

**RTAI:  
Embedded Linux  
VS  
Legacy RTOS**

Gabriele Manduchi  
Antonio Barbalace

**Introduction**

Linux VS RTOS

# Linux VS RTOS

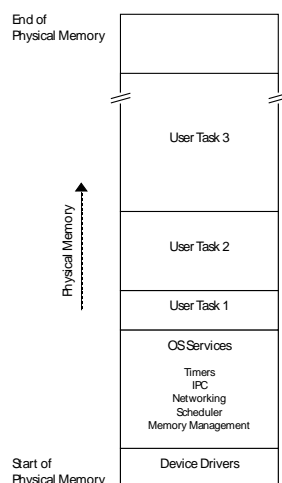
## Linux (or Linux Embedded)

- General purpose OS;
- MMU;
- Kernel call through syscall (software interrupt).

## RTOS like VxWorks, pSOS, Nucleus

- Special Purpose OS;
- Flat Memory model;
- Kernel call through well defined function calls.

# Linux VS RTOS

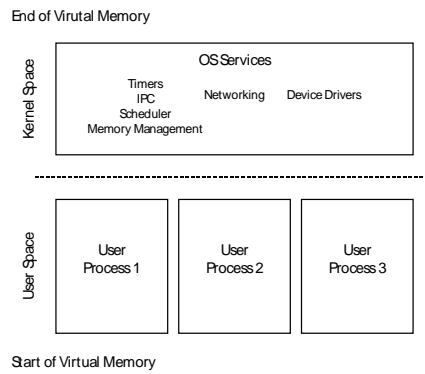


RTOS flat memory model

**MAJOR DRAWBACK:**  
No memory protection

# Linux VS RTOS

Linux memory model



## QUESTIONS:

RT in kernel space?

...maintain legacy approach

RT in user space?

...exploit memory protection

# Linux

From the User Space  
As a soft RTOS

## Linux

Real-Time constraints in:

- Multimedia application;
- VoIP;
- Audio and videos streaming;
- ...

Linux is a soft real-time system.

## Linux

Features of an RTOS:

- Multitasking/multithreading
- Priorities
- Priority inheritance
- Preemption
- Interprocess communication and synchronization
- Dynamic memory allocation

# Linux

## Why Linux isn't (or wasn't soft) Real Time

- Coarse Grained Synchronization
- Paging
- “Fairness” in Scheduling
- Request Reordering
- Batching

## Linux User Space Example

Interrupt flow in Linux for a task waiting for an I/O from disk:

- The I/O is complete. The device raises an interrupt. This cause the block device driver's ISR to run.
- The ISR checks the driver wait queue and finds a task waiting for I/O. It then calls one of the wake-up family functions. The function removes the task from the wait queue and adds it to the scheduler run queue.
- The kernel then calls the function `schedule()` when it gets to a point where scheduling is allowed.
- Finally `schedule()` finds the next suitable candidate for running. The kernel context switches to our task if it has sufficient high priority to get scheduled.

## Linux User Space Example

Kernel response time:

- Interrupt latency
- ISR duration
- Scheduler latency
- Scheduler duration

...how was (or will be) improved to have a deterministic and possibly low latency response

## Linux User Space Example

Interrupt Latency

### **PROBLEMS:**

- Disabling all interrupts for a long time (removal of all global cli in kernel 2.5);
- Registering a fast interrupt handler by improperly written device driver (mostly a programmer error).

