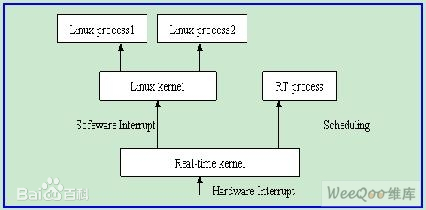
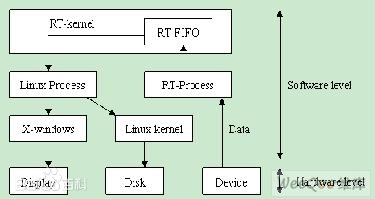
[RTlinux\_百度百科 (baidu.com)](https://baike.baidu.com/item/RTlinux/3458492)

在Linux 操作系统中，调度算法（基于最大吞吐量准则）、设备驱动、不可中断的系统调用、中断屏蔽以及虚拟内存的使用等因素，都会导致系统在时间上的不可预测性，决定了Linux操作系统不能处理硬实时任务。RTLinux为避免这些问题，在Linux内核与硬件之间增加了一个虚拟层（通常称作虚拟机），构筑了一个小的、时间上可预测的、与Linux内核分开的实时内核，使得在其中运行的实时进程满足硬实时性。并且RTLinux和Linux构成一个完备的整体，能够完成既包括实时部分又包括非实时部分的复杂任务。

RTLinux有两种中断：硬中断和软中断。软中断是常规Linux内核中断。它的优点在于可无限制地使用Linux内核调用。硬中断是安装实时Linux的前提。依赖于不同的系统，实时Linux下硬中断的延迟是15μs。





RTLinux的体系结构如图2所示。RTLinux的设计思想是:应用硬件的实时约束将实时程序分割成短小简单的部分,较大部分承担较复杂的任务。根据这一原则,将应用程序分为硬实时和软实时（即程序）2个部分。

硬实时的实现：

图3所示的是RTFIFO结构图。RTLinux将标准Linux内核作为简单实时操作系统（RTOS）（或叫子内核）里优先权最低的线程来运行，从而避开了Linux内核性能的问题。 从图3可以看出，RTLinux拥有两个内核。这就意味着有两组单独的API，一个用于Linux环境，另一个用于实时环境。此外，为保证实时进程与非实时Linux进程不顺序进行数据交换，RTLinux引入了RT-FIFO队列。RT-FIFO被Linux视为字符设备，最多可达150个，分别命名为/dev/rtf0、/dev/rtf1……/dev/rtf63。最大的RT-FIFO数量在系统内核编译时设定。

RTLinux程序运行于用户空间和内核态两个空间。RTLinux提供了应用程序接口。借助这些API函数将实时处理部分编写成内核模块，并装载到RTLinux内核中，运行于RTLinux的内核态。非实时部分的应用程序则在Linux下的用户空间中执行。这样可以发挥Linux对网络和数据库的强大支持功能。

软实时的实现:

RTLinux通过一个高效的、可抢先的实时调度核心来全面接管中断，并把Linux作为此实时核心的一个优先级最低的进程运行。当有实时任务需要处理时，RTLinux运行实时任务；无实时任务时，RTLinux运行Linux的非实时进程。其系统结构见图4。

RTLinux在默认的情况下采用优先级的调度策略，即系统调度器根据各个实时任务的优先级来确定执行的先后次序。优先级高的先执行，优先级低的后执行，这样就保证了实时进程的迅速调度。同时RTLinux也支持其它的调度策略，如最短时限最先调度（EDP）、确定周期调度（RM）（周期段的实时任务具有高的优先级）。RTLinux将任务调度器本身设计成一个可装载的内核模块，用户可以根据自己的实际需要，编写适合自己的调度算法。

对于一个操作系统而言，精确的定时机制虽然可以提高任务调度器的效率，但会增加CPU处理定时中断的时间开销。RTLinux对时间精度和时钟中断处理的时间开销进行了折中考虑。不是像Linux那样将8254定时器设计成10ms产生一次定时中断的固定模式，而是将定时器芯片设置为终端计时中断方式。根据最近的进程的时间需要，不断调整定时器的定时间隔。这样不仅可以获得高定时精度，同时中断处理的开销又最小。

a.中断仿真

在中断控制硬件与LINUX核心之间放置一个软件仿真层。具体做法是，在LINUX源码中出现cli、sti和iret的所有地方都用仿真宏：S\_CLI、S\_STI和S\_IRET来替换。所有的硬件中断就都被仿真器所截获。

当需要关中断时，就将仿真器中的一个变量置0。不论何时若有中断发生，仿真器就检查这个变量。如果是1（LINUX已开中断），就立即调用LINUX的中断处理程序；否则，LINUX中断被禁止，中断处理程序不会被调用，而是在保存着所有挂起中断的信息的变量的相应位置1。当LINUX重新开中断，所有挂起中断的处理程序都会被执行。这种仿真方式可以称之为"软中断"。

b.实时任务

实时任务是在一个由核心控制的调度程序的调度下执行的用户定义的程序。

RT-LINUX最初将实时任务设计成ELF格式的目标文件。这一设计方案的最大缺点就是性能比较差。原因在于，第一，486的缓存是虚拟的。所以每当页表目录的基址寄存器改变时，TLB（转换后备缓冲器）就会失效。由于实时任务的上下文转换频繁，所以TLB的频繁失效就导致系统性能的严重下降。第二，486的保护级别变换耗时不少。比如，陷入更高级别时需要71个循环，而其它指令一般少于10个循环。

解决的办法就是使用可加载模组技术，所有的实时任务都同处于一个地址空间-内核地址空间，不仅避免了频繁的TLB失效，同时也消除了变换保护级别的消耗，而且任务转换也变得相当容易。

c.进程调度

实时系统的进程调度的主要任务就是满足实时任务在时间上的要求。调度算法的种类很多，没有一个策略是放之四海而皆准的，因此采用哪种算法要取决于具体应用。

RT-LINUX采用的方法是允许用户编写自己的调度程序，并可以编译成模组的形式。这样就可以方便地试验不同的策略和算法对于某一特定应用的适合性，从中选出最优。

RT-LINUX自带的是一个基于优先数的抢占式调度程序。此调度程序将LINUX当作具有最低优先数的实时任务。因此，LINUX只在实时系统无任何实时任务是才运行。在从LINUX切换到实时任务时，系统记下软中断的状态并禁止软中断。在切换回来实，再恢复软中断的状态。

d.时钟

调度程序需要精确的时钟才能准确操作。调度通常是在特定的时刻进行任务切换。时钟的偏差会引起预定调度的偏差，导致产生被称为任务发布抖动的现象。这是一种应该尽量避免的不良现象。

RT-LINUX的解决办法是，将IBM PC兼容机中的时钟芯片Intel 8254设置为中断开启终端计数模式。在这种模式下，精度可以达到1毫秒。这样在降低中断处理的影响的同时，获得了较高的时钟精度。

e.IPC

由于标准LINUX核心可以被实时任务在任意时刻抢占，所以实时任务无法安全地调用LINUX的程序。但是总要有一个信息交换的机制。

在RT-LINUX中所用的信息交换方式是RT-FIFO（实时队列）。它与UNIX的管道非常相似，都是一个无结构的数据流。通过RT-FIFO，LINUX的进程之间，实时进程之间，以及LINUX的核心与实时进程之间可以交换信息。

对于一个普通的进程来说，RT-FIFO就是一个特殊的字符文件。这些文件必须自建：

# for i in 0 1 2 3; do mknod /dev/rtf$i c 63 $i; done