

# ***Breve introdução ao “Real Time Application Interface” (RTAI)***

**Sistemas de Tempo-Real  
DETI/UA**

Paulo Pedreiras  
DETI/UA  
Set/2012

# Conteúdo

- Requisitos
- Kernel Linux
- O RTAI
- Como carregar uma aplicação
- Anatomia de uma aplicação

# Requisitos

- Muitas aplicações integram componentes **com e sem requisitos de tempo-real**, logo poderão beneficiar da existência de plataformas com suporte e serviços “standard”, sem requisitos de tempo-real.
- Estas plataformas são habitualmente compostas por um **sistema operativo standard**, que fornece um leque **alargado** de serviços (e.g. *file-system*, HMI, ...) e um executivo/kernel **tempo-real** para suporte aos componentes com este tipo de requisitos
- Soluções baseadas em Linux:
  - Suporte tempo-real nativo em Linux
  - Modificações ao kernel para tempo-real (low-latency patch, KURT, ...)
  - Linux como uma tarefa de um kernel tempo-real (RTAI, RTLinux, ...)

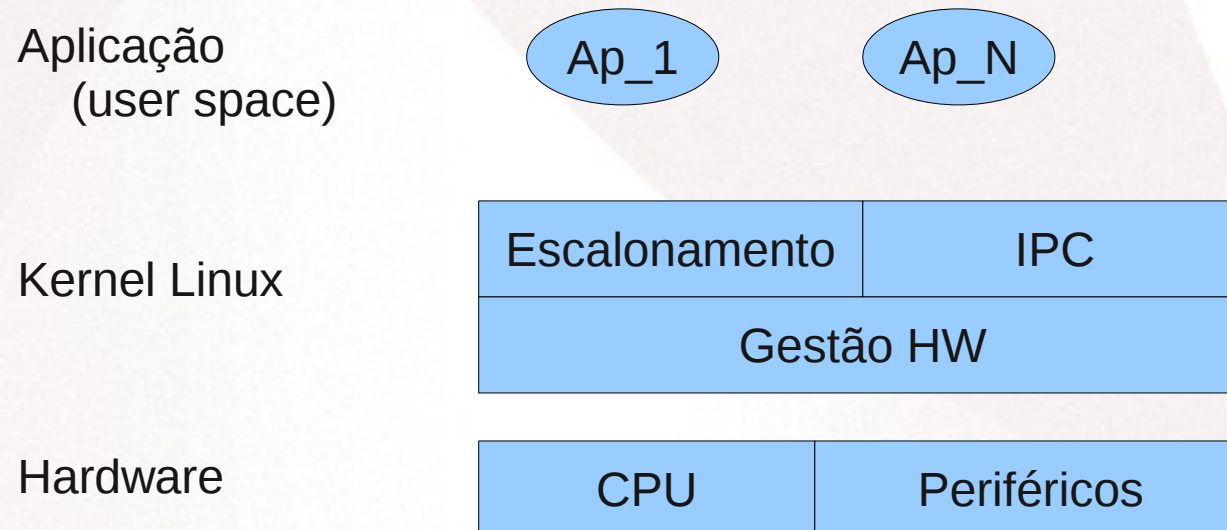
# ***Suporte a tarefas tempo-real em Linux***

- O Linux suporta o as “POSIX 1003.13 real-time extensions”, todavia ...
- O suporte a tarefas tempo-real em Linux usando POSIX é **pouco eficiente**:
  - O kernel sofre de **longos tempos de bloqueio**
    - Resultado de um critério de desenho que beneficia o *throughput*
  - **Preempção** de tarefas durante system calls **não é permitida**
    - Tarefas de alta prioridade podem ser bloqueadas por tarefas de baixa-prioridade por períodos relativamente longos
  - Mecanismos de **escalonamento pouco flexíveis** e eficientes
    - Apenas Round-Robin e FIFO para tempo-real

# Serviços do kernel Linux

O kernel Linux oferece às aplicações (entre outros) os seguintes serviços:

- **Camada de gestão de hardware:** gestão de periféricos e do CPU (tratamento de eventos, interrupções, ...)
- **Camada de escalonamento:** responsável pela ativação de processos, prioridades, ...
- **Comunicação:** comunicação entre processos (FIFO, *shared memory*, ...)