## Linux User Space Example

ISR duration

### **SOLUTION:**

Interrupt handler in two stages

### **BUT:**

Soft irq are not deterministic

## Linux User Space Example

Scheduler latency

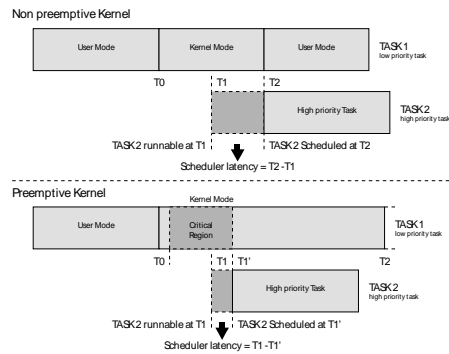
### **PROBLEMS:**

- Nonpreemptive nature of the kernel
- Interrupt disable times

### **SOLUTIONS:**

- Kernel Preemption – by Robert Love
- Low-Latency Patches – by Ingo Molnar and Andrew Morton

# Linux User Space Example



Scheduler latency in preemptable and nonpreemptable kernels

# Linux User Space Example

Scheduler duration

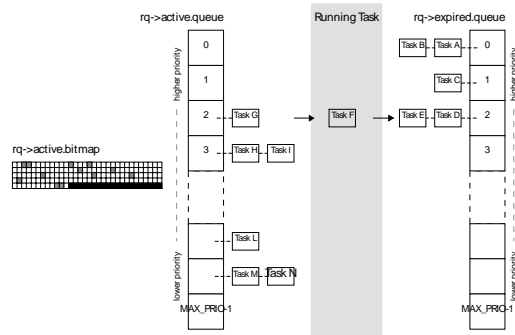
PROBLEMS:

- Non deterministic and slowly goodness and recalculation loop
- Different context switch time between processes and threads (no solution)

SOLUTION:

O(1) Scheduler – Ingo Molnar

# Linux User Space Example



Scheduler latency in preemptible and nonpreemptible kernels

## Linux

Not only User Space  
Approaches for an hard RTOS

## Linux hard RTOS

Approaches:

- Mono kernel (changing the source code along different Vanilla kernel)
- Dual kernel (only small changes to the Linux code)

## Linux hard RTOS

### **Dual kernel**

(most free)

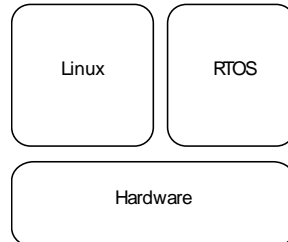
- RTAI/Xenomai
- RTLinuxFree
- ...

### **Mono kernel**

(most commercials)

- BlueCat Linux
- Cadenux
- Metrowerks
- MontaVista Linux
- RTLinuxPro
- TimeSys Linux
- ...

## Linux hard RTOS



Dual kernel approach: two OSs sharing the same hardware, how?

## ADEOS

Adaptive Domain Environment for  
Operating Systems

# ADEOS

Nano kernel  
Open source

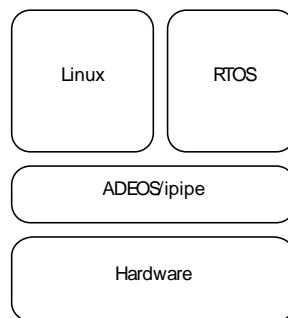
## **NEW NAME:**

ADEOS/pipe

## **CHANGES:**

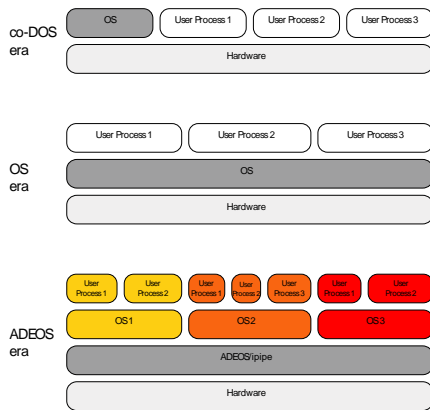
- Functions name
- Various improvements

# ADEOS



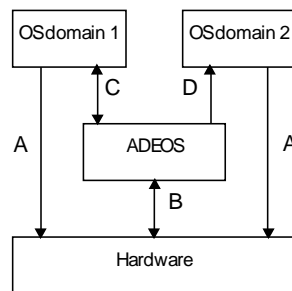
Dual Kernel with ADEOS/pipe nano kernel

# ADEOS



Many different OS (or multiple instance of the same OS) running on the same machine sharing the same hardware.

