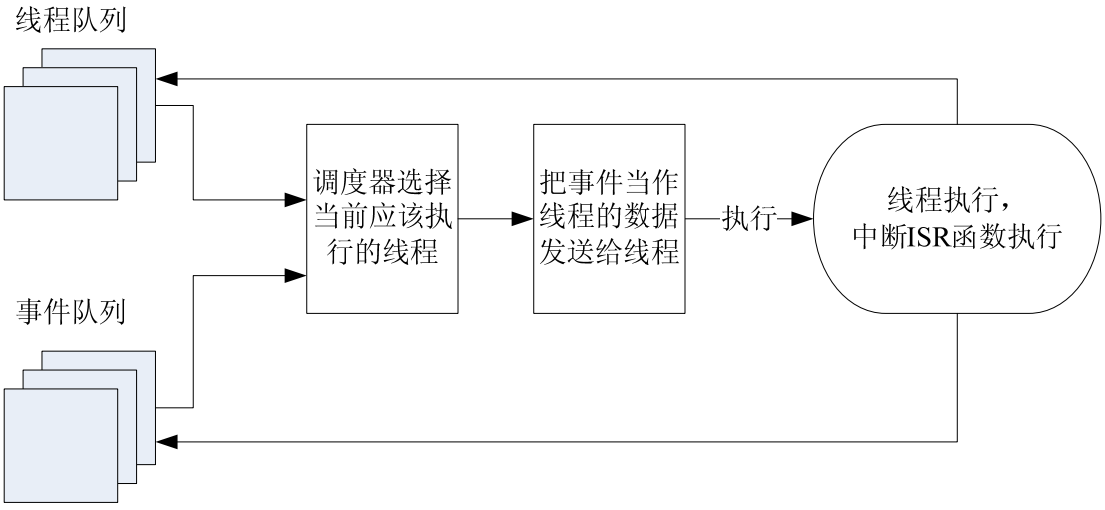
天然的多核操作系统

众多周知，目前 CPU 正往多核化发展，除通用 CPU 外，嵌入式 CPU 甚至 DSP 也是如此，比如 ADI 公司的 blackfin561 就是一颗只有几十 K 内存的双核 DSP。如何提高多核系统的效率，提高 CPU 的并行度，是多核世界面临的一个极具挑战的问题。传统操作系统下，要提高多核计算并行度，需要非常有技巧地操纵多线程编程。都江堰操作系统的出现，使这一难题迎刃而解，即使一个完全不知道线程为何物的程序员，也可以轻易编写出优异的多核应用程序。djyos 有计划移植到 blackfin561 上，有望成为第一个在只有几十 K 内存的 CPU 上实现对双核的完美支持的 RTOS。

1. 传统以线程为核心的调度

程序运行流向发生改变，肯定是因为程序所处的环境发生了某种变化，这种变化可能是从外部 IO 输入了数据，也可能是从通信网络上接收到了数据，或者用户输入了指令，或者程序执行过程中产生了新的数据或改变了某些数据。如果我们把引起程序执行流向发生变化的因素统称为事件的话，那么传统的、以线程为调度核心的操作系统中，在调度器的控制下，事件和线程的逻辑关系如下图所示：



上图中，我们只看到线程，有人问：那进程呢？进程实际上只是代码和数据等组成的环境，其执行程序是由线程来实现的，调度器的实际调度单元是线程。线程是由应用程序的程序员创建来处理特定事件的，线程与事件有对应关系，即某一类事件，只能由应用程序的程序员指定的线程处理。在多核环境下，该线程究竟放在哪个内核中呢？无论是由操作系统来分配，还是由程序员分配，在一段时间内，它都只能占据一个 CPU 核，如果该类型的事件频繁发生，也只能由该 CPU 核一点点处理，另外的 CPU 核即使闲着，也帮不上忙。有兴趣的读者，可以做一个试验，在双核 CPU 上用 adobe 的软件把一个很大的 word 文档转换成 pdf 文档，在转换过程中，如果没有其他程序在运行，CPU 的占用率将一直保持在 51%左右，就是说，一个 CPU 在一直 100%运转，另一个则一直闲着。用户呢？只能眼巴巴地等待那一个 CPU 核把活干完。

举个例子，比如某公司可能有市内送货，省内送货，省际送货三种业务需求，就为三种业务分别聘用了一个送货员，并且配置了 3 辆送货车。相当于 3 个线程，分别处理市内送货、省内送货、省际送货事件，同时准备了 3 个 CPU 内核，可以用来执行线程。如果在某一段时间内，市内送货的任务特别多，也只能由市内送货员逐一递送，无论你采用何种算法分配车辆，总之只有一个送货员在干活，即使加班加点还延误送货时间，另两个送货员也只能隔岸观火。

