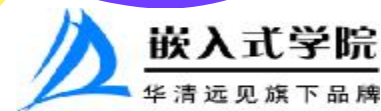
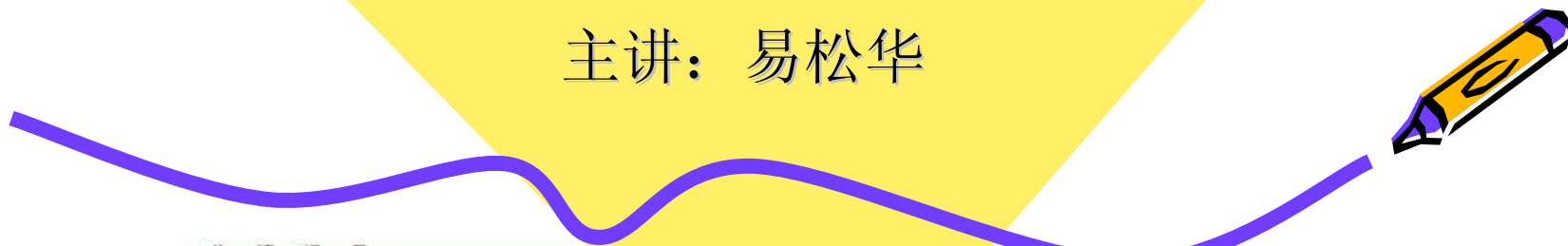


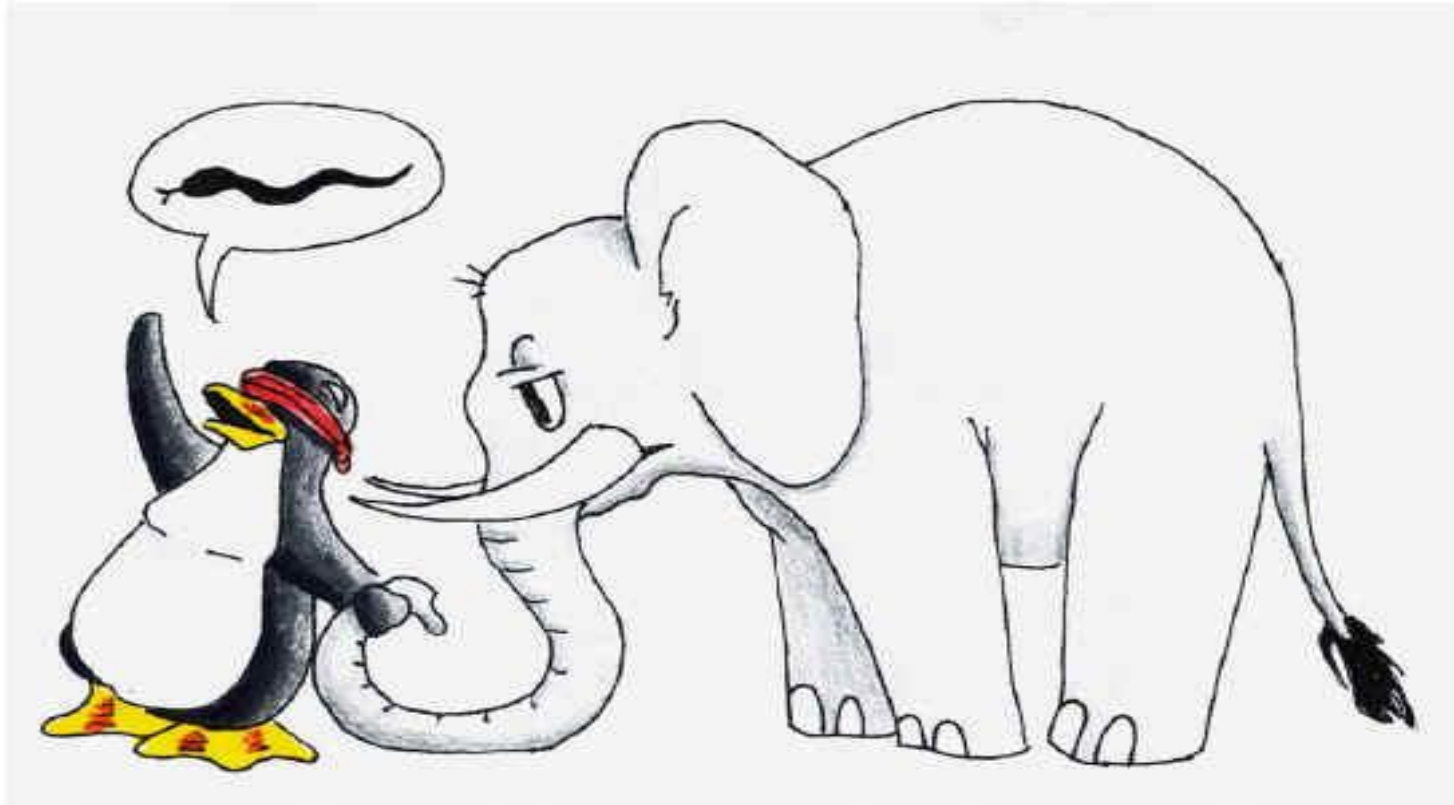


实时Linux技术

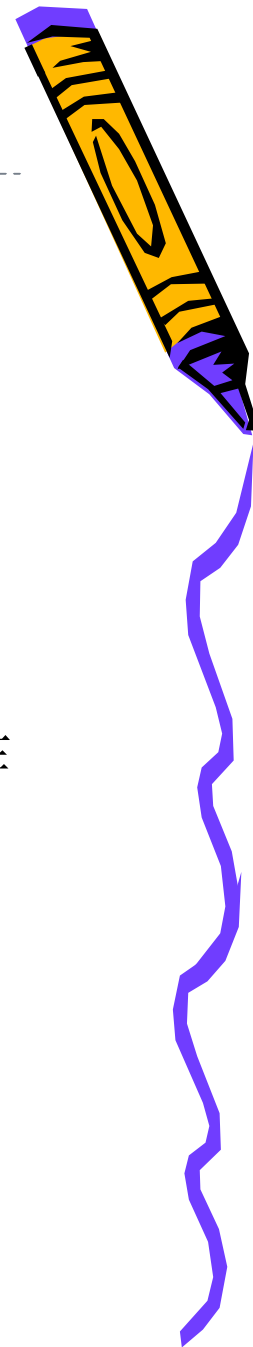
如何在嵌入式Linux中应用实时特性

主讲：易松华





实时VS 快速：怎样选择？

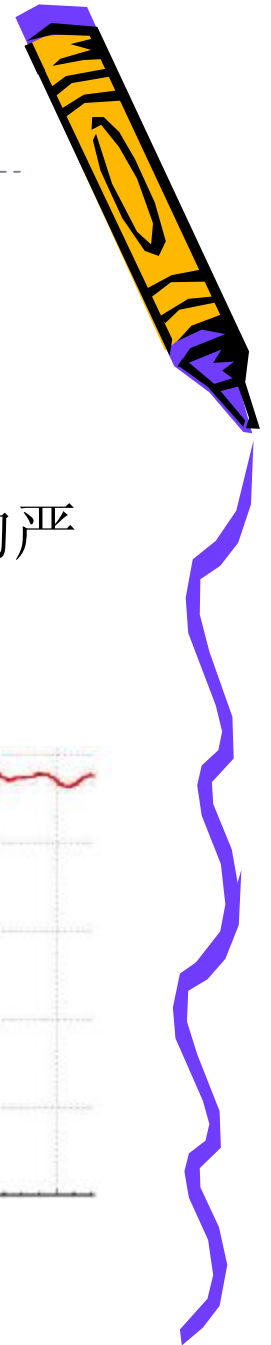
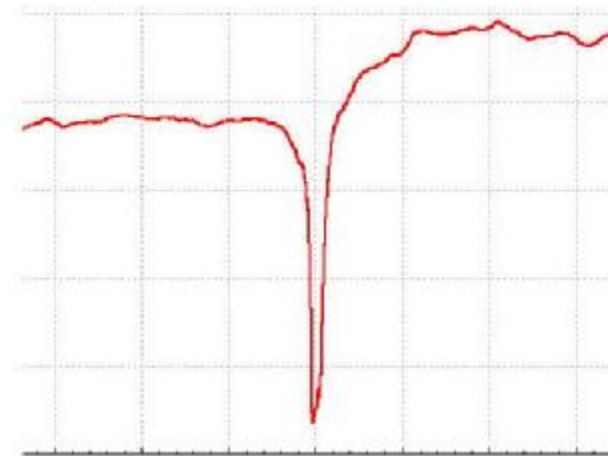


- 实时操作系统 (Real-time OS)
- 分时操作系统 (Time-Sharing OS)
 - 计算机资源会被平均地分配给系统内所有的工作
- 区别：
 - 是否有“时限(deadline)”

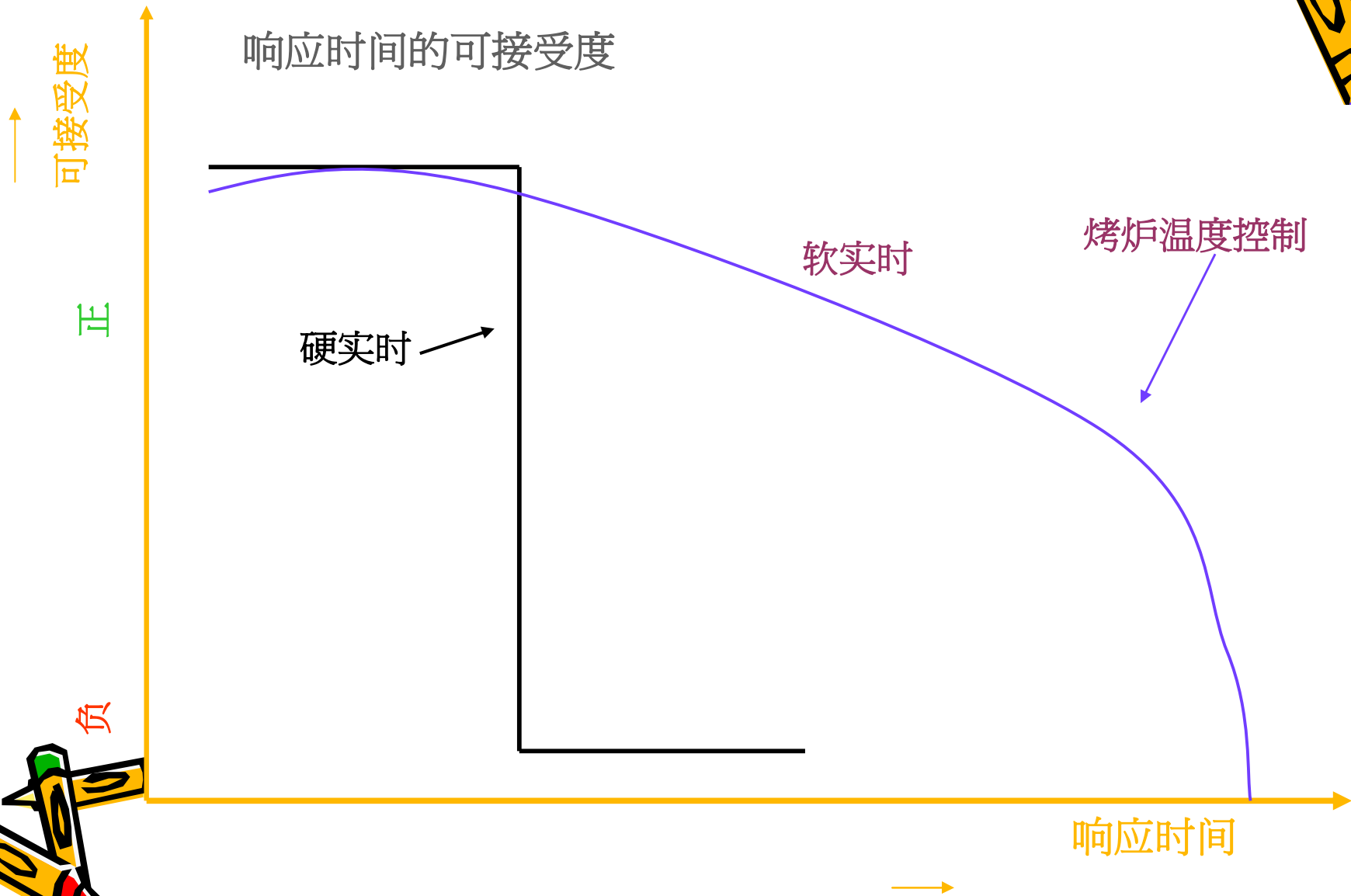
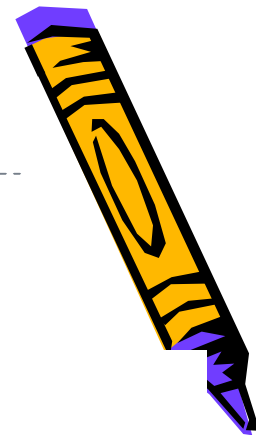