# ADEOS



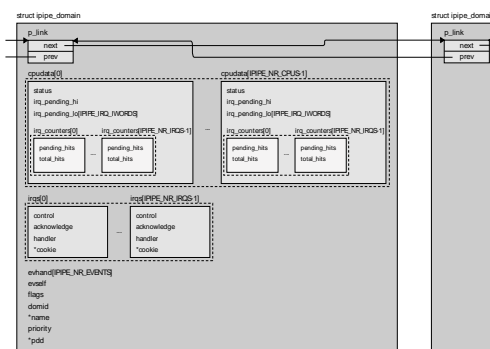
ADEOS/ipipe paths

# ADEOS

Basics:

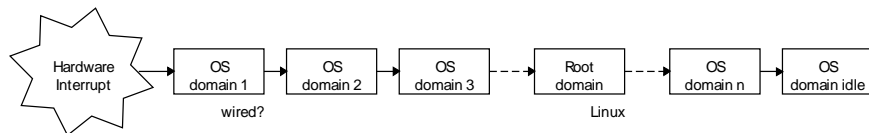
- Domains
- Pipeline
- Interrupts
- Events

# ADEOS



ADEOS/ipipe Domain descriptor

# ADEOS



ADEOS/ipipe interrupts/events pipeline  
(on interrupt)

# ADEOS

Interrupts

## **PROPERTIES:**

- Asynchronous
- Raised in Hardware
- Maskable (a domain could be stalled)
- Propagated down the pipeline
- Logged (Optimistic interrupt protection scheme)

# ADEOS

## Events

### **PROPERTIES:**

- Synchronous
- Raised mostly for software cause
- Non maskable
- Propagated from the source domain up the pipeline
- Immediate dispatching

# ADEOS

```
struct ipipe_domain domain_desc;
void domain_entry (int iflag) {
    if (iflag) {
        ipipe_virtualize_irq(...);
        ipipe_virtualize_irq(...);
        ipipe_virtualize_irq(...);
        ipipe_catch_event(...);
    }
    for (;;)
        ipipe_suspend_domain();
}
int init_module (void) {
    struct ipipe_domain_attr attr;
    ipipe_init_attr (&attr);
    attr.name = "MyDomain";
    attr.priority = IPIPE_ROOT_PRIO + 100; /* Precede Linux in the pipeline */
    attr.entry = &domain_entry;
    return ipipe_register_domain(&domain_desc,&attr);
}
void cleanup_module (void) {
    return ipipe_unregister_domain(&domain_desc);
}
```

ADEOS/ipipe domain creation example

# ADEOS

## References:

- Karim Yagmour et others (OperSys Inc)
- [www.adeos.org](http://www.adeos.org)

# RTAI

Real Time Application Interface  
by Dipartimento di Ingegneria  
Aerospaziale dell'Università di Milano  
(DIAPM)

## RTAI

### History (1/2):

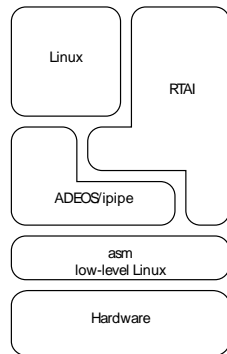
- In the first it was PCDOS-DIAPM-RTOS (16bits real mode);
- Than it was implemented in 32 bits real/protected mode;
- Approaching Linux (2.0.25);
- NMT-RTL appear and goes on DIAPM-RTL variant was born
- RTHAL-RTAI patch for 2.2.xx kernel ready (beginning 1999)

## RTAI

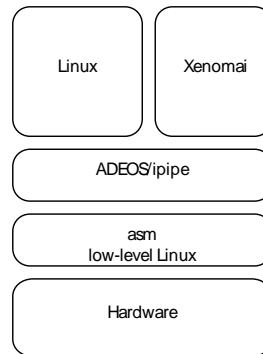
### History (2/2):