# O “*Real-Time Application Interface*”

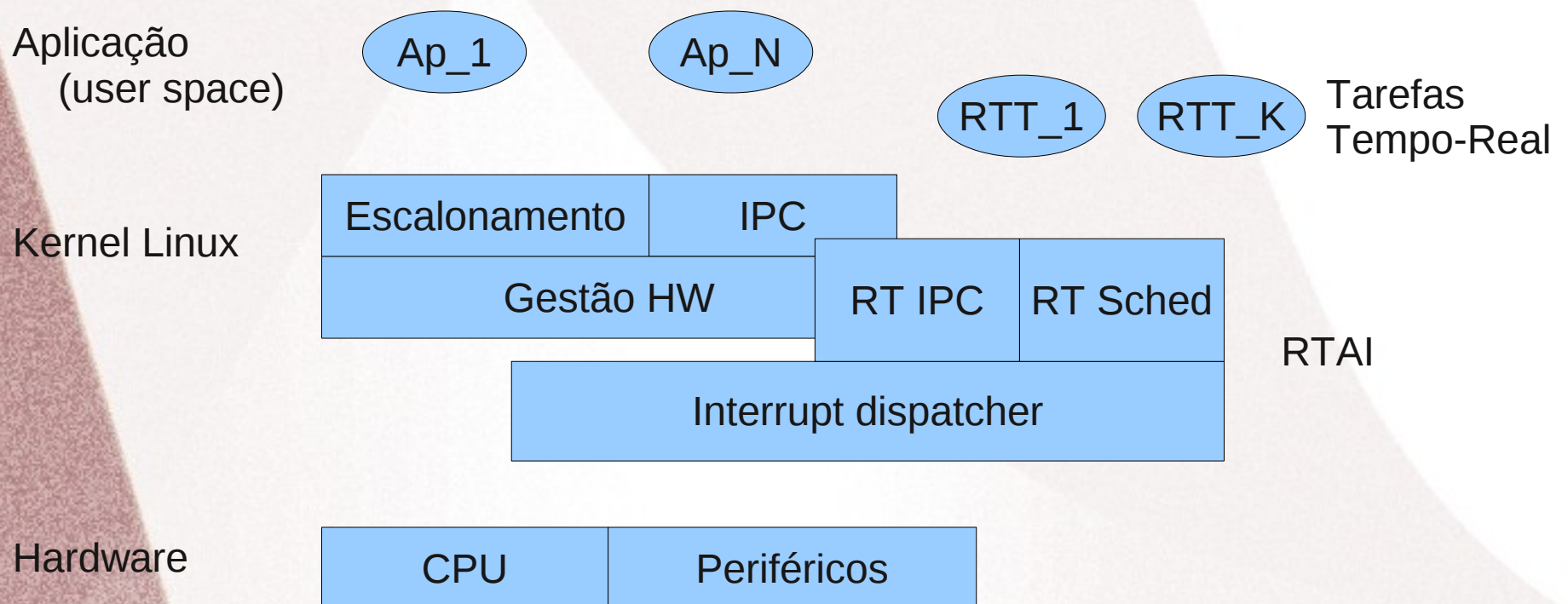
Conjunto de módulos de kernel Linux, com as seguintes propriedades:

- **Linux** torna-se um tarefa **background**
- Captura e age como **dispatcher de interrupções**. Interceta todas as interrupções encaminhando-as, se necessário para o kernel Linux
- Torna o **kernel** Linux completamente **preemptível**
- Pouco intrusivo para o kernel Linux. Funcionalidade do kernel não é alterada.

# Real-Time Application Interface

Disponibiliza as seguintes classes de serviços:

- **Gestão de HW** (periféricos)
- **Escalonamento**, gerindo tarefas, prioridades, ativações, ...
- **Comunicação** entre tarefas tempo-real e entre os domínios tempo-real e processos Linux “normais” (FIFO, *shared memory*, mailbox, ...)



# Blocos RTAI

- A arquitectura de software do RTAI é composta por:
  - Interface para o “Linux HW Management” (HAL): basicamente uma estrutura de dados
  - *Dispatcher*, *scheduler* e IPC
  - Interface usado pelas tarefas do utilizador para gerir componentes
- Do ponto de vista do **Linux** estes serviços encontram-se encapsulados em **módulos do kernel**.