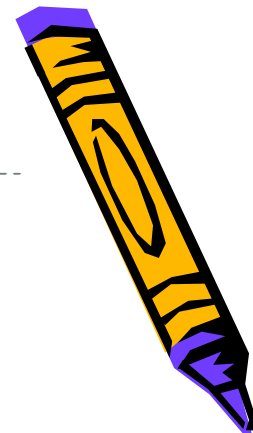
实时系统特点

- 指规定的时限内必须完成规定的操作
 - 并非指速度快慢
 - 硬实时：超过时限完成任务会导致灾难性后果
 - 软实时：超过时限完成对任务会带来系统性能的严重下降
- 手持设备什么地方需要实时系统？



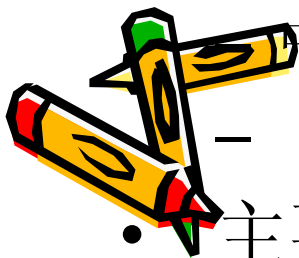
两种类型的Real-Time





实时系统相关技术和衡量指标

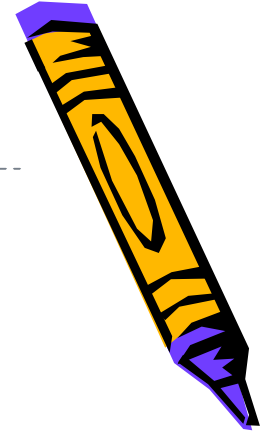
- 多任务和任务切换：任务切换时间
- 内核调度：调度算法
- 可抢占型内核和不可抢占型内核
- 优先级反转问题
- 任务间同步和通信
- 中断延迟
 - 中断延迟时间：关中断的最长时间+ 开始执行中断服务程序的时间
 - 中断响应时间：中断延迟时间+ 操作系统保存现场的时间
 - 中断恢复时间：恢复现场时间+ 执行中断返回指令的时间
 - 中断处理时间
- 主要实时指标：中断延迟、调度延迟



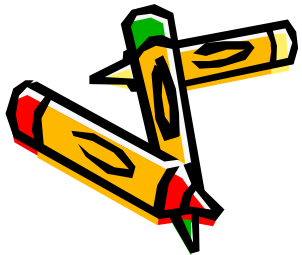
- 高优先级的任务先行
ALWAYS

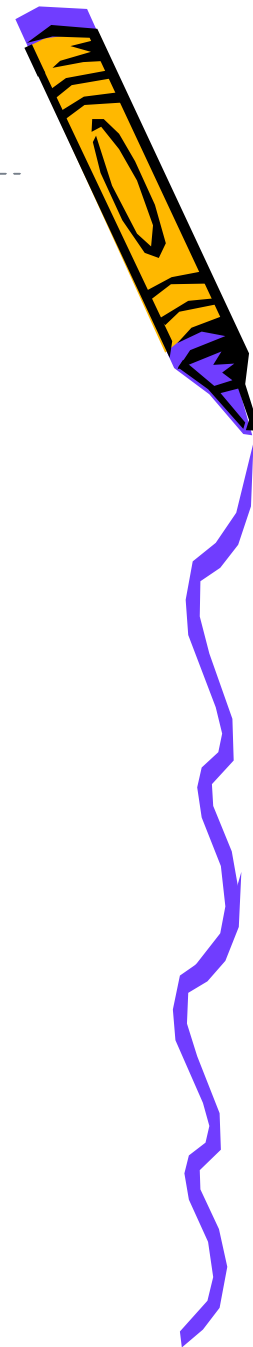
- 所以原则就是：
 - Everything should be pre-emptable
 - Nothing should keep higher priority things from executing





实时Linux概况

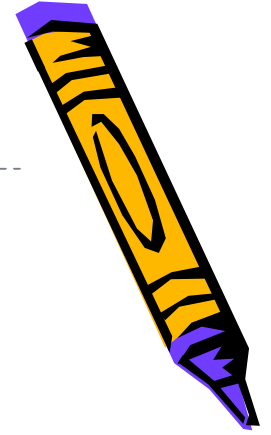




提高Linux实时性能方法

- 双内核方式
 - 实现硬实时
 - 实时内核+标准内核
 - RTLinux、RTAI 和Xenomai等
- 在主流内核上通过patch增加其实时性
 - 提高软实时内核性能
 - TimeSys Linux/RT
 - Montavista Linux
 - Ingo Molnar's RT patch

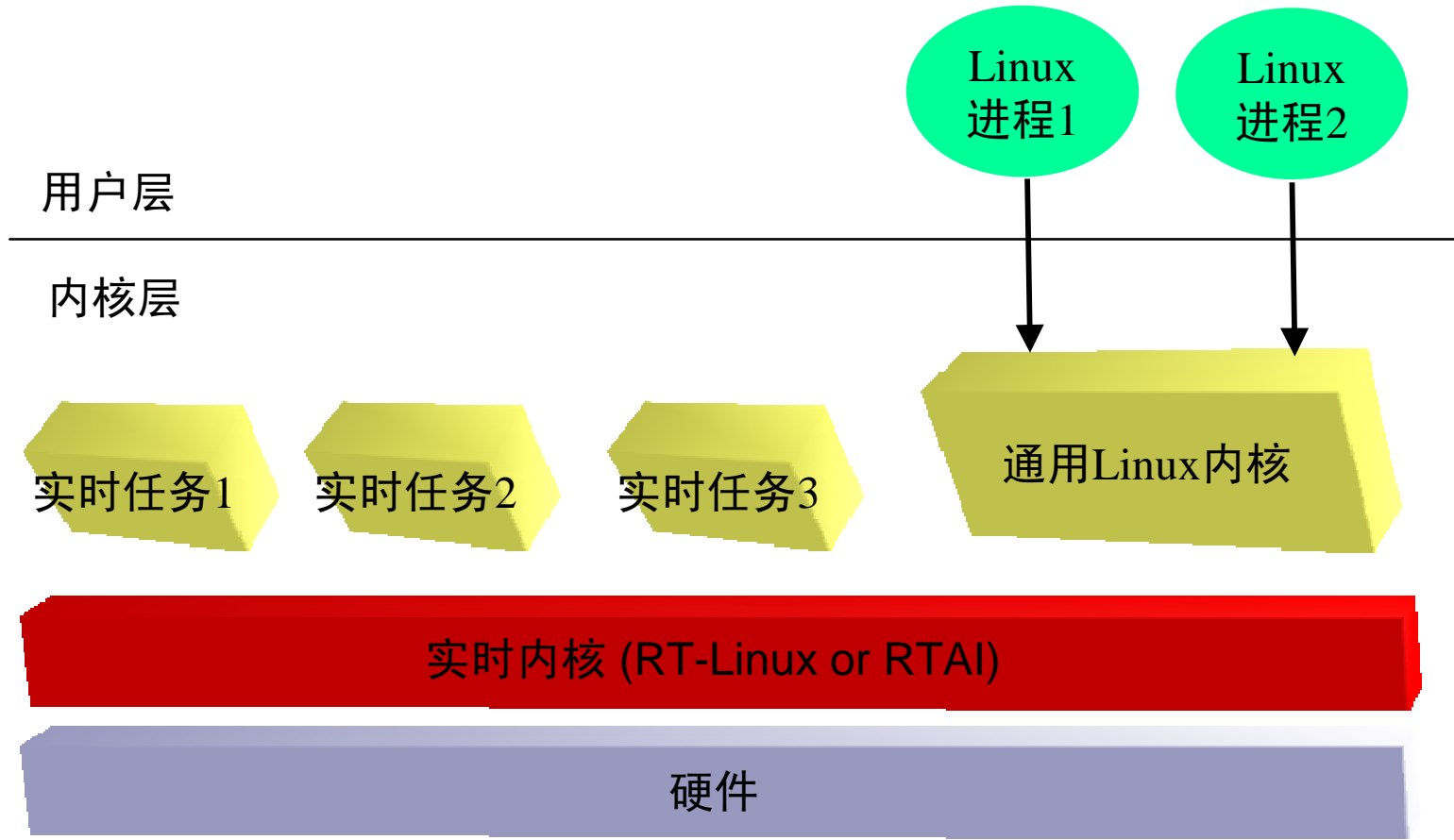




Linux硬实时



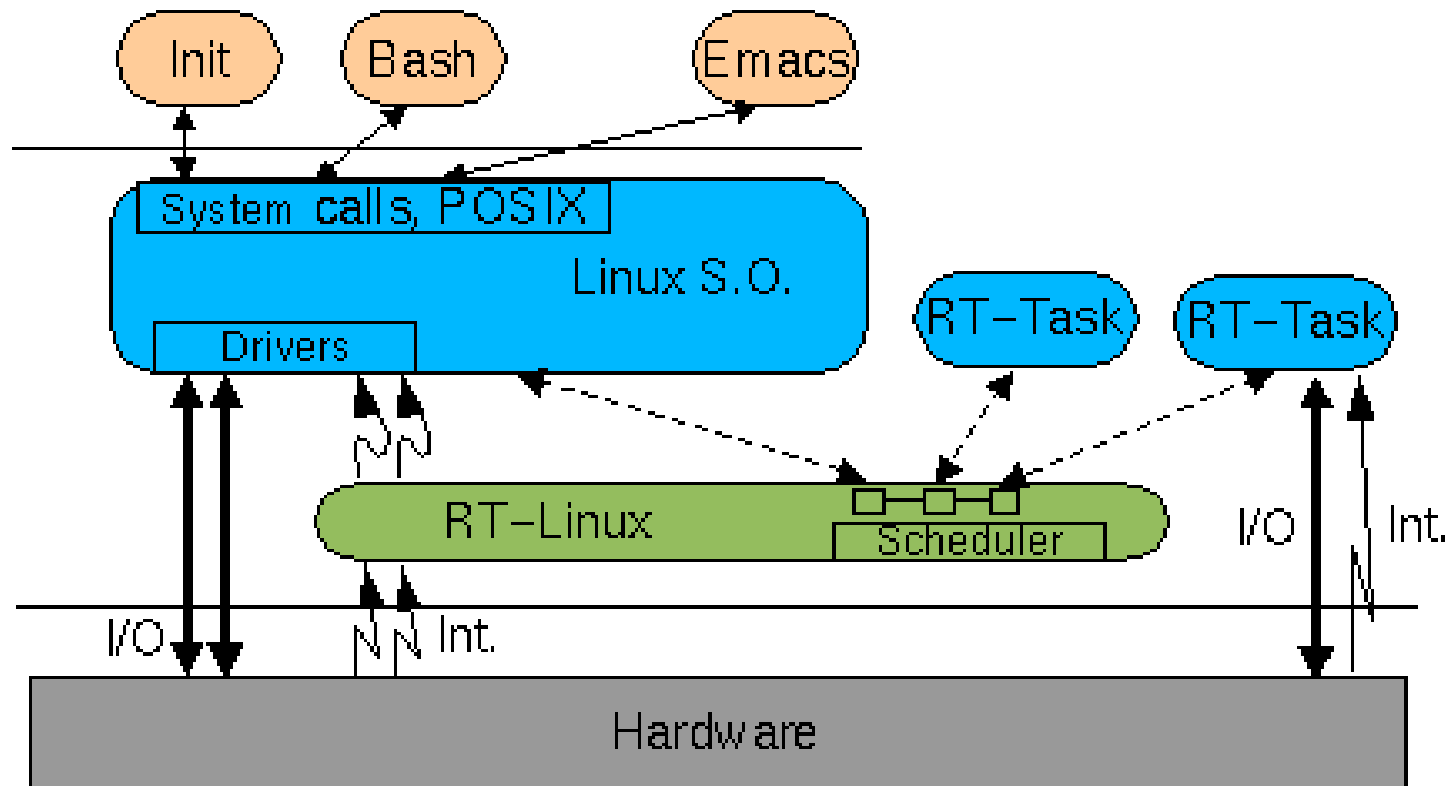
RTLinux



Real-time任务不使用LINUX API或资源
任何实时任务出问题将导致整个系统崩溃

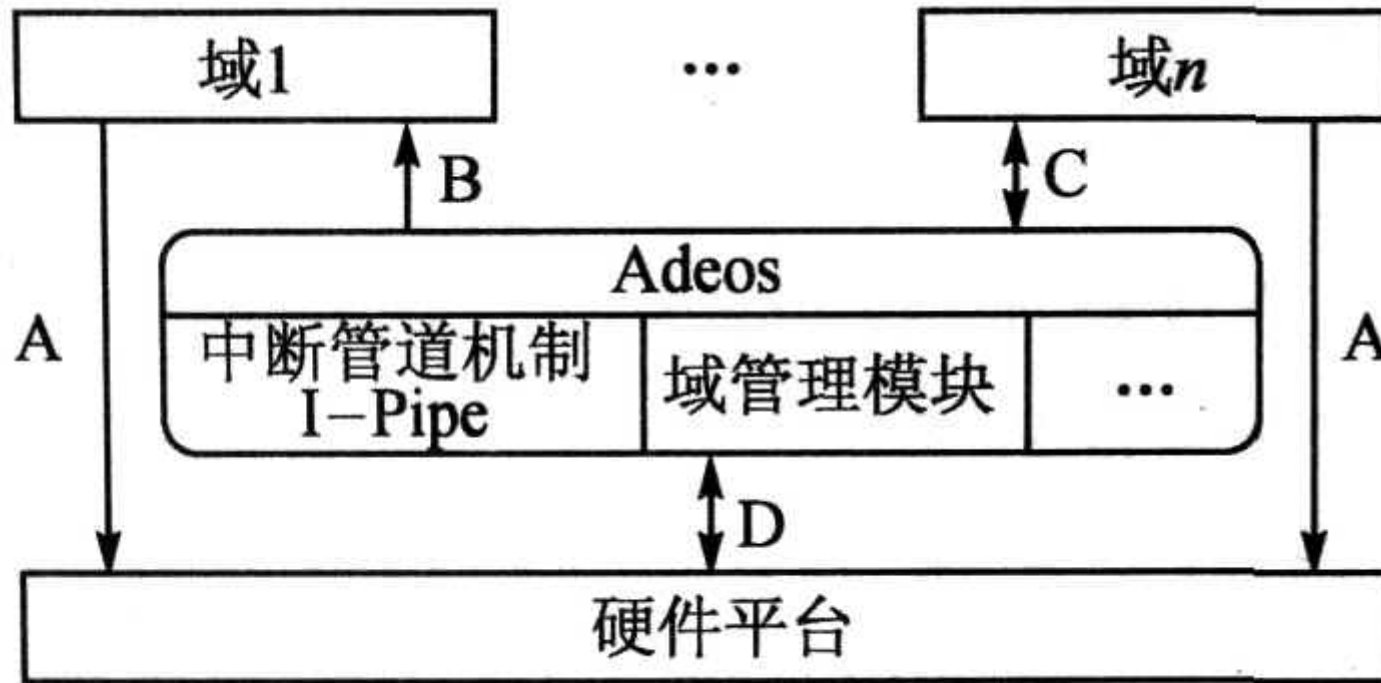
contd...

