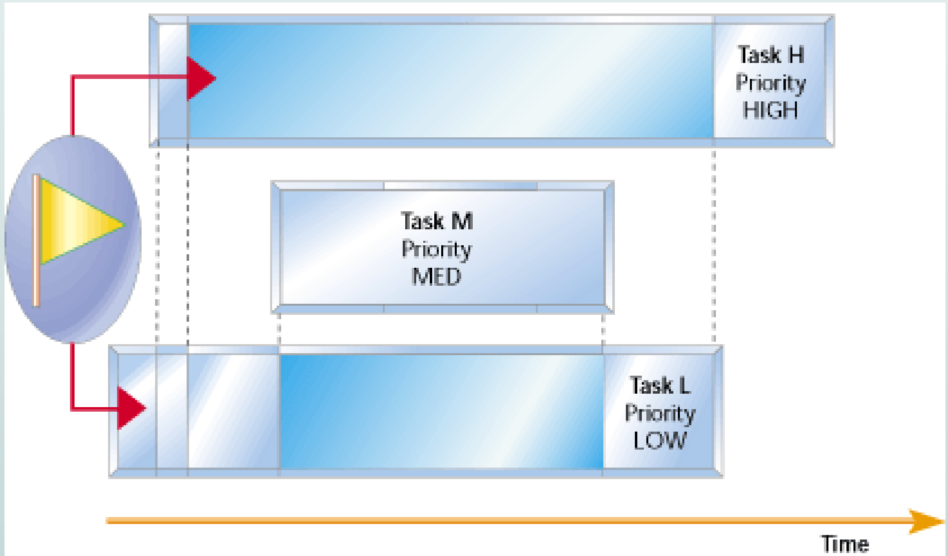
A worse case analysis is required to understand if both hardware and software are *sufficiently real time*.

Priority inversion

If a low priority task L has got some resources required by a high priority task H, the critical task might be delayed (priority inversion).

Priority inversion

If a low priority task L has got some resources required by a high priority task H, the critical task might be delayed (priority inversion).



Priority ceiling and priority inheritance avoid this problem. (Do you remember the Mars Pathfinder failure in 1997?)

A simple real time system

A simple form of a real time system consists of a dedicated system where the only running process is the real time application.

A simple real time system

A simple form of a real time system consists of a dedicated system where the only running process is the real time application.

A minimal OS (DOS-like), which is able to run only the real time application, may or may be not present (in the last case the application contains the code necessary to initialize and to handle the hardware).

A simple real time system

A simple form of a real time system consists of a dedicated system where the only running process is the real time application.

A minimal OS (DOS-like), which is able to run only the real time application, may or may be not present (in the last case the application contains the code necessary to initialize and to handle the hardware).

When the real time application is not running (waiting for some event) the whole system is idle.

A simple real time system

A simple form of a real time system consists of a dedicated system where the only running process is the real time application.

A minimal OS (DOS-like), which is able to run only the real time application, may or may be not present (in the last case the application contains the code necessary to initialize and to handle the hardware).

When the real time application is not running (waiting for some event) the whole system is idle.

Even if this solution seems to be reasonable, in most case the real time application must live with other non-critical applications: somehow an OS support for a multi-tasking environment is required.

RTOS

When OS support is required, there are several approaches:

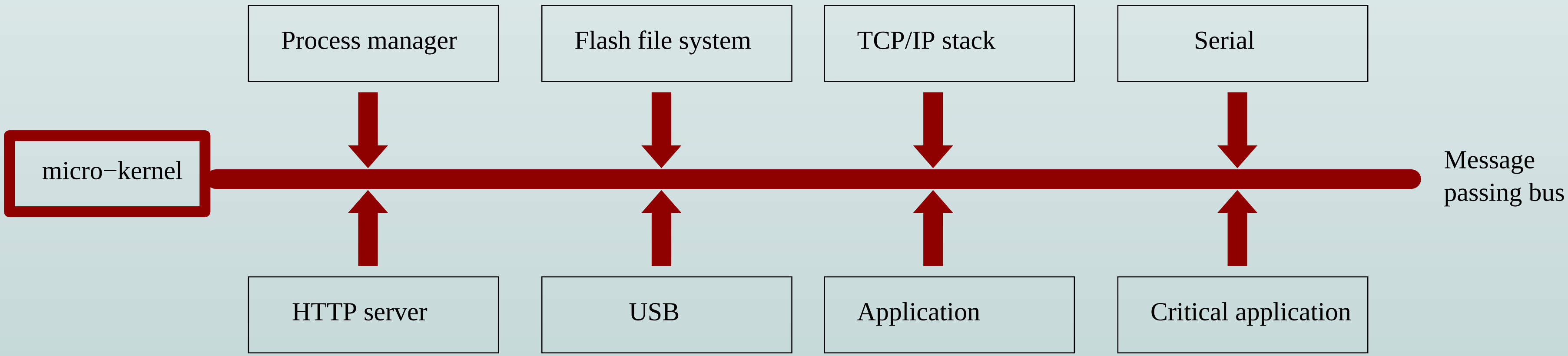
- micro-kernel (VxWorks, QNX, LynxOS, etc)
- non-micro kernel (RTAI, RTLinux, ASMP Linux, etc.)

Micro-kernel architecture

A common software architecture for RTOS consists of a micro-kernel that executes, coordinates and schedules the running processes (both applications and drivers).

Micro-kernel architecture

A common software architecture for RTOS consists of a micro-kernel that executes, coordinates and schedules the running processes (both applications and drivers).



The processes communicate with each other and the hardware by means of a virtual message passing bus.

Micro-kernel characteristics

The main characteristics of micro-kernel architecture are:

Micro-kernel characteristics

The main characteristics of micro-kernel architecture are:

- The most fundamental primitives (signals, timers, scheduling, etc) are handled by the kernel itself.

Micro-kernel characteristics

The main characteristics of micro-kernel architecture are:

- The most fundamental primitives (signals, timers, scheduling, etc) are handled by the kernel itself.
- Drivers, file systems, protocol stacks, user applications, etc. run as separate processes.

Micro-kernel characteristics

The main characteristics of micro-kernel architecture are:

- The most fundamental primitives (signals, timers, scheduling, etc) are handled by the kernel itself.
- Drivers, file systems, protocol stacks, user applications, etc. run as separate processes.
- The processes are scheduled according with their priority (e.g. 0-255) and the kernel has the highest priority.

Micro-kernel characteristics

The main characteristics of micro-kernel architecture are:

- The most fundamental primitives (signals, timers, scheduling, etc) are handled by the kernel itself.
- Drivers, file systems, protocol stacks, user applications, etc. run as separate processes.
- The processes are scheduled according with their priority (e.g. 0-255) and the kernel has the highest priority.

Sometimes the MMU is disabled in order to reduce the non-determinism introduced by the TLB, virtual memory, etc. (VxWorks).

Security

Often real time applications are critical for safety and security. The RTOS must forbid user processes to access others data.

Security

Often real time applications are critical for safety and security. The RTOS must forbid user processes to access others data.

If the MMU is disabled or not present, all the processes and the kernel share the same address space. The RTOS must provide a mechanism to avoid processes to access others private data.

Security

Often real time applications are critical for safety and security. The RTOS must forbid user processes to access others data.

If the MMU is disabled or not present, all the processes and the kernel share the same address space. The RTOS must provide a mechanism to avoid processes to access others private data.

If the MMU is enabled, the hardware already protects data accesses (QNX).

Security

Often real time applications are critical for safety and security. The RTOS must forbid user processes to access others data.

If the MMU is disabled or not present, all the processes and the kernel share the same address space. The RTOS must provide a mechanism to avoid processes to access others private data.

If the MMU is enabled, the hardware already protects data accesses (QNX).

However others resources (hardware caches) are shared and must be invalidated when switching from a process to another.