- April 1999 first version of RTAI for UP and SMP;
- Jan 2001 NMT-RTL now RTLinux was patented;
- Jun 2002 ADEOS was firstly released;
- ADEOS substitutes RTHAL and from now no patent problems;
- LXRT scheduler was introduced;
- More than one branch of RTAI;
- October 2005 Fusion branch become a stand-alone distribution Xenomai;

## RTAI VS Xenomai



RTAI immediate dispatching



Xenomai

RTAI pipeline (no immediate dispatching)

## RTAI VS Xenomai

### RTAI

Currently ported on:

- ARM
- I386, ia64 (SMP, MUP)
- PowerPC

Run:

- Kernel
- User

Main goal :

- Fast dispatching

API interface:

- RTAI
- Posix 1003.1b
- RTDM

### Xenomai

Currently ported on:

- ARM
- i386, ia64 (SMP)
- PowerPC, PowerPC64 (SMP)
- Blackfin

Run:

- Kernel
- User

Main goal :

- Portability

API interface:

- Xenomai native
- Posix 1003.1b
- RTAI
- VxWorks
- pSos+
- VRTX
- ulTRON
- RTDM

## RTAI VS Xenomai

From our tests:

**LESS INTEGRATION PROBLEMS:**

Xenomai

**BETTER PERFORMANCE:**

RTAI (immediate dispatching)

**EASY APPLICATION PORTING:**

Xenomai

**DEBUGGING SUPPORT:**

Both

**CLEAR, AND WELL ORGANIZED SOURCE CODE:**

Xenomai

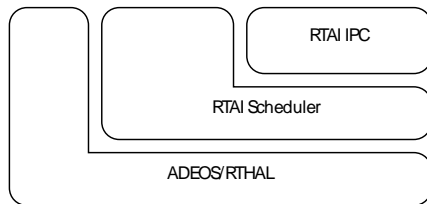
## RTAI

Immediate dispatching

# RTAI

Basics:

- ADEOS/RTHAL
- RTAI Scheduler
- RTAI IPC



# RTAI

ADEOS/RTHAL

**PROVIDES:**

- Immediate Interrupts dispatching
- Interrupt registering/deregistering functions
- PIC management routines
- Hardware Timers (8254/APIC) management routines
- registering/deregistering timers handler functions
- Trap handlers registering/deregistering functions
- Provides `rt_printk/rt_synch_printk`
- Syscall dispatching

# RTAI

## ADEOS/RTHAL interrupts

### Pure ADEOS Xenomai

Raised an External interrupt n

CPU decode phase

CPU call IDT[n] entry

```
linux/arch/i386/kernel/entry.S:404 interrupt[n]()
linux/arch/i386/kernel/entry.S:475 common_interrupt()
linux/arch/i386/kernel/ipipe-root.c: __ipipe_handle_irq(struct pt_regs
regs)
detect head Domain (Xenomai) is wired
linux/kernel/ipipe/core.c: __ipipe_dispatch_wired(struct ipipe_domain
*head, unsigned irq)
call Xenomai interrupt handler (registered with rthal_irq_request)
```

### RTHAL ADEOS modified patch

Raised an External interrupt n

CPU decode phase

CPU call IDT[n] entry

```
linux/arch/i386/kernel/entry.S:404 interrupt[n]()
linux/arch/i386/kernel/entry.S:475 common_interrupt()
rtai/base/arch/i386/hal/hal.immed:1474 rtai_hirq_dispatcher(struct
pt_regs regs)
call RTAI interrupt handler (register with rt_request_irq)
```

