## 时钟和定时器电路：

1. 内核必须完成两种定时测量：
2. 保存当前时间和日期
3. 维持定时器，这种机制告诉内核或用户程序，某一时间间隔已经过去了。
4. Linux给PC的第一个PIT（可编程间隔定时器）进行编程，使它大约1000Hz的频率向IRQ0发出时钟中断，即1ms产生一次时钟中断。这个时间间隔叫做一个节拍（tick）
5. 可编程间隔定时器（PIT）：PIT的作用提醒内核时间间隔已经过了。这个设备是通过发出一个特殊的中断，叫时钟中断来通知内核又一个时钟间隔过去了。PIT永远以内核确定的固定频率不停地发出中断。
6. 实时时钟：所有的PC都包含一个叫实时时钟（RTC）的时钟，它是独立于CPU和所有其他芯片的。即使当PC被切断电源，RTC还在继续工作。因为它靠一个小电池或蓄电电池供电。RTC能在IRQ8上发出周期性的中断，可以对RTC进行编程以使RTC到达某个特定的值时激活TRQ8线，也就是作为一个闹钟来工作。
7. 时间戳计数器：处理器中包含一个计数器，它在每个时钟信号到来时加1。该计数器利用64位的时间戳计数器（TSC）寄存器来实现的，可以通过汇编语言指令rdtsc读这个寄存器。当使用这个寄存器时，内核必须考虑时钟信号的频率：例如，如果时钟节拍的频率是1GHz，那么，时间戳计数器每纳秒增加一次。算出CPU实际频率的任务是在系统初始化期间完成的。
8. 节拍：产生一次时钟中断的时间间隔。
9. CPU本地定时器：
10. 概念：CPU本地定时器是一种能够产生单步中断或周期性中断的设备，它类似于可编程间隔定时器，不过，还是有几点区别：
11. 可以对本地定时器编程来产生很低频率的中断。
12. 本地APIC定时器把中断只发送给自己的处理器，而PIT产生一个全局性中断，系统中的任一CPU都可以对其处理。
13. APIC定时器是基于总线时钟信号的。每隔1，2，4，8，16，32，64或128总线时钟信号到来时对该定时器进行递减可以实现对其编程的目的。相反，PIT有其自己的内部时钟振荡器，可以更灵活编程。

## Linux计时体系结构：

1. 概念：Linux的计时体系结构是一组与时间流相关的内核数据和函数。
2. （1）在单处理器系统上，所有的计时活动都是有全局定时器（可以使可编程间隔定时器也可以是高精度事件定时器）产生的中断触发的。
3. 在多处理器系统上，所有普通的活动（像软定时器的处理）都是由全局定时器产生的中断触发的，而具体CPU的活动（像监控当前运行进程的执行时间）是由本地APIC定时器产生的中断触发的。

3、、计时体系结构的数据结构：

3.1、定时器对象：

（1）概念：为了使用一种统一的方法来处理可能存在的定时器资源，内核使用了“定时器对象”，它是timer\_opts类型的一个描述符。

（2）1个字段和4个标准的方法：

name ：标识定时器源的一个字符串

mark\_offset ：记录上一个节拍的准确时间，由时钟中断处理器程序调用

get\_offset ： 返回自上一个节拍开始所经过的时间

monotonic\_clock ：返回自内核初始化开始所经过的纳米数

delay : 等待指定数目的“循环”

（3）定时器对象中最重要的方法是mark\_offset和get\_offset。mark\_offset方法由时钟中断处理器程序调用，并以适当的数据结构记录每个节拍到来时的准确时间。Get\_offset方法使用已记录的值来计算自上一次时钟中断以来（节拍）经过的时间（以us为单位）。由于这两种方法，使用Linux计时体系结构能够达到子节拍的分辨度，也就是说内核能够以比节拍周期更高的精度来测定当前的时间。这种操作被称为“定时插补”。

3.2、变量cur\_timer存放了某个定时器对象的地址。最初，cur\_time指向timer\_none，这个timer\_none是一个虚拟的定时器资源对象，内核在初始化的时候使用它。在内核初始化期间，select\_timer()函数设置cur\_time指向适当定时器对象的地址。

3.3、jiffies变量

（1）jiffies变量是一个计数器，用来记录自系统启动以来产生的节拍总数。每次时钟中断发生时，（每个节拍）它便加1。Jiffies是一个32位的变量，因此每隔大约50天它的值会回绕到0，这对Linux服务器来说是一个相对较短的时间间隔。不过，由于试用了time\_after、time\_after\_eq、time\_before和time\_before\_eq四个宏，内核能够处理jiffies变量的溢出。

（2）jiffies变量通过连接器被换算成64位计数器的低32位，这个64位计数器被称作jiffies\_64。++jiffies\_64操作同时也会增加32位的jiffies变量的值，因为后者对应jiffies\_64的低32位。