RTLinux

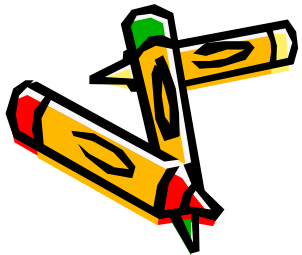


contd...

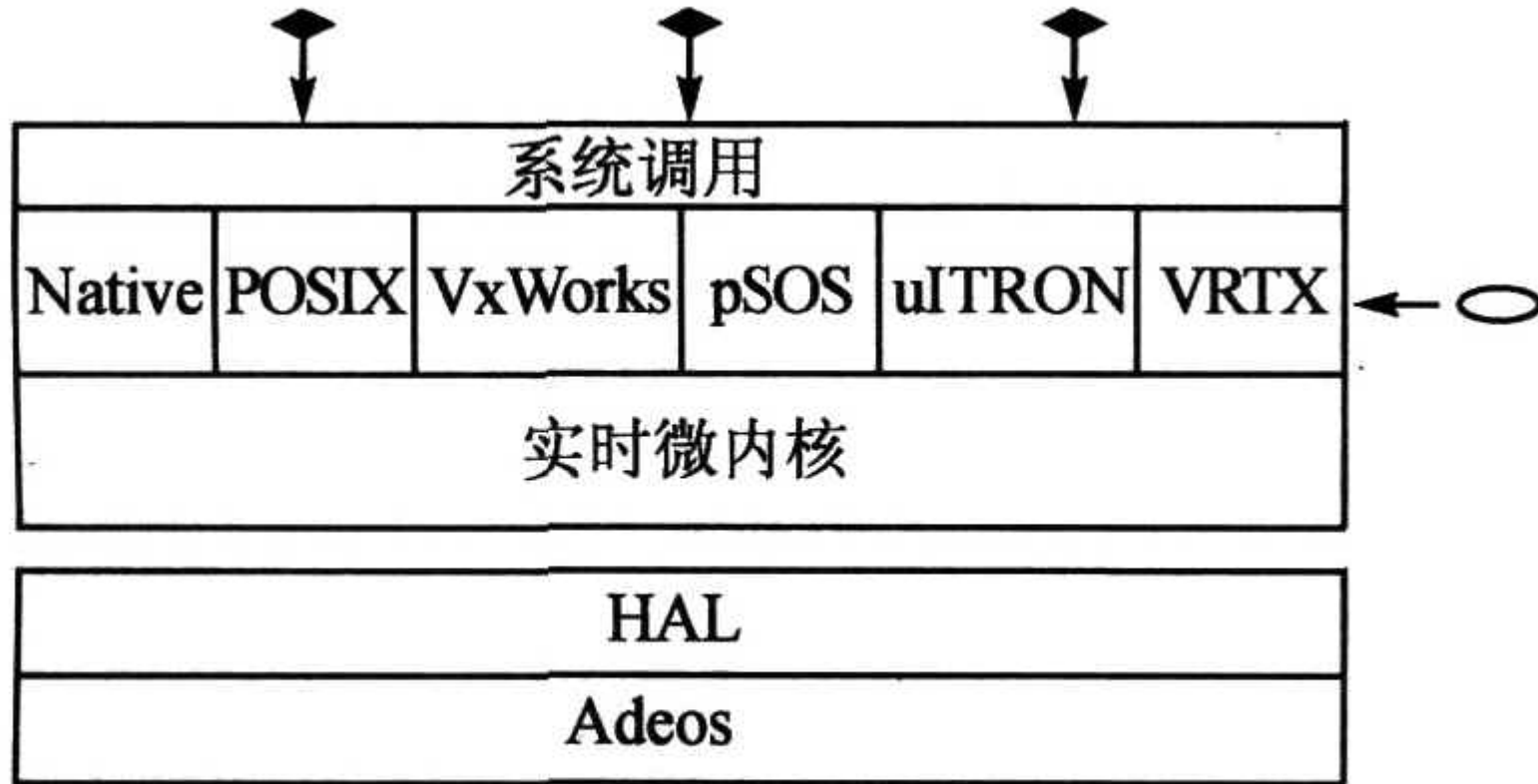
Xenomai



Xenomai 示意图



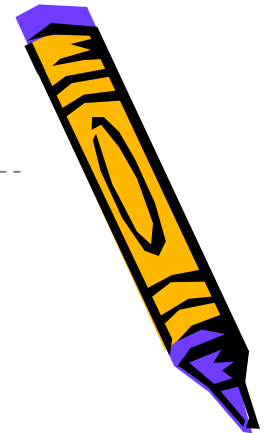
Xenomai



◆ 用户层应用程序 ○ 基于内核的应用程序

Xenomai SKIN 示意图

contd...

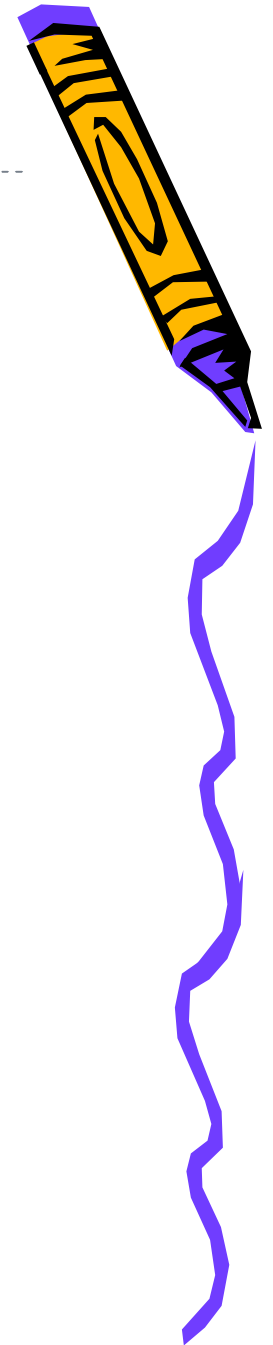


Xenomai 编程

```
...
#define ITER 10

static RT_TASK t1;
static RT_TASK t2;
int global = 0;

void taskOne(void *arg)
{
    int i;
    for (i=0; i < ITER; i++)
    {
        rt_printf("I am taskOne and global = %d.....\n", ++global);
    }
}
void taskTwo(void *arg)
{
    int i;
    for (i=0; i < ITER; i++)
    {
        rt_printf("I am taskTwo and global = %d-----\n", --global);
    }
}
```



contd...

Xenomai 编程 (Cont')

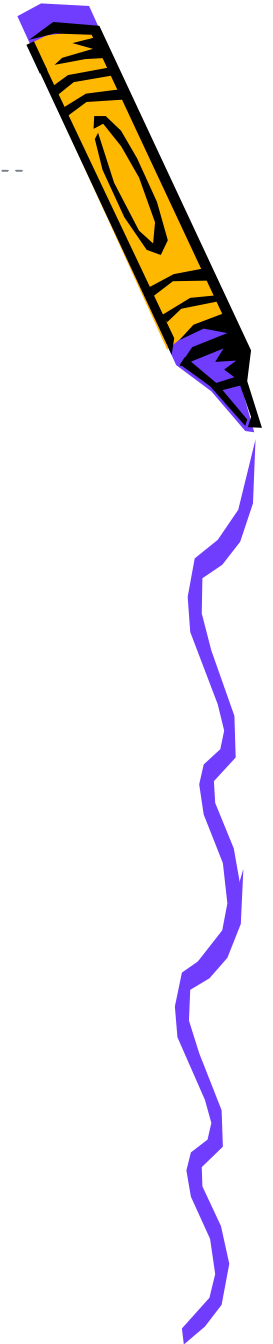
```
int main(int argc, char* argv[])
{
    /* Perform auto-init of rt_print buffers if the task doesn't do so */
    rt_print_auto_init(1);

    /* Avoids memory swapping for this program */
    mlockall(MCL_CURRENT|MCL_FUTURE);

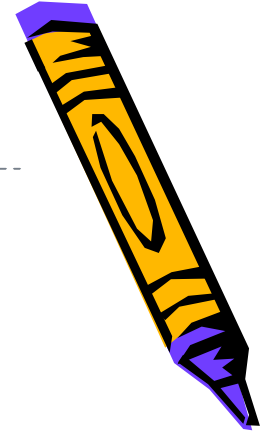
    /* create the two tasks */
    rt_task_create(&t1, "task1", 0, 1, 0);
    rt_task_create(&t2, "task2", 0, 1, 0);

    /* start the two tasks */
    rt_task_start(&t1, &taskOne, 0);
    rt_task_start(&t2, &taskTwo, 0);

    return 0;
}
```

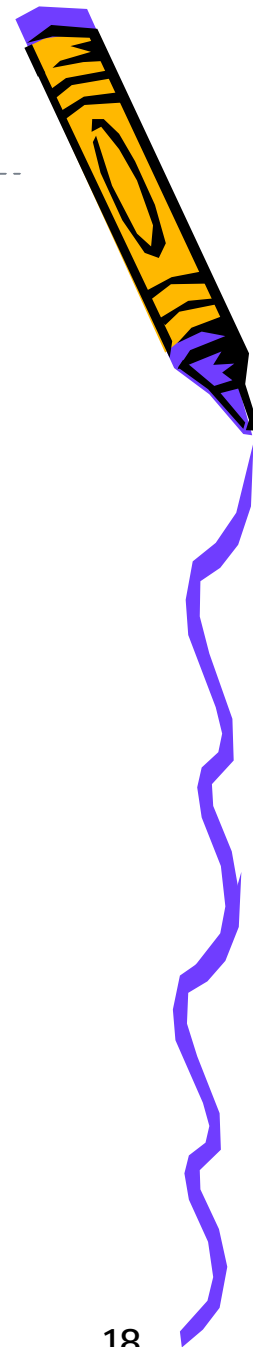


contd...



Linux软实时





- **Linux 实时化采取的措施:**
 - 最小化关中断时间
 - 中断线程化
 - 使内核完全可抢占
 - 减少临界区
 - 用**mutex**来执行同步 (减少spin locks)
 - 允许自愿抢占
 - **Mutex**支持优先级继承
 - 高精度定时器
 - 最小化**mutexes** 和临界区
 - 优化调度的策略





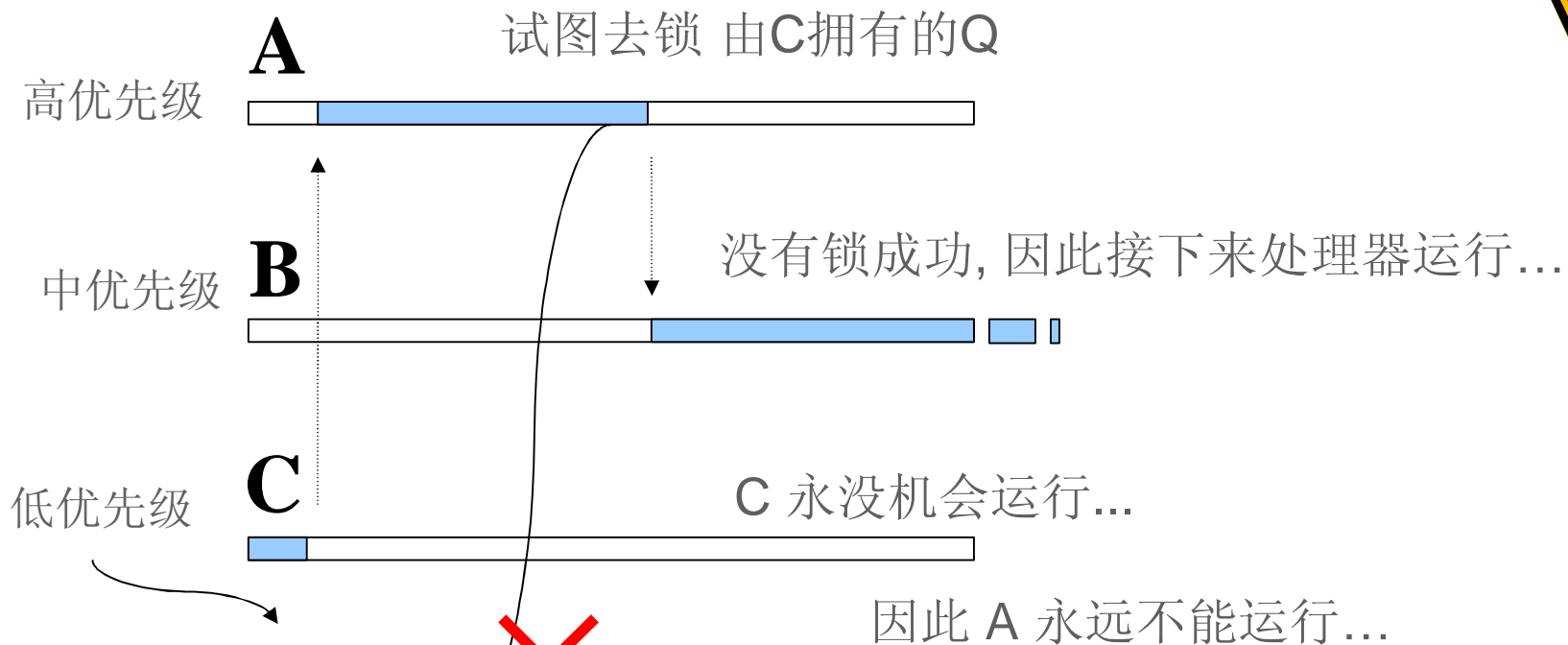
- 原来的Linux UP自旋锁：
 - 锁住时候禁用IRQ – 不能打断
 - 对RT来说不好
- 原来的Linux SMP自旋锁：
 - 自旋（锁总线）
 - 性能不好



