第17章　系统内核机制

本章介绍Linux系统内核提供的机制，以便系统内核中的各个部分可以有效地协作。

17.1　Bottom Half处理

有时在系统内核中，你也许不想处理任何任务。例如在一个中断处理过程中，当发生一个中断时，处理器停止正在处理的工作，同时操作系统将中断发送到相应的设备驱动程序。但设备驱动程序不能花费太多的时间来处理中断，因为在这期间系统将不能做任何的工作，所以一些工作可以以后再进行处理。bottom half处理程序正是用来实现此功能的。图17-1显示了内核中有关bottom half处理的数据结构。

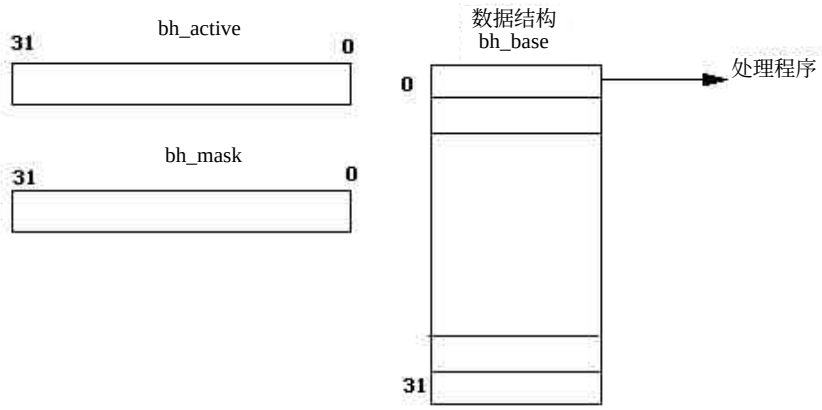

图17-1　Bottom Half 数据结构示意图

系统内核中可以有多达32个不同的bottom half处理程序。bh_base中保存着指向每一个bottom half处理程序的指针。bh_active和bh_mask根据安装和活动状态进行设置。如果bh_active的第N位置位，那么bh_base中的第N个元素将包含bottom half处理程序的地址。如果bh_active的第N位置位，那么第N个bottom half处理程序就可以在调度程序认为合适的时候调用它。一般情况下，bottom half处理程序都有一个和它们关联的任务列表。

一些内核的bottom half处理程序和设备有关，但以下几个却较为通用：

1.timer

当系统中的周期计时器中断时，此处理程序将标记为活动状态。它用来驱动内核中的计时器队列机制。

2. console

用于处理有关控制台的信息。

3. tqueue

用于处理有关tty 信息。

4. net

用于处理一般的网络问题。

5. immediate

这是一个通用的处理程序，可以被多个设备驱动同时使用。

当设备驱动程序或内核的其他部分需要把一些工作等待以后完成时，它将会将工作添加到相应的系统队列中，例如timer 队列，然后通知内核需要做bottom half处理。它通过将bh_active中适当的位置 1而通知内核。如果驱动程序将一些工作放入了 immediate队列中，它将会把bh_active的第8位置1，这样immediate bottom half处理程序就可以运行和处理它了。在每一次系统调用结束之前，系统内核都将会查看 bh_active。如果有任何的一位被置位，内核将会调用相应的bottom half处理程序。检查的顺序是先检查位0，然后是位1，以此类推直到位31。

17.2 任务队列

任务队列是系统内核将任务推迟到以后再做的方法。 Linux系统有一个机制可以把任务放入到队列中等待以后处理。

任务队列通常和bottom half处理程序一同使用，例如当timer bottom halfe处理程序运行时，系统将处理计时器任务队列。一个任务队列就是一个简单的数据结构，如图 17-2所示。它是一个由tq_struct数据结构组成的链表，每一个数据结构都包含处理程序的地址和指向相关数据的指针。

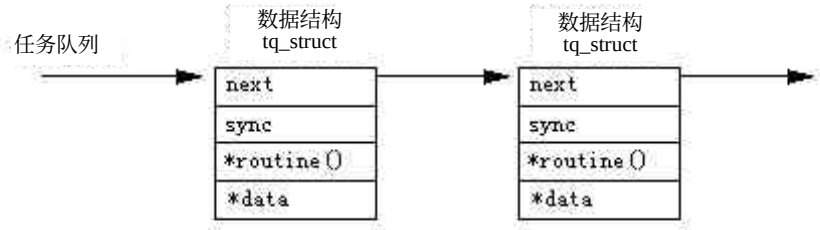

图17-2 任务队列示意图

当系统处理任务队列中的任务时，系统内核将调用处理程序，同时内核传递给处理程序一个指向数据的指针。

系统内核中的任何东西，例如设备驱动程序，都可以创建和使用任务队列，但系统内核只创建和管理三个任务队列：

1. timer

此队列中的任务将会在下一个系统时钟时执行。每次系统时钟开始的时候，系统内核都要检查队列中是否含有任何的入口，如果有的话，内核就将 timer 队列 bottom half处理程序设置为活动的。当调度进程下一次运行时， timer 队列 bottom half处理程序和其他的bottom half处理程序将会运行。

2. immediate

此队列bottom half处理程序的优先权要比timer 队列 bottom half处理程序的优先权低，所以将会在稍后运行。

3. scheduler

此队列由调度程序直接处理。它用来支持系统中其他的任务队列。

当系统内核处理任务队列时，队列中的第一个元素的指针将会从队列中移走，并代以空指针。事实上，这种移走队列中元素的操作是自动的，并且是不能中断的。然后系统内核将轮流调用队列中每一个元素的处理程序。