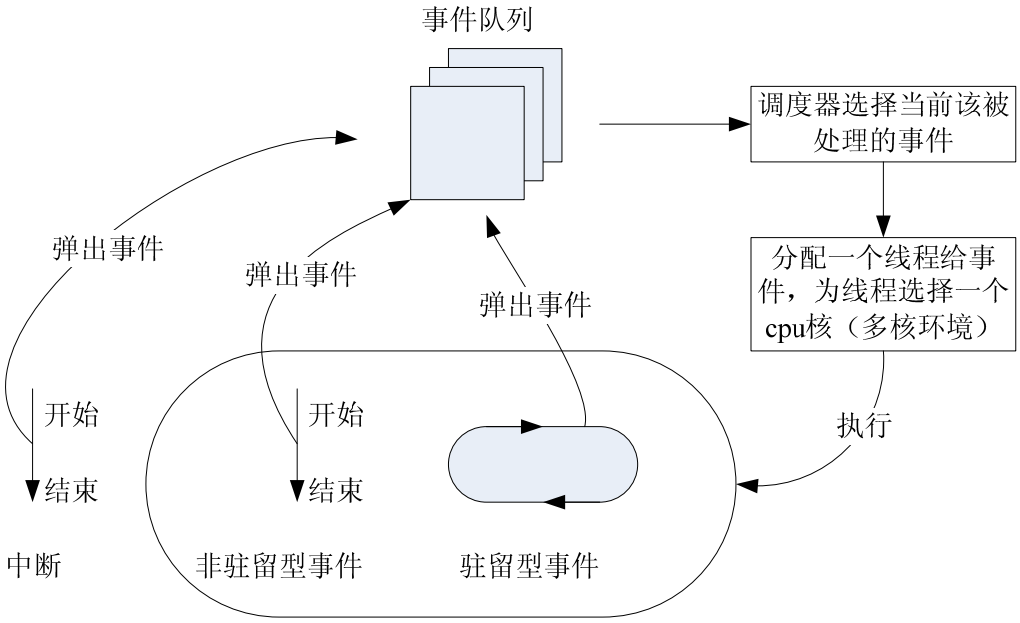
也许你会说，我哪里有这么傻，我就请三个送货员（三个线程），每人占一个 cpu（一辆车），

不限定他往哪里送，不就行了吗？OK，你知道你的程序的执行平台有几个 CPU 核该请几个送货员吗？你希望你的应用程序只能在 3 核的系统上运行吗？当产生（送货）事件的时候，应用程序自己决定由哪个送货员发送吗？还是由送货员之间自己扯皮？

2. 都江堰操作系统以事件为核心的调度

产生上述送货员问题的根本原因，在于线程是应用程序创建的，并且为其编制线程入口程序，实际上应用程序代码和线程是绑定在一起的，操作系统不能动态地根据某一类任务的大量突发产生而动态地为其创建更多的线程，分配更多的 CPU 核。别告诉我在一些现代编程工具（如 VB、VC 等）支持下，程序员无需自己创建线程，对于操作系统来说，那些编程工具也是应用程序。

都江堰操作系统的调度算法，是以事件为核心的，应用程序只管为事件编制处理程序，而线程是由操作系统分配的，线程绑定在哪个 CPU 核上，也是由操作系统决定的，这样就解除了应用程序代码与 CPU 核的绑定。如果某一类事件频繁发生，操作系统将会给它分配多个线程，这些线程将根据各 CPU 核的繁忙程度，分配到不同的 CPU 上。都江堰操作系统中事件和线程的逻辑关系如下图：



在这种调度算法跟传统算法刚好相反，传统调度算法中，事件作为线程的数据（也可称作资源），被线程处理，而都江堰操作系统中，线程和 CPU 都是事件的资源，事件使用线程来处理自己的需求。应用程序只需要通过登记事件类型把可能需要处理的任务告诉 CPU，弹出事件告诉 CPU 某发生了类型的事件的实例。而分配线程和 CPU 是操作系统调度器的事，操作系统完全可以根据各 CPU 核的繁忙程度灵活分配 CPU 核。特别是，如果某一类的事件突发大量产生，操作系统可以为其分配大量的线程，并且把这些线程均匀分布在不同的 CPU 核上，让多核开足马力。而所有这些过程，都是操作系统完成的，程序员无需考虑创建线程和分配 CPU 的具体细节。