# RTAI

## ADEOS/RTHAL interrupts And Linux interrupts?

### Before RTAI

Raised an External interrupt n

CPU decode phase

CPU call IDT[n] entry

```
linux/arch/i386/kernel/entry.S:404 interrupt[n]()
linux/arch/i386/kernel/entry.S:416 common_interrupt()
linux/arch/i386/kernel/irq.c:54 do_IRQ(struct pt_regs *regs)
linux/kernel/irq/handle.c:118 __do_IRQ(unsigned int irq, struct pt_regs *regs)
linux/kernel/irq/handle.c:79 handle_IRQ_event(unsigned int irq, struct pt_regs *regs, struct irqaction
*action)
loop through the actions and call the interrupt handlers
```

### With RTAI

Raised an External interrupt n

CPU decode phase

CPU call IDT[n] entry

```
linux/arch/i386/kernel/entry.S:404 interrupt[n]()
linux/arch/i386/kernel/entry.S:475 common_interrupt()
rtai/base/arch/i386/hal/hal.c:1474 rtai_hirq_dispatcher(struct pt_regs regs)
call RTAI interrupt handler
rtai/base/include/asm-i386/rtai_hal.h:275 hal_pend_domain_uncond(irq, domain,cpuid)
rtai/base/include/asm-i386/rtai_hal.h:285 hal_fast_flush_pipeline(cpuid)
linux/arch/i386/kernel/ipipe-core.c: __ipipe_sync_stage(unsigned long syncmask)
for every interrupt pended in the log do
call handler
linux/arch/i386/kernel/irq.c:54 do_IRQ(struct pt_regs *regs)
linux/kernel/irq/handle.c:118 __do_IRQ(unsigned int irq, struct pt_regs *regs)
linux/kernel/irq/handle.c:79 handle_IRQ_event(unsigned int irq, struct pt_regs *regs, struct irqaction
*action)
loop through the actions and call the interrupt handlers
```

# RTAI

## 8254

Native RTAI timer, it supports at least two modes that are used:

- Periodic Mode
- Oneshot Mode

This timer is used in:

- UP
- SMP

Counter 0 shared with Linux.

ADEOS/RTHAL timers

## APIC

In the beginning was used only with multiprocessor hardware.

Same modes:

- Periodic Mode
- Oneshot Mode

Used in:

- SMP
- MUP

# RTAI



## Periodic Mode

Automatic timer register reload

## Oneshot Mode

Compulsory timer register update

# RTAI

```
bstatic int irq_handler(int irq, void *cookie) {
    // do something in IRQ handler

    rt_enable_irq(vmechip_irq);
    return IRQ_HANDLED;
}
static int VGD4_init(void) {
    result = rt_request_irq(vmechip_irq, VGD4_irqhandler, 0, 1);
    if (result != 0) {
        printk(KERN_ERR"VG4 driver: can't get assigned pci irq vector %02X\n", vmechip_irq);
        return -1;
    } else {
        rt_enable_irq(vmechip_irq);
    }

    return 0;
}
static void VGD4_exit(void) {
    rt_disable_irq(vmechip_irq);
    rt_release_irq(vmechip_irq);

    return;
}
```

ADEOS/RTHAL interrupt handler registration  
example

# RTAI

Scheduler

## **PROVIDES:**

- Task creation/destruction routine
- Task suspend/yield/sleep routine
- Task Timer related function
- Scheduling disciplines/policy
- LXRT Linux (user space) Real-Time capability
- Task interface with IPC mechanisms