17.3 计时器

操作系统需要能够调度可能在将来发生的事件，所以系统中需要存在一种机制，使得事件在较为精确的时间调度运行。任何一个希望支持操作系统的处理器都应该有一个可编程的内部时钟，可以周期性地中断处理器。这就像一个节拍器一样可以协调系统中的各个任务。

Linux系统中有两种系统时钟，图17-3显示了这两种机制。

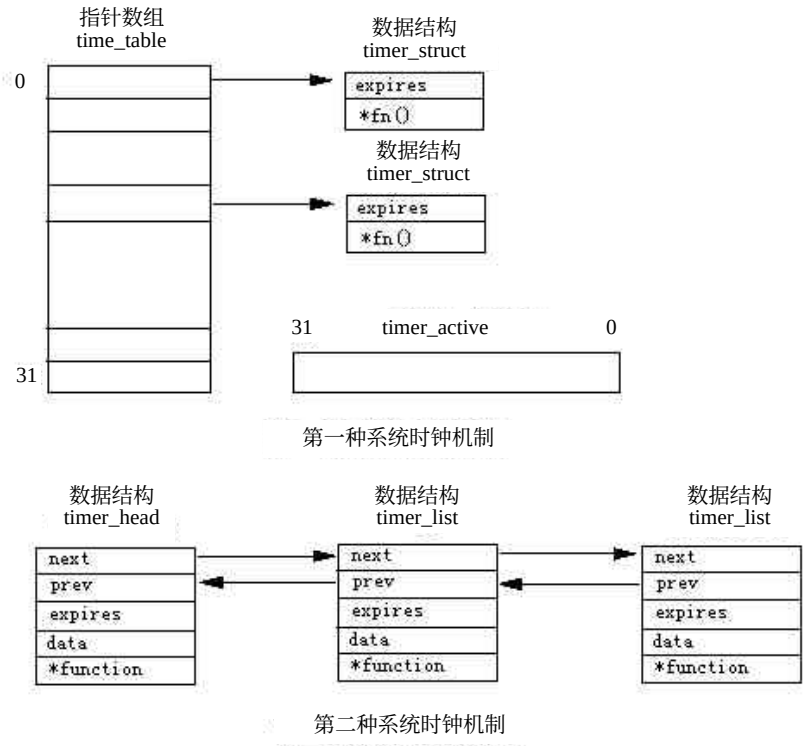

图17-3 系统时钟结构示意图

第一种机制是一种老的系统时钟机制，它是一个包括 32个指针的静态数组，每一个指针都指向一个timer_struct数据结构，并且还有一个timer_active来标识活动的系统时钟。

第二种系统时钟机制是一个timer_list数据结构的链表，它以失效的时限按升序排列。

17.4 等待队列

系统中经常发生进程需要等待系统资源的情况。例如，进程可能需要文件系统中描述目录的VFS 索引节点，但该索引节点并不在系统的缓冲区缓存中。在这种情况下，进程必须等到系统内核将索引节点从物理设备中读取到缓冲区缓存中才能继续运行。

Linux系统内核使用了一个简单的数据结构，它包括一个指向进程 task_struct的指针和一个指向队列中下一个元素的指针。

当进程添加到等待队列的末端时，它们或者是可以中断的，或者是不可以中断的。

当系统内核处理等待队列时，等待队列中的每一个进程都被设置成 RUNNING状态。当等待队列中的进程可以被调度运行时，系统内核所做的第一件事就是把进程从等待队列中移走。

等待队列可以用于同步系统资源的存取，Linux系统内核也常常使用它来实现信号量。

17.5 信号量

信号量用于保护系统中关键的代码或者数据结构。应当注意的是，每一次对关键数据，例如VFS索引节点的存取，都是由系统内核代表某一个进程来完成的。允许一个进程修改其他进程可能正在使用的数据是十分危险的。Linux系统中使用信号量技术使得某一时刻只有一个进程可以存取关键区域的代码和数据。其他希望存取此资源的进程只有等待直到此资源空闲为止。等待的进程将会被挂起，但系统中的其他进程可以正常地运行。

Linux系统中信号量的数据结构包括以下的信息：

1. count

记录希望使用此资源的进程个数。正值意味着此资源可用，负值或零值意味着进程正在等待此资源。其初始值为1，说明此时有一个且仅有一个进程可以使用此资源。当进程使用此资源时，它们将会减少此字段的值。当它们释放此资源时，将会增加此字段的值。

2. waking

等待此资源的进程个数，同时也表示当此资源空闲时，可以唤醒以便被执行的进程个数。

3. wait queue

当进程等待此资源时，它们将会被放入此等待队列。

4. lock

用于存取waking 字段时所用的锁。

假设信号量的初始值为1，第一个进程将会发现此计数器为正值并将其减1，也就是说此时计数器的值为0。现在此进程拥有了由信号量保护的关键代码或者数据。当进程放弃此资源时，它将会把信号量加1。最为理想的状态就是没有其他任何的进程竞争此资源。

当一个进程正在使用此资源时，如果其他进程也试图使用此资源，这时进程将把信号量减1。信号量现在变为了一个负值（-1），所以其他进程无法使用此资源，它只有等待使用资源的进程释放资源以后才能使用。系统内核使等待的进程进入到睡眠的状态，直到使用资源的进程退出时才唤醒它。等待的进程将自己插入到信号量的等待队列中。