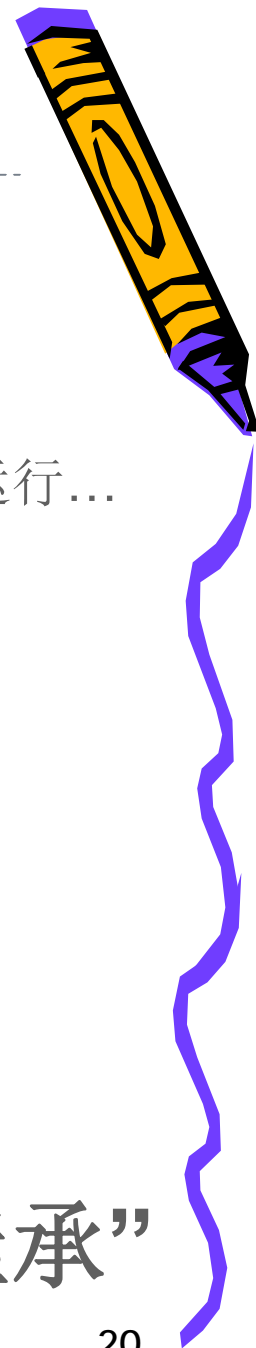
Solution:
“睡眠Spinlock”



问题：优先级反转

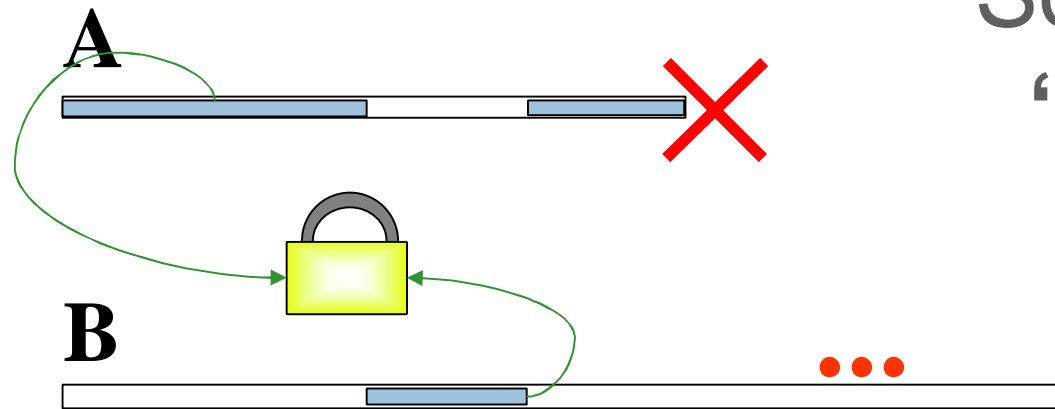


Solution:
“Priority继承”

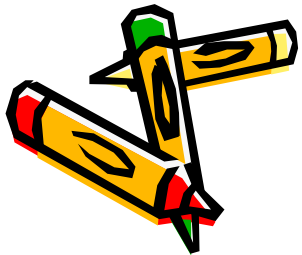


Robust Mutexes (鲁棒性互斥锁)

- 问题:
- 进程间信号量
- 进程A 获取了信号量并且异常退出了
- 进程B 在同一信号量处阻塞
- 在一般LINUX里面: *mutex locked forever*
 - 因此 进程B被永远阻塞了
 - ...直到系统重启



Solution:
“Robust
Mutex”



优先级队列

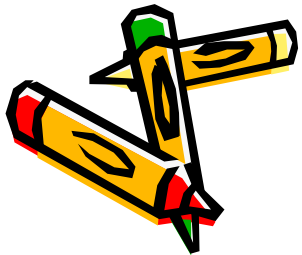
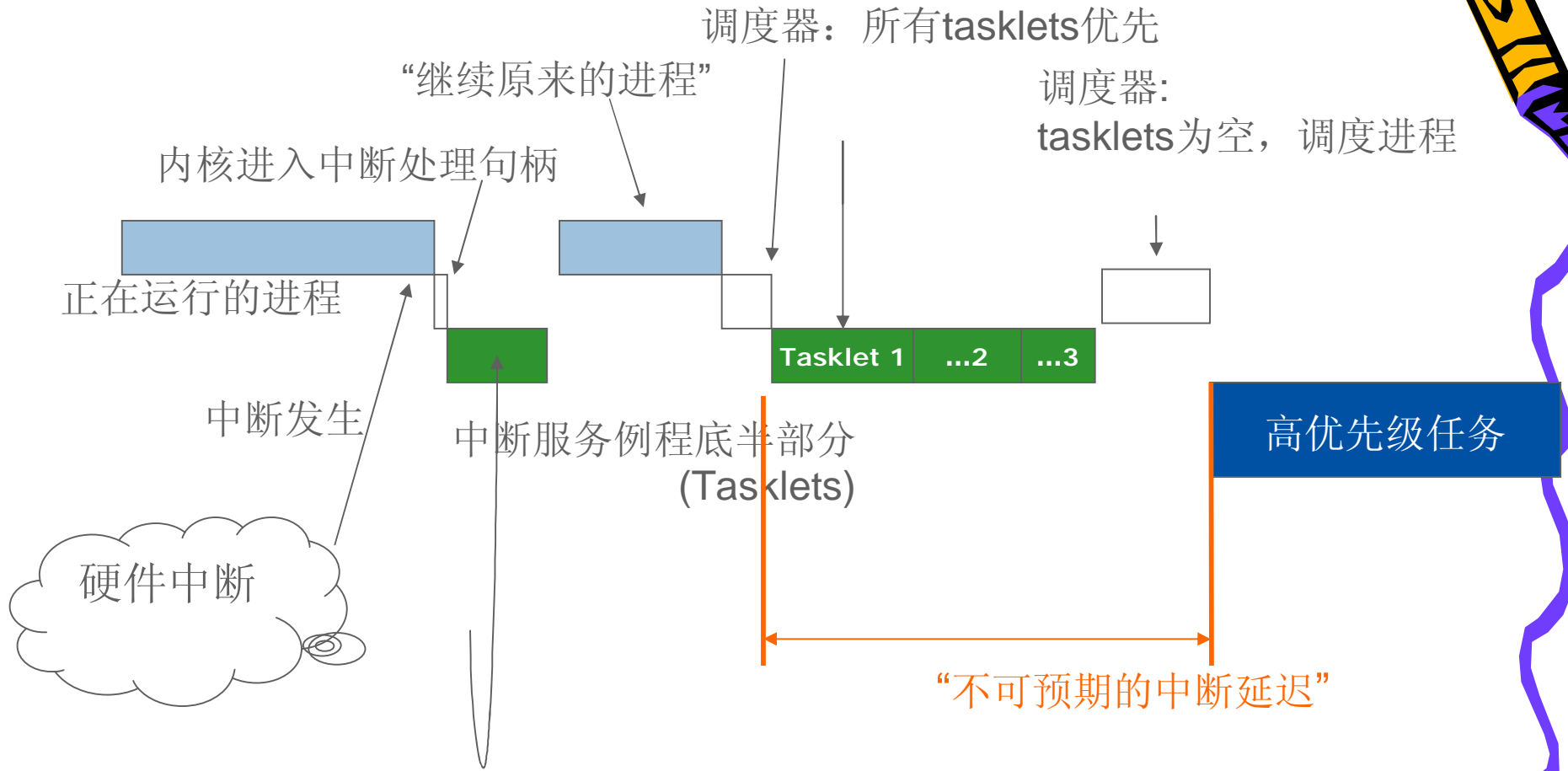
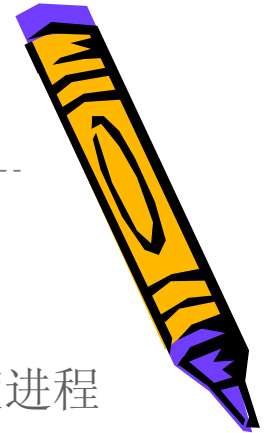
- 问题:
 - 有1000个进程在等待一个互斥锁
 - 互斥锁被解开 - **谁会先得到?**
 - 在非实时linux中, 第一个等待进程将得到该锁
 - 在实时linux中, 优先级最高的进程将被唤醒且得到该锁



**Real Time
is NOT fair,
remember?**

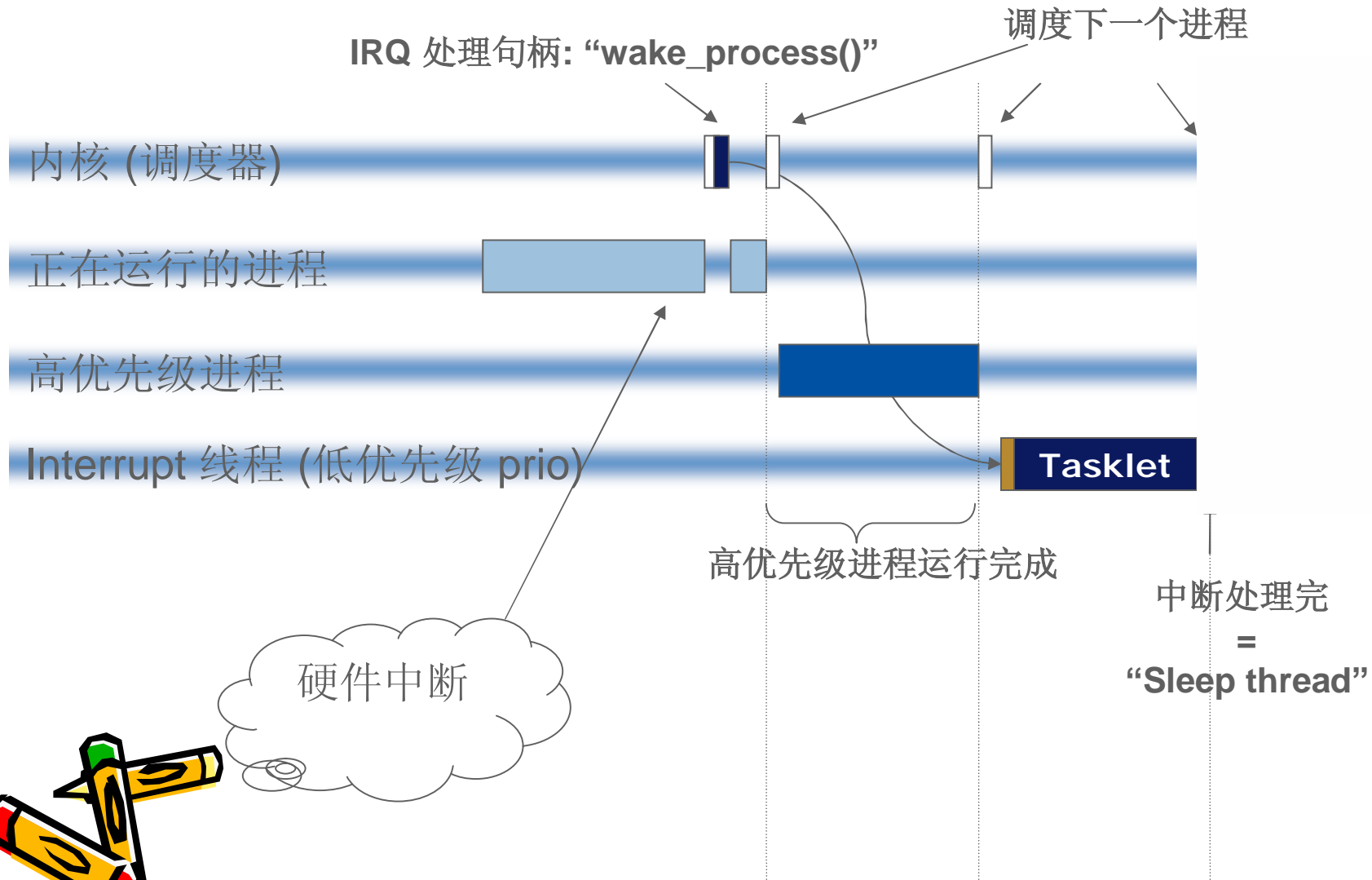
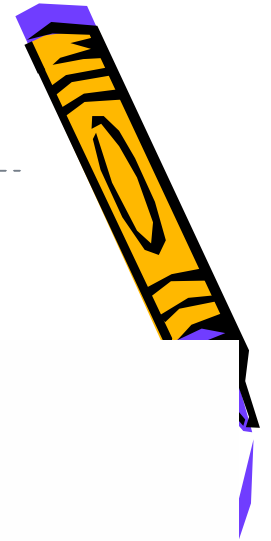