# RTAI

## Scheduler

## MAIN ENTITY: task

```
typedef struct rt_task_struct {
    long *stack __attribute__((__aligned__(L1_CACHE_BYTES)));
    int uses_fpu;
    int magic;
    volatile int state, running;
    unsigned long runnable_on_cpus;
    long *stack_bottom;
    volatile int priority;
    int base_priority;
    int policy;
    int sched_lock_priority;
    struct rt_task_struct *prio_passed_to;
    RTIME period;
    RTIME resume_time;
    RTIME periodic_resume_time;
    RTIME yield_time;
    int rr_quantum;
    int rr_remaining;
    int suspdepth;
    struct rt_queue queue;
    int owdres;
    struct rt_queue *blocked_on;
    struct rt_queue msg_queue;
    int tid; /* trace ID */
    unsigned long msg;
    struct rt_queue ret_queue;
    void (*signal)(void);
    FPU_ENV fpu_env __attribute__((__aligned__(L1_CACHE_BYTES)));
    struct rt_task_struct *prev;
    struct rt_task_struct *next;
    struct rt_task_struct *tprev;
    struct rt_task_struct *tnext;
    struct rt_task_struct *rprev;
    struct rt_task_struct *rnext;

    /* Appended for calls from LINUX. */
    long *fun_args;
    long *bstack;
    struct task_struct *lxntsk;
    long long retval;
    char *msg_buf[2];
    long msg_size[2];
    char task_name[16];
    void *system_data_ptr;
    struct rt_task_struct *nextp;
    struct rt_task_struct *prevp;

    /* Added to support user specific trap handlers. */
    RT_TRAP_HANDLER task_trap_handler(HAL_NR_FAULTS);

    /* Added from rtai-22. */
    long unblocked;
    void *rt_signals;
    volatile unsigned long pstate;
    unsigned long usp_flags;
    unsigned long usp_flags_mask;
    unsigned long force_soft;
    volatile int is_hard;

    void *trap_handler_data;
    struct rt_task_struct *linux_syscall_server;

    /* For use by watchdog. */
    int resync_frame;

    /* For use by exit handler functions. */
    XHDL *ExitHook;

    RTIME nexttime[2];
    struct tcb_t tcb;

    /* Real time heaps. */
    struct rt_heap_s heap[2];

    volatile int scheduler;

#ifdef CONFIG_RTAI_LONG_TIMED_LIST
    rb_root_t rbr;
    rb_node_t rbn;
#endif
    struct rt_queue resq;
} RT_TASK __attribute__((__aligned__(L1_CACHE_BYTES)));
```

# RTAI

## task RT\_TASK

3 double linked lists:

- fields prev/next **task list**
- fields rprev/rnext **ready list**
- fields fprev/fnext **timed list**

A list head for every CPU

rt\_smp\_linux\_task[cpuid]