# *Virtualização das interrupções*

- O RTAI mantém uma imagem do estado das interrupções visto pelo Linux.
  - Funções **Cli ()** and **Sti ()** afetam apenas uma estrutura de dados do RTAI e **não o controlador de interrupções** propriamente dito
- As **interrupções reais são intercetadas** pelo RTAI. São encaminhadas para o Linux apenas quando apropriado (RSI registada e interrupção *enabled*).
  - Linux **não bloqueia** tarefas tempo-real. Do ponto de vista do RTAI o *kernel* torna-se completamente preemptível

# ***Real-Time Application Interface***

- Principais módulos

- Rtai\_hal:

- **Criação** e **inicialização** de estruturas de dados (TCBs, ..), inicialização do esquema de captura e gestão de interrupções

- Rtai\_sched:

- Atribui o **CPU** às diversas **tarefas** de acordo com a sua prioridade.
    - Diversas classes de escalonadores suportados (processador único, multi-processador)

- Rtai\_shm/Rtai\_fifos/....:

- Mecanismos **IPC**. Serviços para criação, leitura e escrita. Mecanismos funcionam entre tarefas tempo-real e entre estas e processos linux (vistos como /dev/fifo\_x; /dev/shm\_x; ...)

# Carregar uma aplicação

Componentes RTAI são vistos pelo Linux como módulos do Kernel

- Adicionar módulo
  - **`/sbin/insmod mod_name.ko`**
- Remover módulo
  - **`/sbin/rmmod mod_name`**
- Listagem de módulos carregados
  - **`/sbin/lsmmod`**
- Informação de estado e debug via “kernel ring buffer”
  - Instrução **`dmesg`** para consultar

# *Carregar uma aplicação*

## **Passo 1:** carregar o RTAI

- Exemplo de script para carregar RTAI

...

```
prefix="/usr/realtime"
```

```
insmod='sudo /sbin/insmod'
```

```
MODULES=$prefix/modules
```

```
sync
```

```
insmod $MODULES/rtai_hal.ko IsolCpusMask=0;
```

```
insmod $MODULES/rtai_up.ko;
```

```
insmod $MODULES/rtai_sem.ko;
```

...

# *Carregar uma aplicação*

- **Passo 2:** carregar o módulo da aplicação  
`sudo /sbin/insmod app_name.ko`
- Os módulos RTAI necessitam de ser carregados **apenas uma vez** após o arranque do sistema
- A aplicação pode ser carregada/descarregada o **número de vezes necessário** (e.g. para debug)
- Os módulos RTAI podem ser **removidos** e novamente **carregados** a **qualquer altura** sem que o Linux seja afectado

# *Anatomia de uma aplicação*

- Duas funções especiais
  - Função de **inicialização**
    - Executada automaticamente aquando do carregamento do módulo
      - Alocação de recursos (e.e shared memory, fifos), criação das tarefas, inicialização do sistema de contagem de tempo
  - Função de **terminação**
    - Executada automaticamente aquando da remoção do módulo
      - Paragem das tarefas, libertação dos recursos alocados
  - Não há função “main()”!

# Anatomia de uma aplicação

- **Função de inicialização**

```
int __str_test_init(void)
{
    RTIME tick_period;
    RTIME startTime;
    int res;
    printk("[str_test] Initializing RTAI module ...\n");
    /* RT FIFO initialization */
    res = rtf_create(FIFO_ID, FIFO_SIZE);
    if (res) { ...
    /* Initialize user RT tasks */
    rt_task_init(&utask0, fun0, 0, STACK_SIZE, 0, USE_FPU, 0);
    ...
}
```

# *Anatomia de uma aplicação*

```
/* Compute system tick (tick_period in internal timer units) */  
tick_period = start_rt_timer(nano2count(PERIOD));  
startTime = rt_get_time() + NUM_TASKS*tick_period*1000;  
/* Start User RT Tasks */  
rt_task_make_periodic(&utask0, startTime, TASK0_PERIOD*tick_period);  
...  
printk("[str_test] Module successfully loaded!\n");  
return 0;  
}
```



# Anatomia de uma aplicação

- **Função de remoção**

```
void __str_test_exit(void)
{
    int res;
    stop_rt_timer();
    rt_busy_sleep(10000000);
    /* Remove user RT tasks */
    rt_task_delete(&utask0);
    ...
    /* Release FIFO */
    res=rtf_destroy(FIFO_ID);
    ...
    printk("\n Finito!\n");
}
```

# Anatomia de uma aplicação

- **Corpo de uma tarefa**

```
static void fun1(long t)
{
    /* Local vars and startup code here (executed once)*/
    ...
    /* Task body */
    while(1) {
        /* task code here */
        ....
        /* Can use IPC */
        rtf_put(FIFO_ID,&tevt,sizeof(tevt));
        ....
        /* Sleep until next activation */
        rt_task_wait_period();
    }
}
```