标准 IRQ 机制的问题

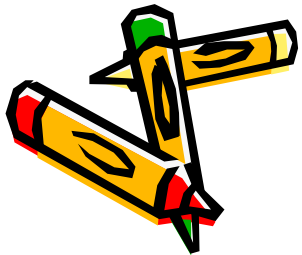
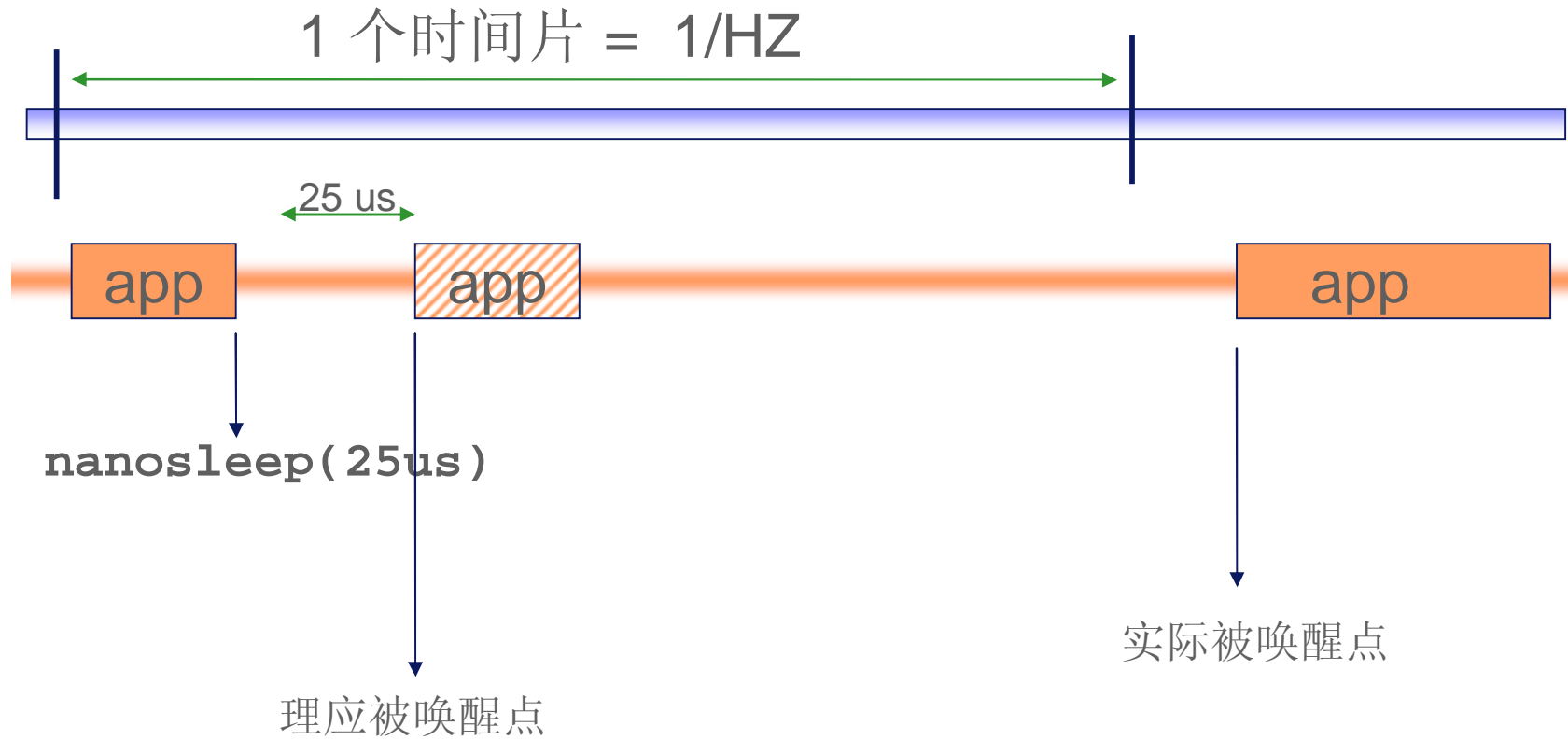


Solution:
“IRQs线程化”

RT-patch Thread Context Interrupt Handlers

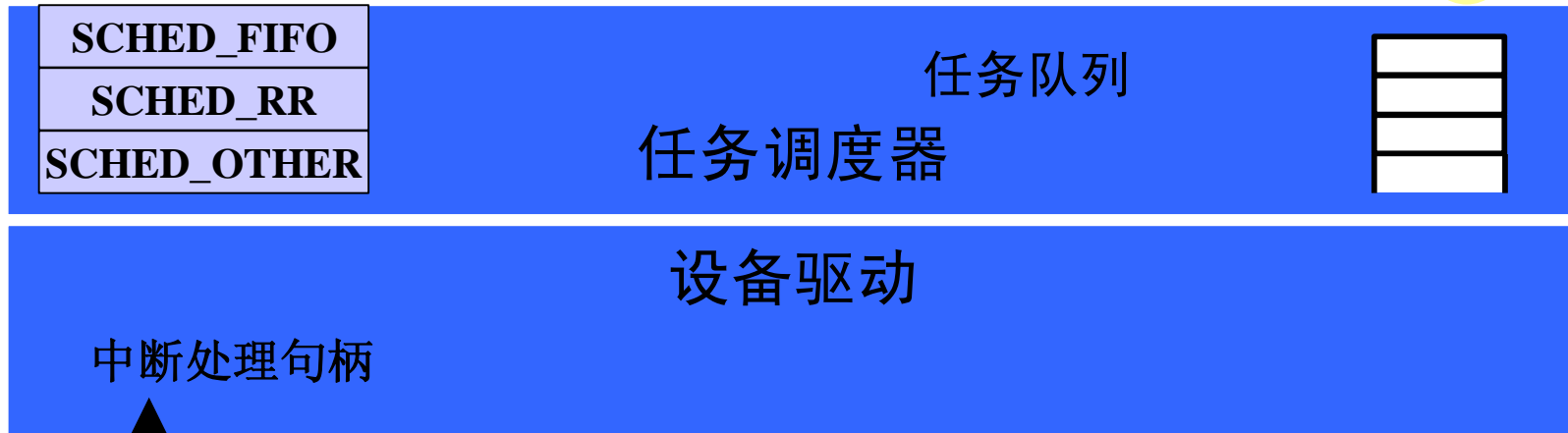
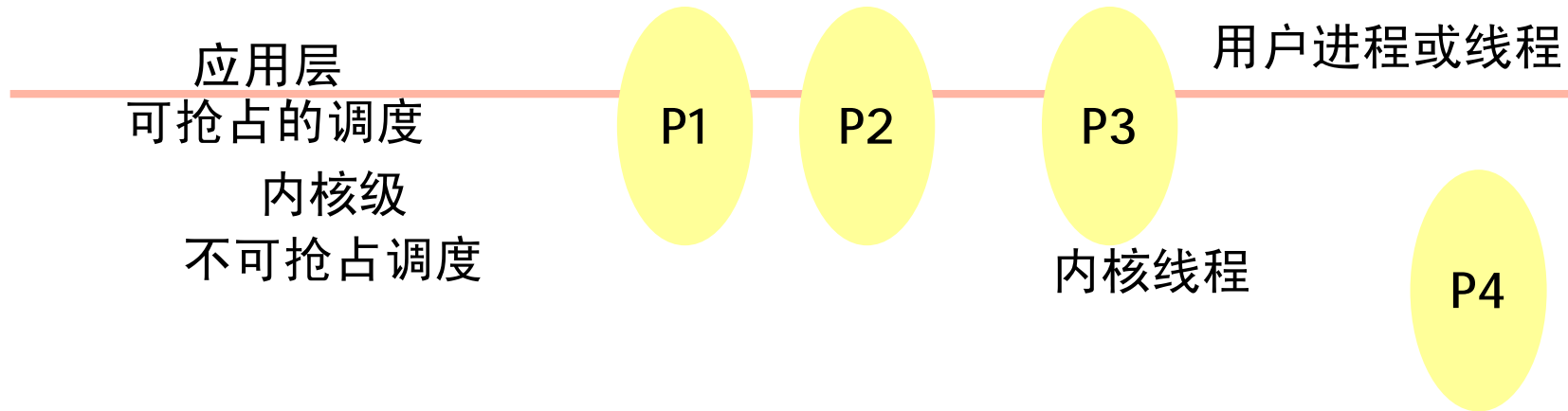


基于时间片的分时



Solution:
“Hi-res Timers”