Real time in Linux (1)

The official Linux kernel provides some features useful for soft real time systems:

Real time in Linux (1)

The official Linux kernel provides some features useful for soft real time systems:

- The scheduler is $O(1)$

Real time in Linux (1)

The official Linux kernel provides some features useful for soft real time systems:

- The scheduler is $O(1)$
- A process can have a `SCHED_RR` of `SCHED_FIFO` priority

Real time in Linux (1)

The official Linux kernel provides some features useful for soft real time systems:

- The scheduler is $O(1)$
- A process can have a `SCHED_RR` of `SCHED_FIFO` priority
- User processes running in user mode can always be preempted by higher priority processes

Real time in Linux (1)

The official Linux kernel provides some features useful for soft real time systems:

- The scheduler is $O(1)$
- A process can have a SCHED_RR of SCHED_FIFO priority
- User processes running in user mode can always be preempted by higher priority processes
- User processes running in kernel mode may also be preempted (kernel preemption)

Real time in Linux (2)

Process preemption either in user or kernel mode is not a big deal;

Real time in Linux (2)

Process preemption either in user or kernel mode is not a big deal; when the kernel is executing functions not related to any process (interrupt handlers, scheduler, etc) it cannot be preempted!

Real time in Linux (2)

Process preemption either in user or kernel mode is not a big deal; when the kernel is executing functions not related to any process (interrupt handlers, scheduler, etc) it cannot be preempted!

The system goes in kernel mode as a result of:

- system calls
- exceptions
- interrupts

Real time in Linux (2)

Process preemption either in user or kernel mode is not a big deal; when the kernel is executing functions not related to any process (interrupt handlers, scheduler, etc) it cannot be preempted!

The system goes in kernel mode as a result of:

- system calls
- exceptions
- interrupts

The time spent in kernel mode cannot be predicted and the kernel itself cannot be controlled, thus Linux kernel cannot be classified as a hard RTOS.

Handling external events

The main problem for Linux is to handle asynchronous events (such as device interrupts).

Many OSes uses different approaches to solve this problem. We'll consider:

- The Real Time Application Interface (RTAI)
- The Asymmetric Multiprocessor Linux (ASMP Linux)

Adeos (1)

RTAI is based on Adaptive Domain Environment for Operating Systems; Adeos allows the user to run different instances of OSes on the same shared hardware.

Adeos (1)

RTAI is based on Adaptive Domain Environment for Operating Systems; Adeos allows the user to run different instances of OSes on the same shared hardware.

Each OS runs into a *Domain*:

- interrupts are sent to all domains according to their *domain priority* and availability.
- a scheduler alternates the execution of different domains.

Adeos (2)

Adeos must have complete control of the hardware so it can provide a virtual abstraction of that hardware to hosted OSes.

Adeos (2)

Adeos must have complete control of the hardware so it can provide a virtual abstraction of that hardware to hosted OSes.

Hosted OSes may or may not recognize Adeos; in the second case Adeos must run transparently to the hosted OS.

Adeos (2)

Adeos must have complete control of the hardware so it can provide a virtual abstraction of that hardware to hosted OSes.

Hosted OSes may or may not recognize Adeos; in the second case Adeos must run transparently to the hosted OS.

Because kernels (such as the Linux kernel) cannot be modified in order to remove all the instructions that interacts with the hardware (`cli`, `sti`, etc), the hosted Linux kernel is executed at *ring level 1*.

Adeos (3)

As soon as the hosted Linux kernel tries to execute a *privileged instruction*, the CPU raises an exception and the domain is interrupted.

Adeos (3)

As soon as the hosted Linux kernel tries to execute a *privileged instruction*, the CPU raises an exception and the domain is interrupted.

Then the Adeos exception handler transforms the Assembly instruction into a logical instruction (`cli` $->$ `stall`).

Adeos (3)

As soon as the hosted Linux kernel tries to execute a *privileged instruction*, the CPU raises an exception and the domain is interrupted.

Then the Adeos exception handler transforms the Assembly instruction into a logical instruction (`cli` $->$ `stall`).