3.4、xtime变量：

（1）xtime变量存放当前时间和日期，它是一个timespec类型的数据结构，该结构有两个字段：

tv\_sec：存放自1970年1月1日午夜以来经过的秒数

tv\_nsec：存放自上一秒开始经过的纳秒数

（2）xtime变量通常是每个节拍更新一次，大约每秒更新1000次。用户程序从xtime变量获得当前时间和日期。内核也经常引用它。

（3）xtime\_lock顺序锁消除了对xtime变量的同时访问而可能发生的竞态条件。xtime\_lock同样也保护jiffies\_64变量。一般而言，这个顺序锁用来定义计时体系结构中的一些临界区。

## 单处理器系统上的计时体系结构：

1. 在单处理器系统上，所有与定时有关的活动都是由IRQ线0上的可编程间隔定时器产生的中断触发的。
2. 在内核初始化期间，time\_init()函数被调用来建立及时提醒。

## 多处理器系统上的计时体系结构：

1. 多处理系统可以依赖两种不同的时钟中断源：可编程间隔定时器或高精度事件定时器产生的中断，以及CPU本地定时器产生的中断。
2. 全局时钟中断处理程序由time\_init()初始化。
3. 所有本地APIC定时器都是同步的，因为它们都基于公共总线时钟信号。函数calibrate\_APIC\_clock()通过正在启动的CPU本地APIC来计算在一个节拍内受到了多少个总线时钟信号。这个函数计算出来的值对系统中其他CPU同样有效。
4. 本地时钟中断处理程序：
5. 该处理程序执行系统中与特性CPU相关的计时活动，即监管内核代码并检测当前进程在特定CPU上已经运行了多长时间。
6. 更新时间和日期：
7. 内核必须周期性地更新该变量该变量，才能使它的值相当的精确。

## 更新系统统计数：

1. 更新本地CPU统计数：
2. 记录系统负载：
3. 任何Unix内核都要记录系统进行了多少CPU活动。这些统计数据由各种管理使用程序来使用。用户输入uptime命令后可以看到一些统计数据：如相对于最后1分种、5分钟、15分钟的 “平均负载”。在单处理器系统上，值0意味着没有活跃的进程在运行，而值1意味着一个单独的进程100%占有CPU，值大于1说明几个运行着的进程共享CPU。
4. 监管内核代码：
5. Linux包含一个被称作readprofiler的最低要求的代码监管器，Linux开发者用其发现内核在内核态的什么地方话费时间。监管器确定内核的“热点”——执行最频繁的内核代码片段。
6. 检查非屏蔽中断（NMI）监视器：
7. 看门狗系统：探测引起系统冻结的内核bug。
8. 看门狗基于本地和I/O APIC一个巧妙的硬件特性：它们能在每个CPU上产生周期性的NMI中断。
9. 原理：一旦每个时钟节拍到来，所有的CPU，不管其正在做什么，都开始执行NMI中断处理程序，该中断处理程序又调用do\_nmi()。这个函数获得CPU的逻辑号n，然后检查irq\_stat数组第n项的apic\_timer\_irqs字段。如果该字段工作正常，那么第n项的值必定不同于在前一个NMI中断中读出的值。当CPU正常运行时，第n项的apic\_timer\_irq字段就会被本地时钟中断处理程序增加。如果计数器没有被增加，说明本地时钟中断处理程序在整个时钟节拍期间根本就被执行。这个CPU被冻结了。 当NMI中断处理程序检测到一个CPU冻结时，就会敲响所有的钟：它把引起恐慌的信息记录在系统日志文件中，转存该CPU寄存器和内核栈的内容，最后杀死当前进程。
10. 软定时器和延迟函数：
11. 实现一个定时器并不难，每个定时器都包含有一个字段，表示定时器将需要多长时间才到期。这个字段的初始值就是jiffies的当前值加上适合的节拍数。这个字段的值不再改变。每当内核检查定时器时，就把这个到期字段值和当前这一刻jiffies的值相比较，当jiffies大于或等于这个字段存放的值时，定时器到期。超时表示与定时器相关的时间间隔已经用完的那个时刻。
12. Linux使用两种类型的定时器，动态定时器和间隔定时器。第一种由内核使用，而间隔定时器可以由进程在用户态创建。
13. 这里是有关Linux定时器的警告：因为对定时器函数的检查总是由可延迟函数进行，而可延迟函数被激活以后很长时间才能被执行，因此，内核不能确保定时器函数正好在定时器到期时开始执行，而只能确保在适当的时间执行它们，或者假定延迟到几百毫秒之后执行它们。因此，对于必须严格遵守定时时间的那些实时应用而言，定时器并不适合。