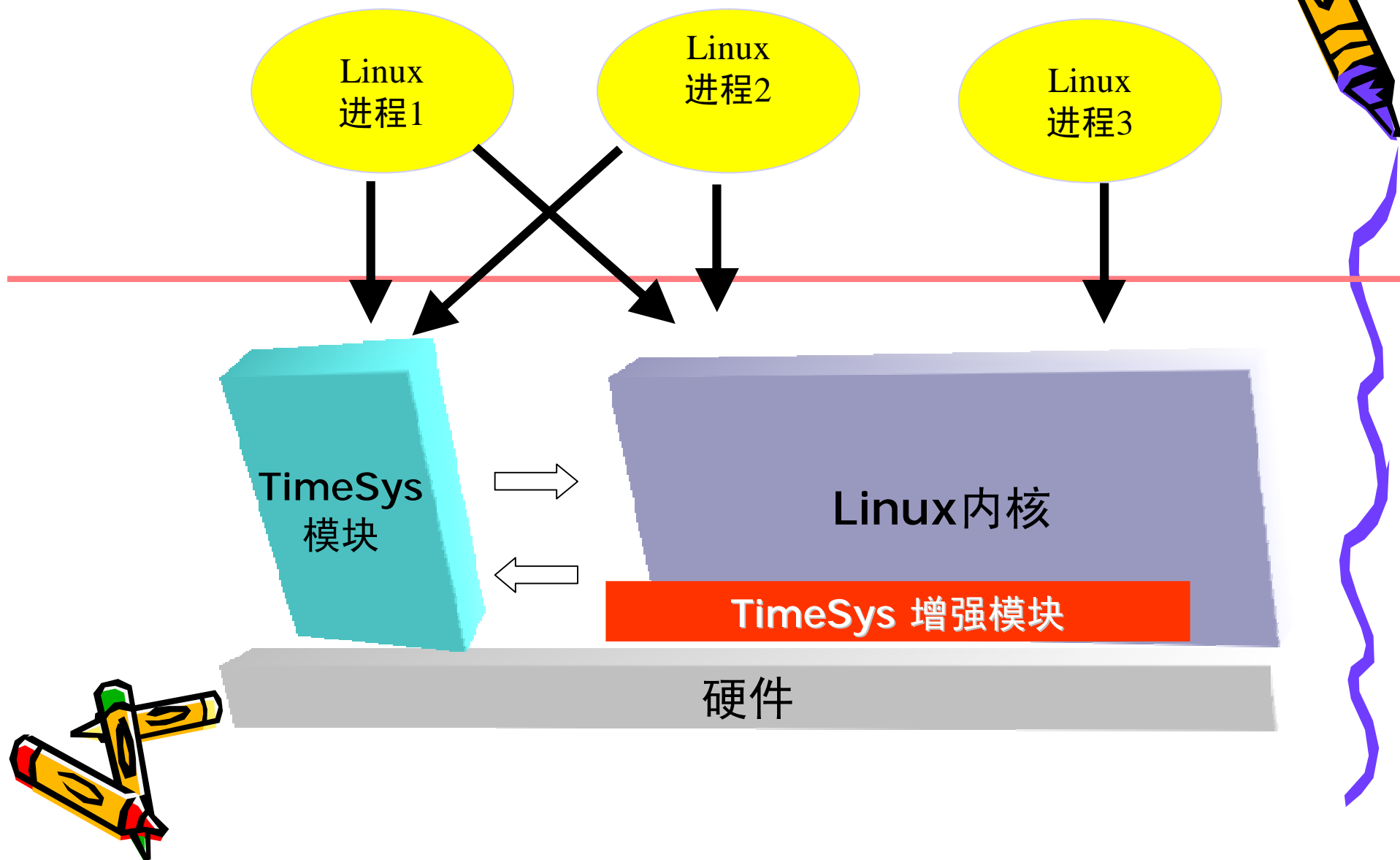
•POSIX 实时扩展



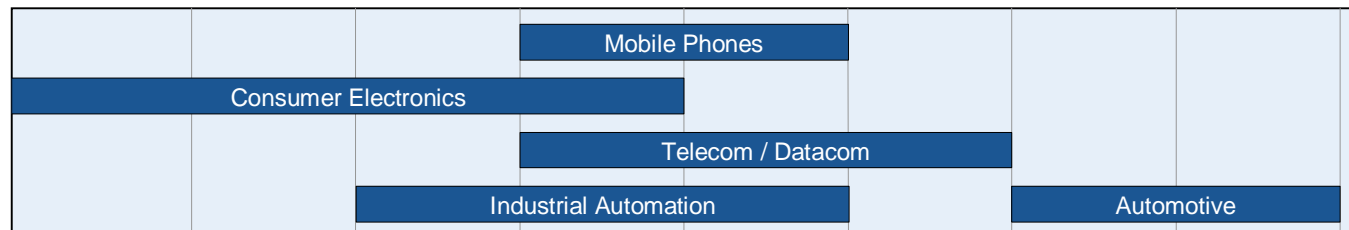
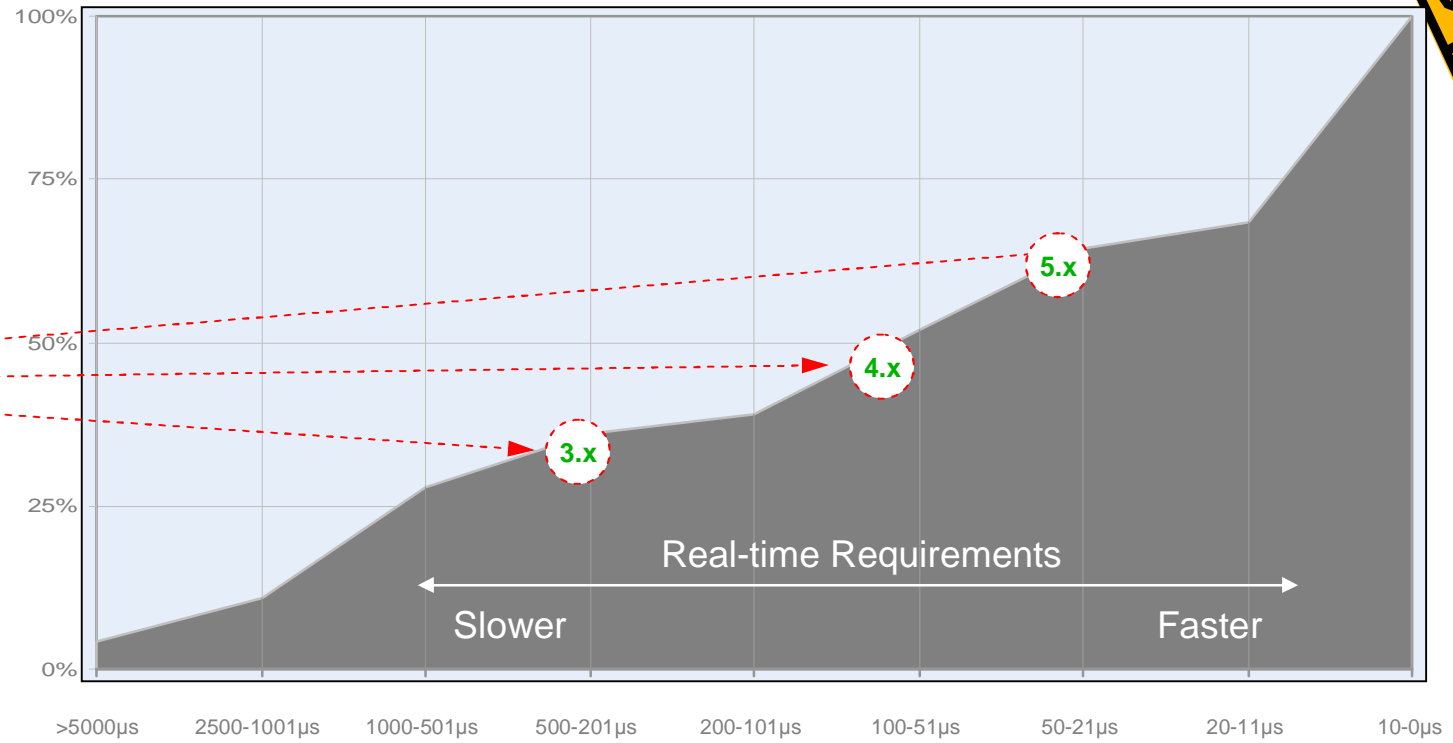
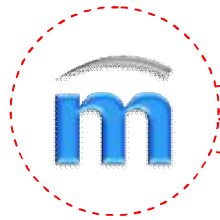
通过在RAM中增加物理页来增强性能



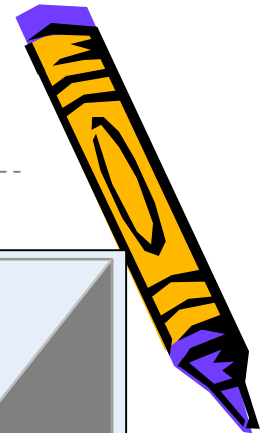
TimeSys Linux/RT



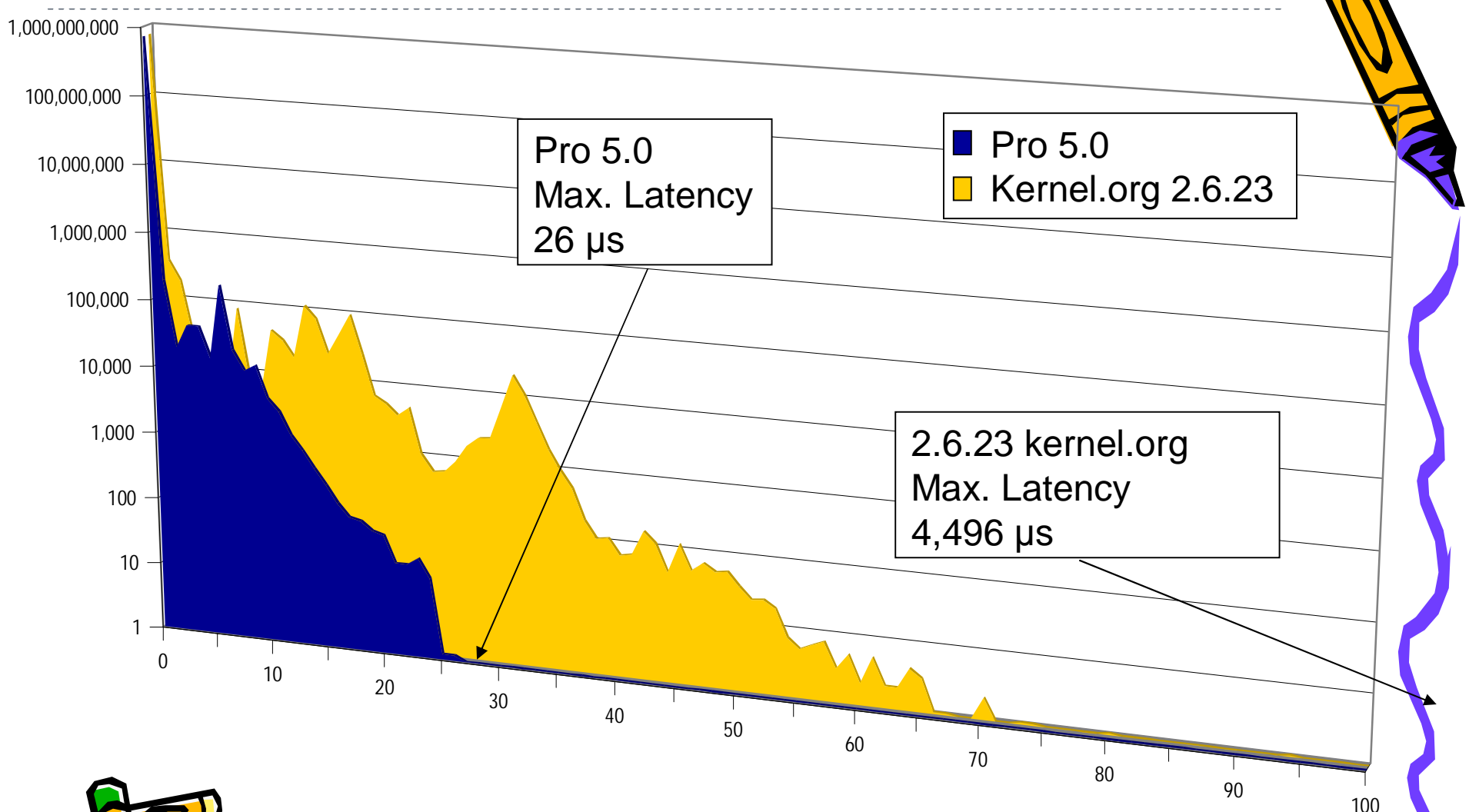
MontaVista 内核实时性



Sources: *The Embedded Software Strategic Market Intelligence Program*, VDC, July 2005 & *RTOSes Balance Performance with Ease of Use*, COTS Journal, November 2004



Quality and Integration

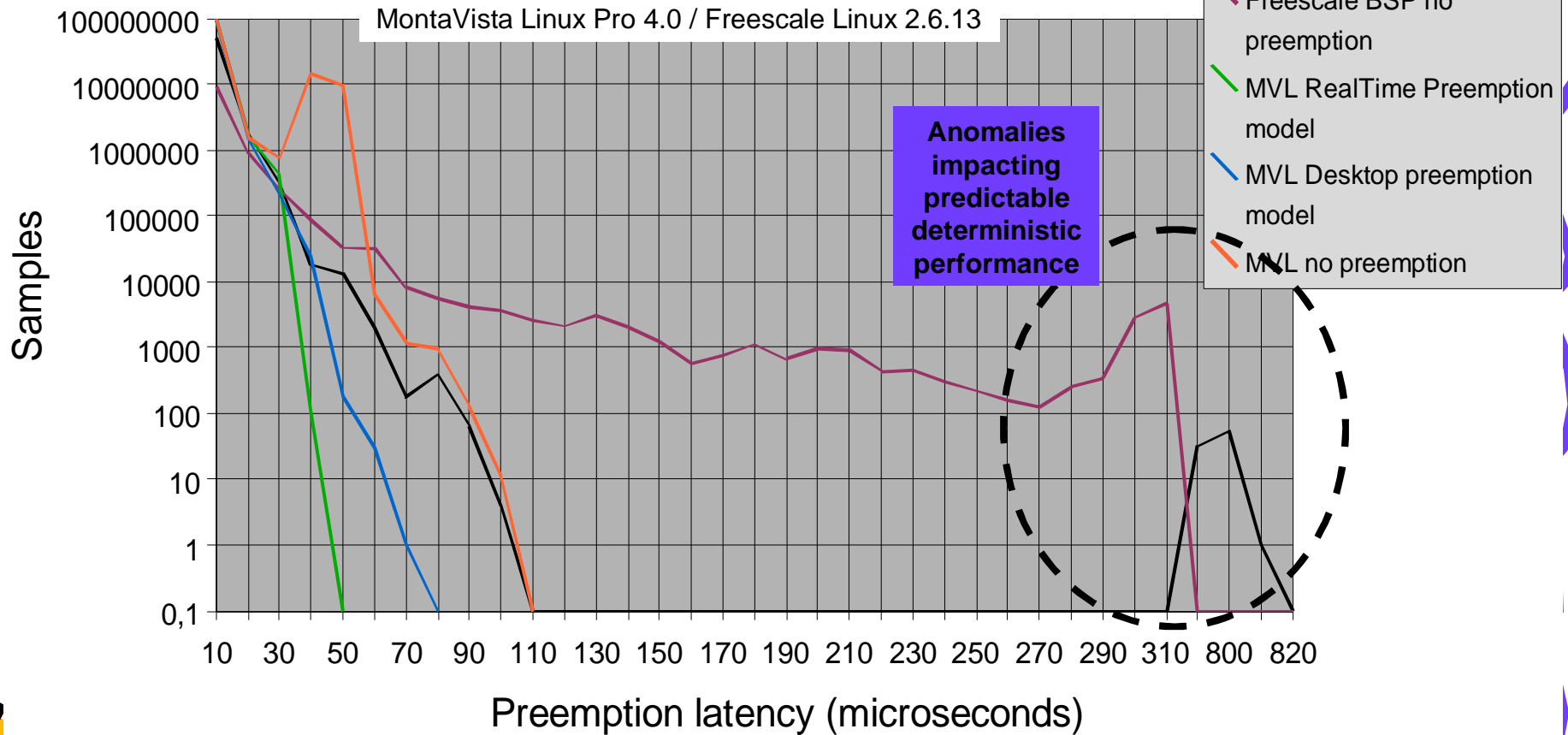


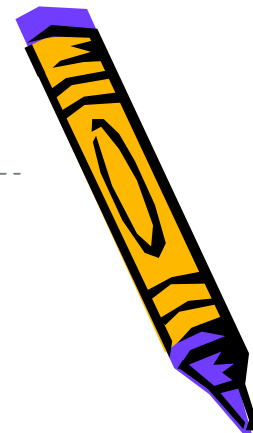
抢占延迟,专业版 5.0 and 2.6.23 kernel.org
(测试条件: 双核 Pentium M, 2.0 GHz, 加载满负荷负载数据)





MPC8349E-mI TX – Rio Grande Preemption latency





- UNIX = 公平
- 用户空间抢占
- 固定调度延迟--O(1) 调度器
- Robert Love的内核抢占
- Ingo Molnar自愿抢占
- RT补丁
- Paul McKenney的 RCU 改进
- Ingo Molnar新的CFS调度器
- Gregory Haskin的 优化调度器改进

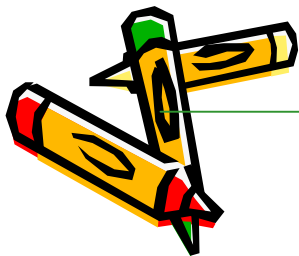


社区实时 Linux 情况 (总结)

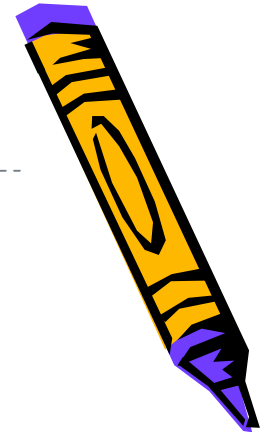


• Making Linux Real-time required addressing: Mainline?