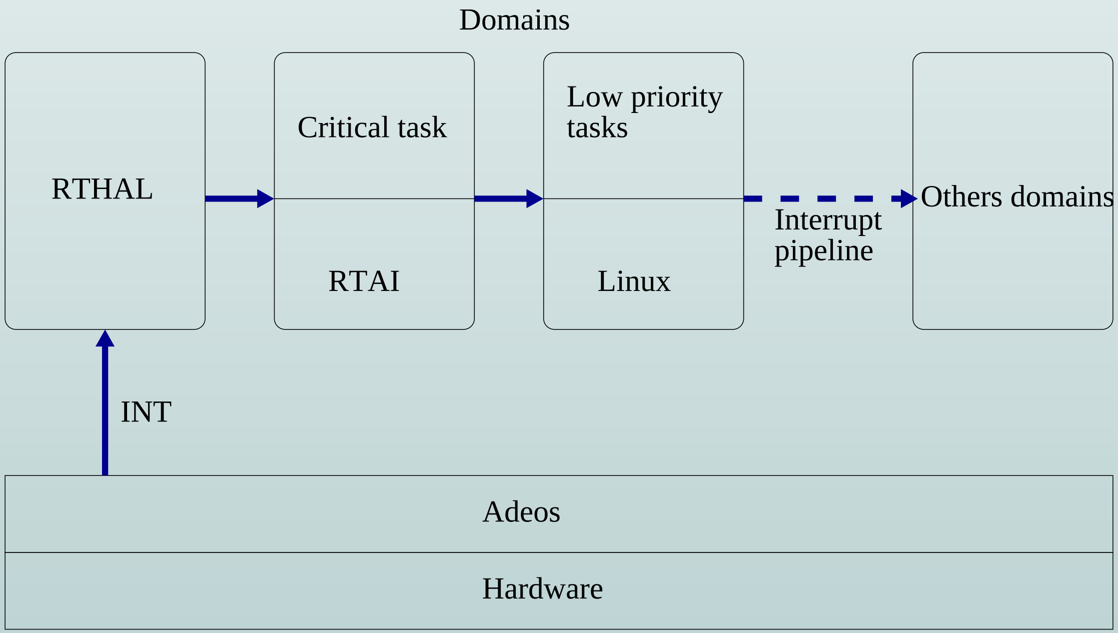
If a domain is able to talk with Adeos, it can stop the interrupt propagation, so no more domains will receive the interrupt.

Real Time Hardware Abstraction Layer

A high priority domain can be implemented to handle the interrupts or to execute high priority tasks: the RTAI uses this schema to implement its RTHL.

Real Time Hardware Abstraction Layer

A high priority domain can be implemented to handle the interrupts or to execute high priority tasks: the RTAI uses this schema to implement its RTHL. Low priority activities are handled by the next domain where Linux is running.



System latency

The system latency is the time spent by both hardware and software to detect real time events and to switch to the process that handles that event.

- interrupt latency
- context switch time

System latency

The system latency is the time spent by both hardware and software to detect real time events and to switch to the process that handles that event.

- interrupt latency
- context switch time

sending a byte along the serial port and getting an answer:

	L. load				H. load			
OS	Avg.	St.dev.	Min	Max	Avg.	St.dev.	Min	Max
Linux 2.6.7	979.83	285.36	568.95	1577.59	1041.25	290.10	568.71	1616.99
RTAI 24.1.13	205.36	3.67	195.58	214.43	205.43	3.68	195.58	219.72
Prop. RTOS	214.72	4.02	203.52	259.29	217.55	11.10	206.14	343.77

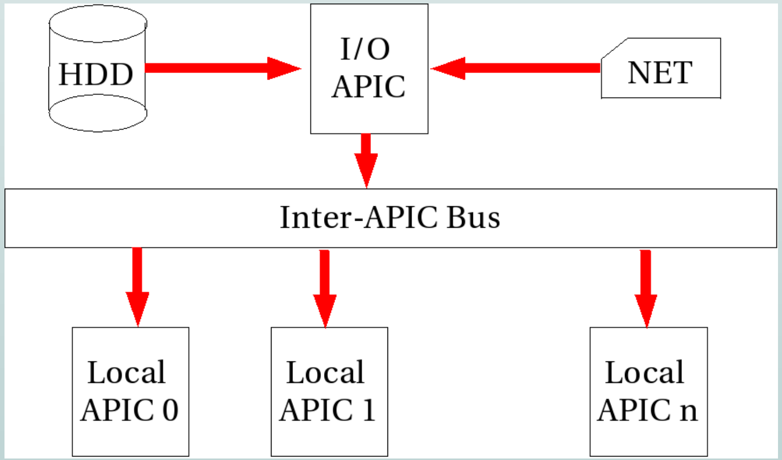
SMP Linux

According to Intel, a MP system is symmetric if it is:

SMP Linux

According to Intel, a MP system is symmetric if it is:

1. Memory symmetric All processors share the same physical memory
2. I/O symmetric Any processors can access the I/O sub-system and handle interrupts



ASMP Linux

In an Asymmetric kernel a sub-set of the hardware resources (CPU, devices, memory, etc) is allocated only for real time applications.