# RTAI

task **RT\_TASK**  
task list

## INSERTION POLICY:

Add every new created task

## LIST ORDER:

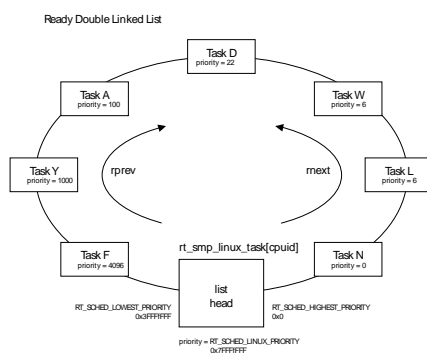
FIFO

## NOTE:

Tail task next = 0

# RTAI

task **RT\_TASK**  
ready list



## INSERTION POLICY:

Add ready task

## REMOVE POLICY:

First task after the head

## LIST ORDER:

Sorted on `rt_task.priority`

## MOST PRIORITARY:

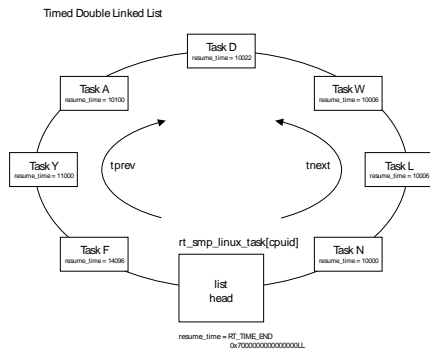
0

## LESS PRIORITARY:

RT\_SCHED\_LINUXPRIO

# RTAI

task RT\_TASK  
timed list



**INSERTION POLICY:**

Add timed task

**REMOVE POLICY:**

First n deadlines tasks after the head

**LIST ORDER:**

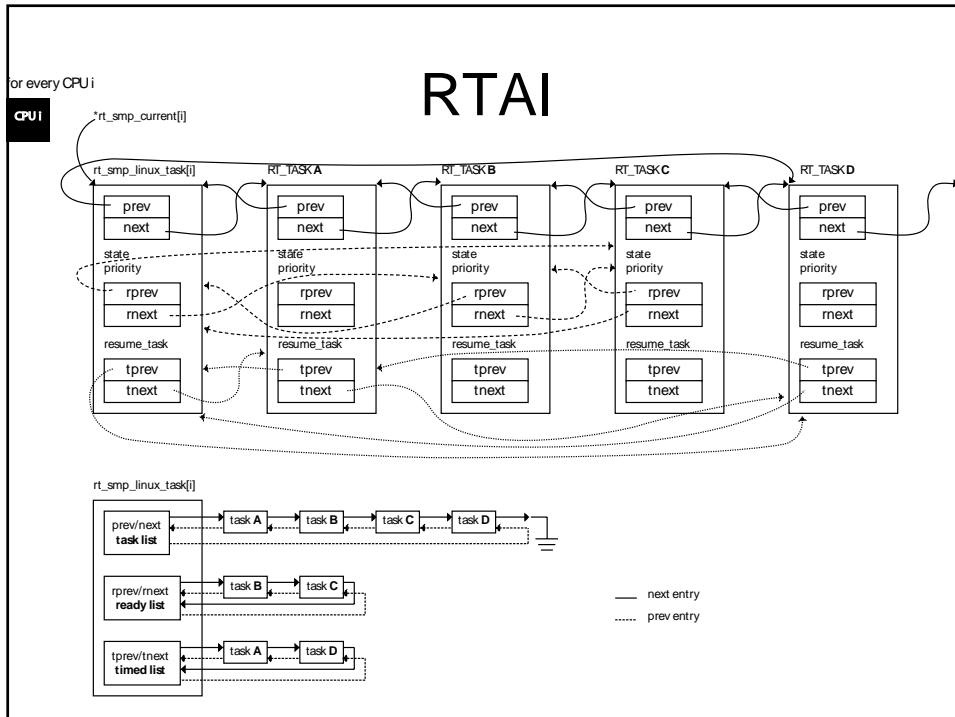
Sorted on  
rt\_task.resume\_time

**MOST PRIORITY:**

0

**LESS PRIORITY:**

MAX\_TIME



# RTAI

Scheduler

Scheduling policy:

- RMS
- EDF
- FIFO, **default scheduler**
- RR

# RTAI

Scheduler

Scheduling point:

- on timer interrupt
- on ISR exit
- on task self-suspending

Self-suspending examples:

- Yield
- Suspend
- Sleep

# RTAI

Scheduler

LXRT

Scheduler kernel

Scheduler user

# RTAI

```
void task_body (long cookie) {
    int i, l;
    while(1){
        for(i=0; i<100; i++) {
            l += i*(l+i);
        }
        printk(KERN_ERR"kernel_task executed\n");
        rt_task_wait_period();
    }
}

int init_module (void) {
    int err;
    err = rt_task_init(&task_desc, task_body, 0, TASK_STKSZ, TASK_PRIO, 0, 0);
    if (err != 0) {
        printk(KERN_ERR"error module loading\n");
        return -1;
    }
    rt_set_oneshot_mode();
    start_rt_timer(0);
    rt_task_make_periodic_relative_ns(&task_desc, 10000000, 10000000000);
    return 0;
}

void cleanup_module(void) {
    stop_rt_timer();
    rt_task_delete(&task_desc);
}
}
```

Scheduler task creation example

# RTAI

IPCs

**PROVIDES:**

- Bits (a sort of signals)
- Fifo
- Message
- Mailbox
- Message queue
- NetRPC (now require RTNet)
- Semaphore
- Shared memory

# RTAI

RTAI References:

Paolo Mantegazza, DIAPM

[www.rtai.org](http://www.rtai.org)

Xenomai References:

[www.xenomai.org](http://www.xenomai.org)