- | | |
|--|-------------------|
| – Minimized interrupt disable times BKL still present !!! | Some |
| – Interrupt handling via schedulable threads Not acceptable to all drivers | NO |
| – Fully pre-emptable kernel Short critical sections | NO |
| – Perform synchronization via mutexes (not spin locks) Allows involuntary pre-emption | Partly |
| – Mutex support for priority inheritance / queueing | |
| – High Resolution timers | Yes |
| – Minimize number of mutexes / critical sections | Some |
| – Optimizing scheduler decisions New Scheduler (CFS) in mainline | Some |
| – RCU | Yes ₃₂ |



社区实时 Linux 情况 (总结)



Realtime Linux Road Map

This is the current status of Realtime Linux using the Realtime-Preempt patches:

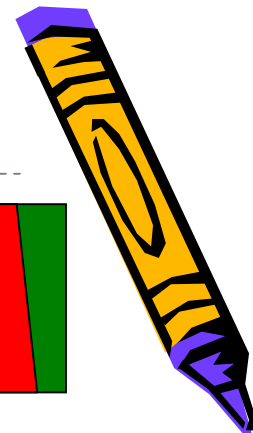
| Architecture | x86 | x86/64 | powerpc | arm | mips | 68knommu |
|----------------------------------|----------------|----------------|----------------|----------------|----------------|--------------------|
| Feature | | | | | | |
| Deterministic scheduler | ● | ● | ● | ● | ● | ● |
| Preemption Support | ● | ● | ● | ● | ● | ● |
| PI Mutexes | ● | ● | ● | ● | ● | ● ³ |
| High-Resolution Timer | ● | ● ¹ | ● ¹ | ● ¹ | ● ¹ | ● |
| Preemptive Read-Copy Update | ● ² | ● ² | ● ² | ● ² | ● ² | ● ² |
| IRQ Threads | ● ⁴ | ● ⁴ | ● ⁴ | ● ⁴ | ● ⁴ | ● ^{3,4,5} |
| Full Realtime Preemption Support | ● | ● | ● | ● | ● | ● ³ |

- Available in mainline Linux
- Available when Realtime-Preempt patches applied

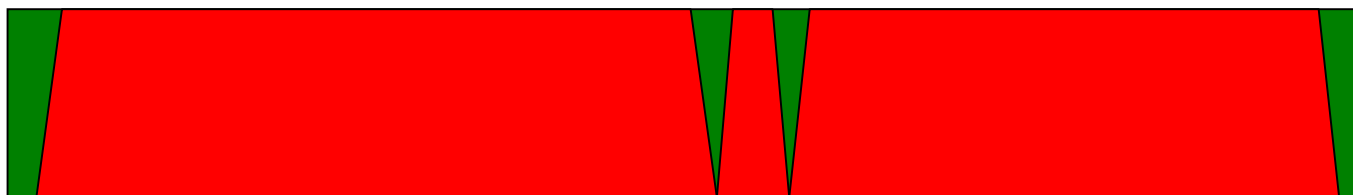
- 1) Since kernel 2.6.24 in mainline Linux
- 2) Since kernel 2.6.25 in mainline Linux
- 3) Realtime-Preempt patches 2.6.24.7-rt15 or higher required
- 4) Since kernel 2.6.30 in mainline Linux
- 5) Not yet adapted to generic interrupt code



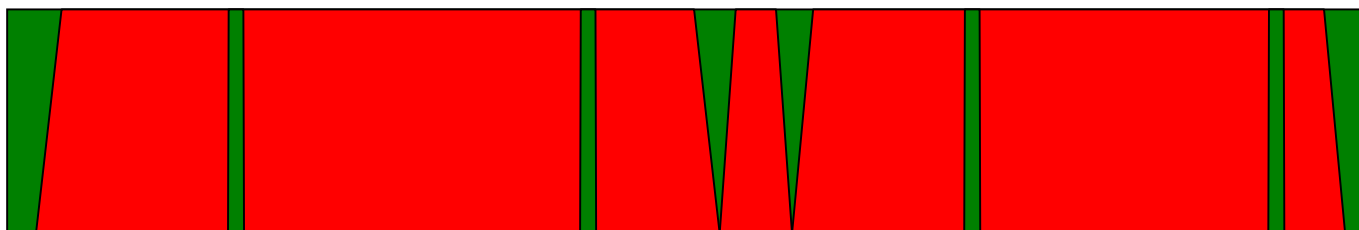
实时Linux抢占示意图



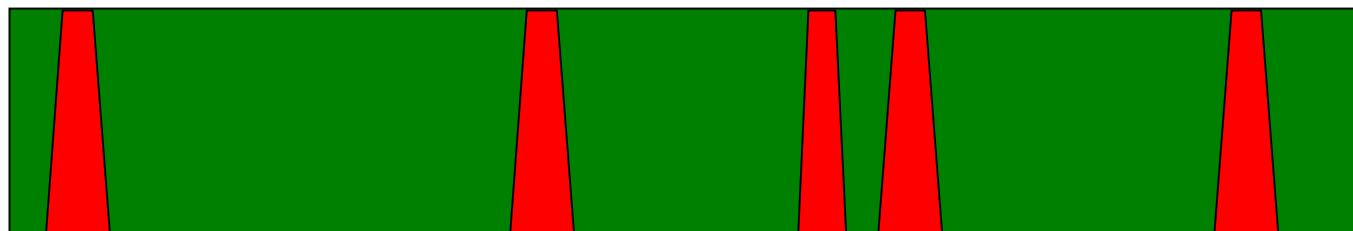
无抢占



自愿抢占



抢占内核



完全抢占



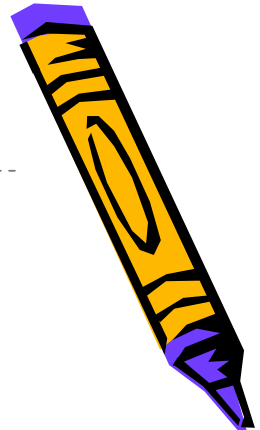
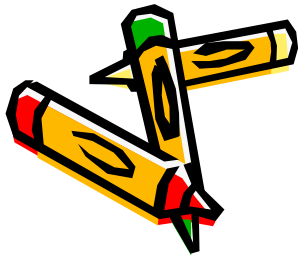
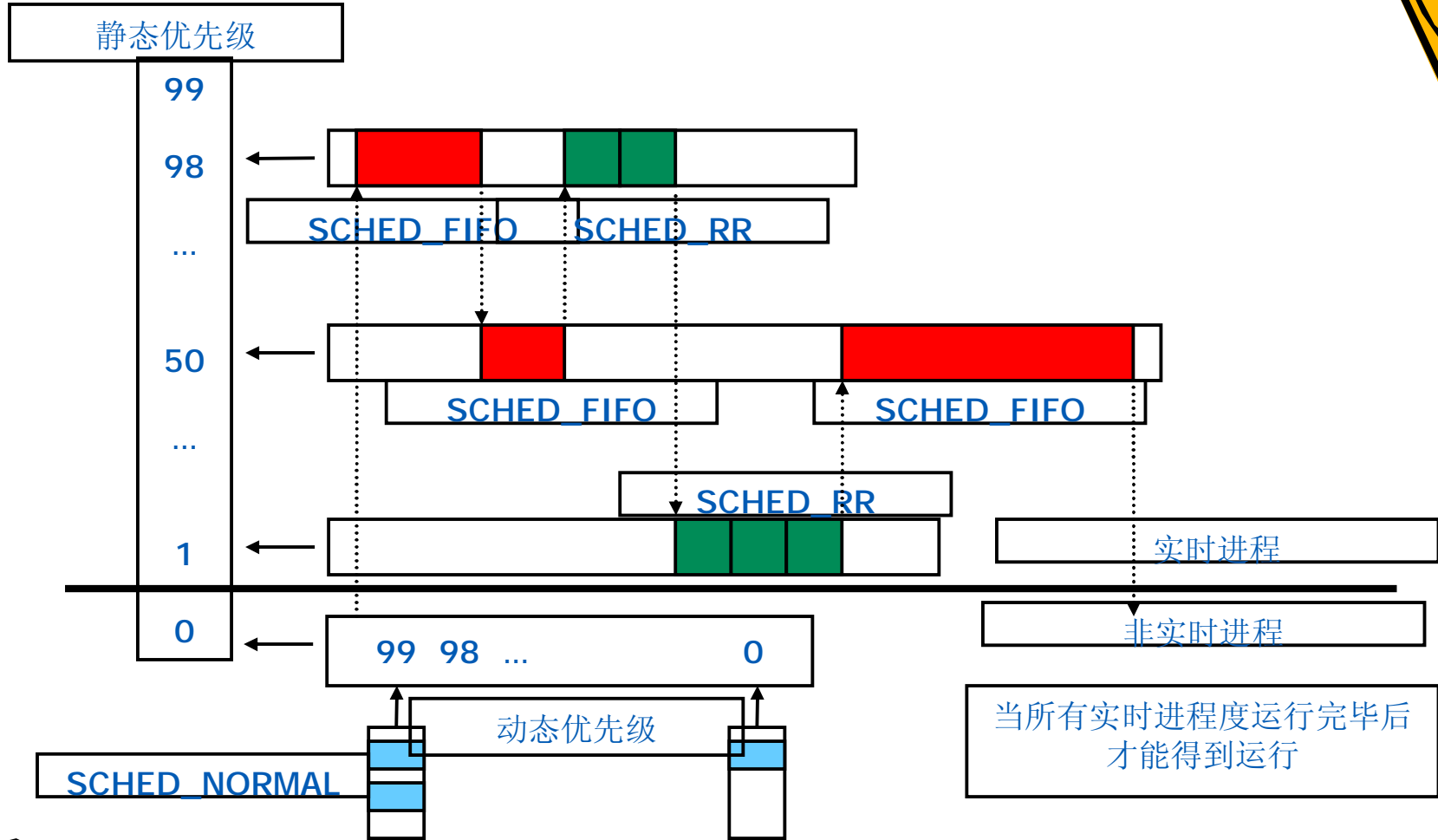
可抢占