ASMP Linux

In an Asymmetric kernel a sub-set of the hardware resources (CPU, devices, memory, etc) is allocated only for real time applications.

- A selected CPU, (the Asymmetric CPU), only runs real time applications.

ASMP Linux

In an Asymmetric kernel a sub-set of the hardware resources (CPU, devices, memory, etc) is allocated only for real time applications.

- A selected CPU, (the Asymmetric CPU), only runs real time applications.
- Real time applications should not be blocked by interrupts not related to their real time activities.

ASMP Linux

In an Asymmetric kernel a sub-set of the hardware resources (CPU, devices, memory, etc) is allocated only for real time applications.

- A selected CPU, (the Asymmetric CPU), only runs real time applications.
- Real time applications should not be blocked by interrupts not related to their real time activities.
- Hardware caches of the A-CPU must always be preserved, even if real time applications are sleeping.

ASMP Linux

In an Asymmetric kernel a sub-set of the hardware resources (CPU, devices, memory, etc) is allocated only for real time applications.

- A selected CPU, (the Asymmetric CPU), only runs real time applications.
- Real time applications should not be blocked by interrupts not related to their real time activities.
- Hardware caches of the A-CPU must always be preserved, even if real time applications are sleeping.
- All remaining processes and interrupts must be handled by the others CPUs.

ASMP Linux: advantages

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.

ASMP Linux: advantages

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.

ASMP Linux: advantages

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.

The A-CPU never goes in kernel mode unless the real time application explicitly ask for it.

ASMP Linux: advantages

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.

The A-CPU never goes in kernel mode unless the real time application explicitly ask for it.

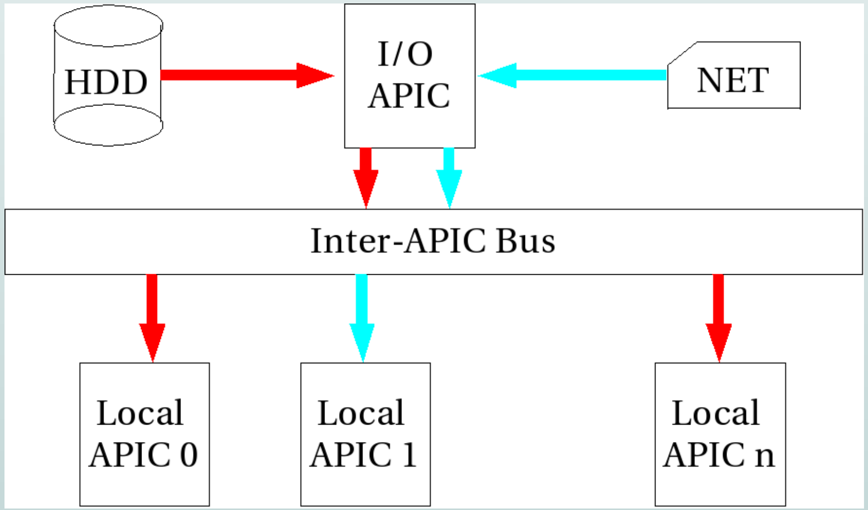
A device can be associated with the A-CPU realizing a smart device.

ASMP Linux: advantages

Interrupts coming from non-real time device (timer, keyboard, etc.) are always delivered to normal CPUs.

ASMP Linux: advantages

Interrupts coming from non-real time device (timer, keyboard, etc.) are always delivered to normal CPUs.



Interrupts coming from the smart device are delivered only to the A-CPU.

ASMP Linux: preliminary tests

The ASMP Linux OS has been evaluated measuring the OS latency

ASMP Linux: preliminary tests

The ASMP Linux OS has been evaluated measuring the OS latency