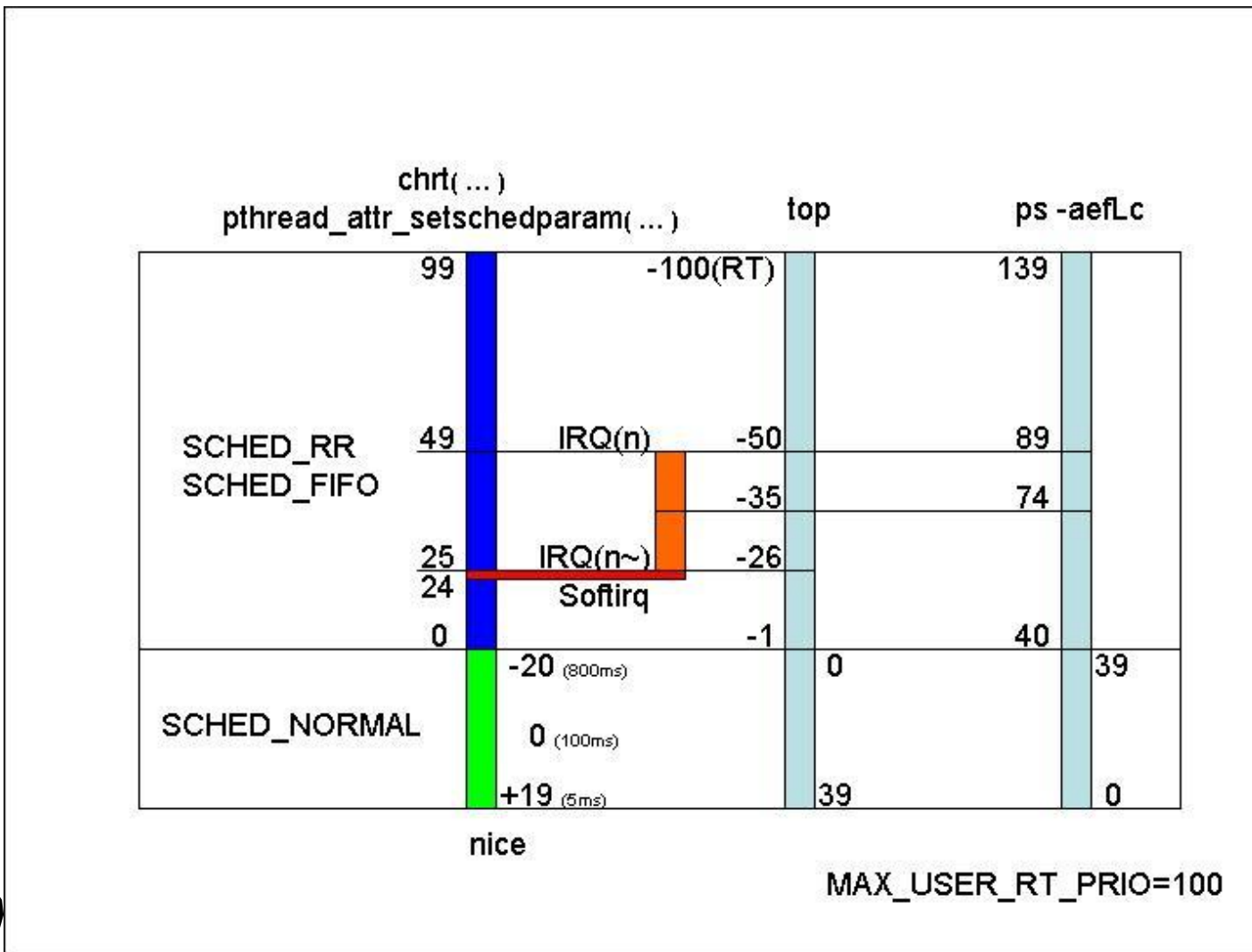
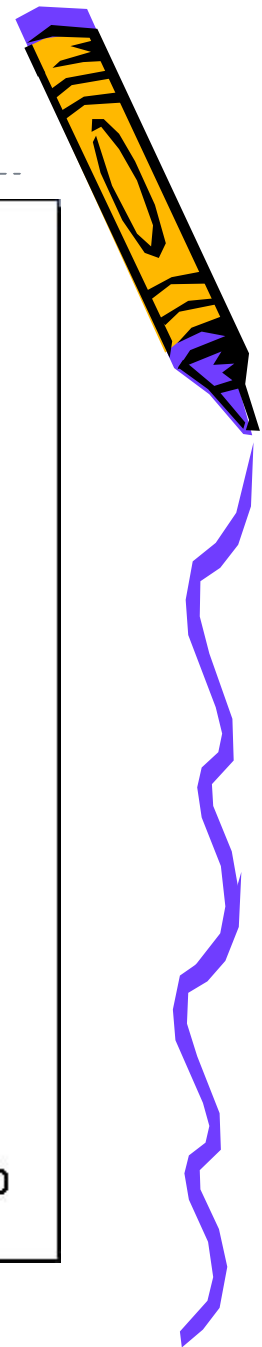
不可抢占



实时Linux优先级

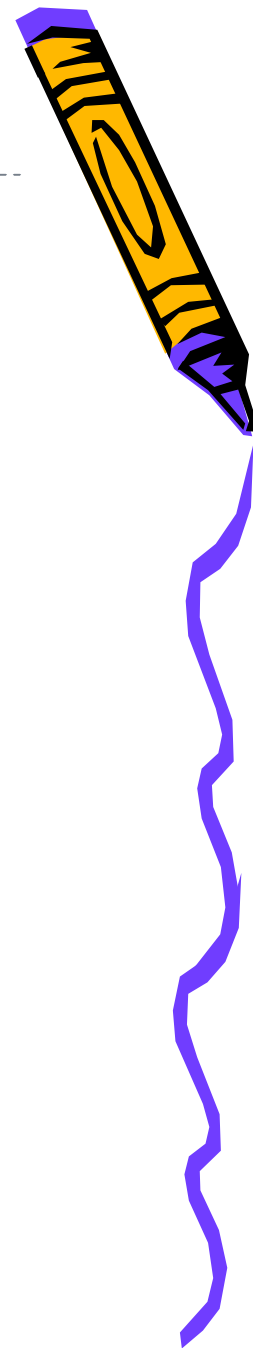


实时Linux优先级-分配情况



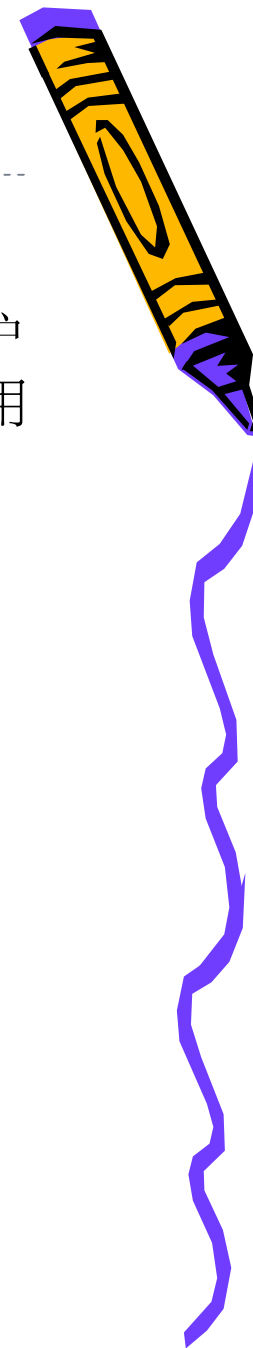
实时Linux- 内核驱动

- 代码的可移植性
- 在所有对时间不敏感的代码中，开发者应当更倾向于运用**high-level**锁的**API**来保证正确地转换到实时锁
- 在不可被抢占的临界区内，我们用被称为**raw_spin_lock**的自旋锁代替旧的不可抢占锁，该自旋锁的类型为**raw_spinlock_t**驱动的
 - 默认编译是完全可被抢占的



实时Linux- 内核驱动

- 内核进程管理
 - 在完全可抢占的实时内核中访问List是受Mutex保护的wake_up, interruptible_sleep 和相关函数不能用于不可中断中（在IRQ上下文【context】中）
 - 死锁检测并提出警告(调度是原子操作)
 - 非零preempt_count
 - 当中断被禁止时
 - 互斥锁深层嵌套时候不能用自选锁
- 通用内核代码之间共享锁
- 临界区管理

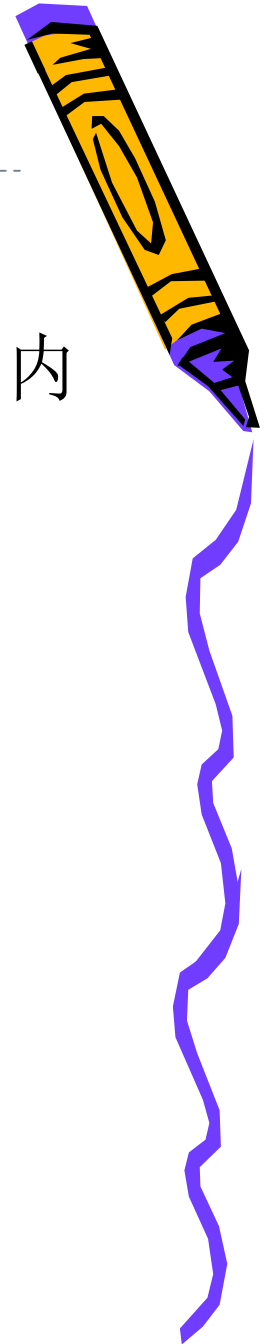




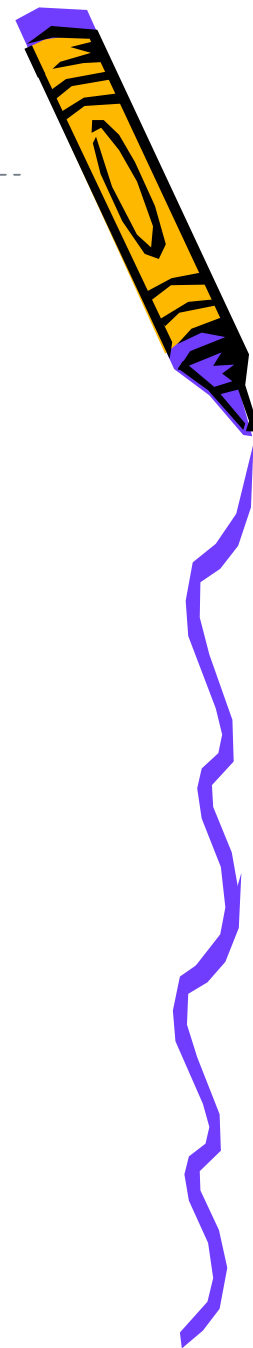
- 在实时可抢占模型中，除了定时器中断，一般中断处理都是创建一个内核线程，并将中断服务函数被指定到该线程上下文
 - 中断服务将被优先级化，且可被抢占
 - 一个新的IRQ标识位IRQF_NODELAY可被用来使你的设备中断按原来的方式运行 (IRQ上下文中【 context 】)
- 系统里面基本限制带IRQF_NODELAY标识位的中断功能函数
 - Wake_up() 在 IRQF_NODELAY中断中执行,需用 wake_up_process()代替.
 - 不可能再在中断上下文中给mutex上锁，需要用不可抢占的 raw_spinlock代替



- 在项目开发中，工程师需要深入理解的以下内容：
 - 内存使用和分配
 - 定时器和信号派发
 - 创建任务/设置策略



实时Linux应用编程(Cont')



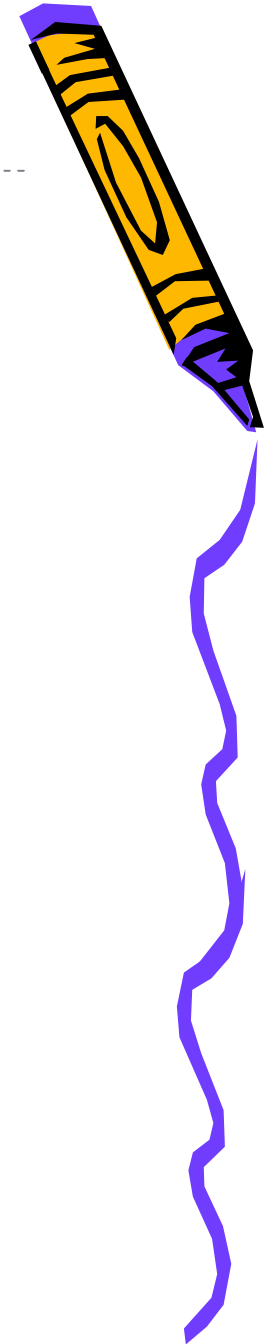
- 在使用PREEMPT_RT特性的实时系统中编程的基本流程：
 - 1 启动程序.
 - 2 分配所有应用程序需要用到的资源
 - 3 打开所有需要打开的文件，创建所有需要用到的通信机制（IPC,信号等）
 - 4 设置调度策略.
 - 5 设置优先级
 - 6 开始执行程序的实时处理

更多请参考：

http://rt.wiki.kernel.org/index.php/HOWTO:_Build_an_RT-application



实时Linux应用编程(Cont')

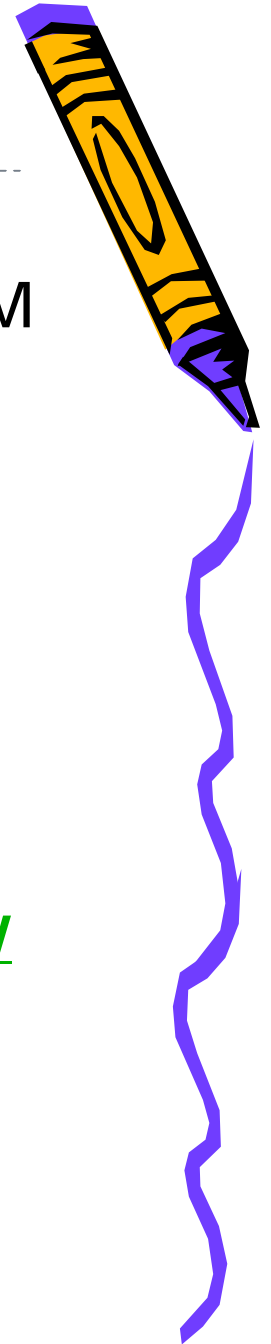


- 编写你自己的 **Realtime**程序

```
#include <sched.h>
struct sched_param sched_params;
/* Change our priority to as high as possible and use
   SCHED_FIFO */
sched_params.sched_priority =
    sched_get_priority_max(SCHED_FIFO);
if (0 != sched_setscheduler(getpid(), SCHED_FIFO,
    &sched_params)) {
    printf("sched_setscheduler failed (YOU MUST BE
        ROOT TO USE THIS)\n");
    exit(1);
}
```



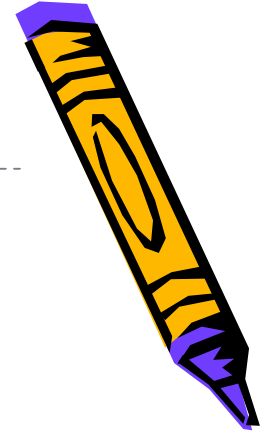
- LTP(Realtime Test Tree) (formerly IBM Test Cases)
- Cyclictest
- PI Mutex Test
- FTQ (Fixed Time Quanta Benchmark)
- Preemption Test
- low latency audio / scheduling latency tests/realtime audio
- ipbench





Q&A





谢谢!

