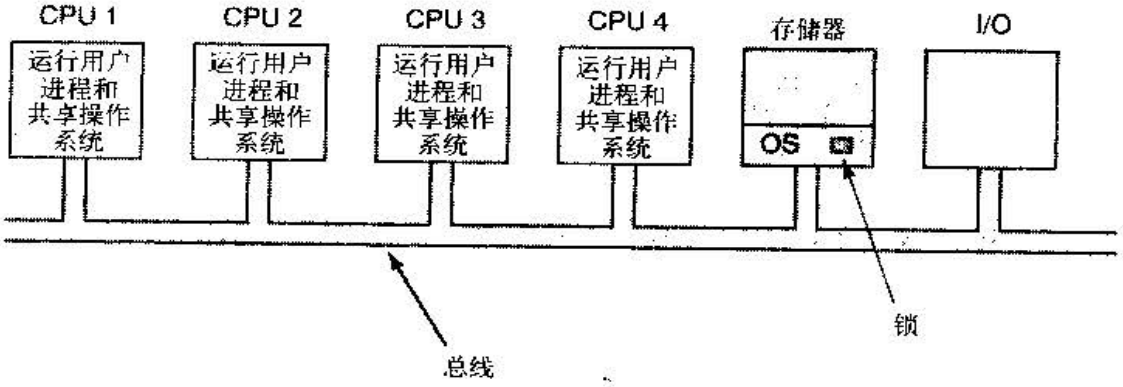

图8-9 SMP多处理机模型

这个模型是可以工作的，但是它几乎同主从模式一样糟糕。同样假设，如果所有时间的10%花费在操作系统内部。那么在有20个CPU 时，会出现等待进入的CPU长队。幸运的是，比较容易进行改进。操作系统中的很多部分是彼此独立的。例如，在一个CPU运行调度程序时，另一个CPU则处理文件系统的调用，而第三个在处理一个缺页异常，这种运行方式是没有问题的。

这一事实使得把操作系统分割成互不影响的临界区。每个临界区由其互斥信号量保护，所以一次只有一个CPU可执行它。采用这种方式，可以实现更多的并行操作。而某些表格，如进程表，可能恰巧被多个临界区使用。例如，在调度时需要进程表，在系统fork调用和信号处理时也都需要进程表。多临界区使用的每个表格，都需要有各自的互斥信号量。通过这种方式，可以做到每个临界区在任一个时刻只被一个CPU执行，而且在任一个时刻每个临界表（critical table）也只被一个CPU访问。

大多数的现代多处理机都采用这种安排。为这类机器编写操作系统的困难，不在于其实际的代码与普通的操作系统有多大的不同，而在于如何将其划分为可以由不同的CPU并行执行的临界区而互不干扰，即使以细小的、间接的方式。另外，对于被两个或多个临界区使用的表必须通过互斥信号量分别加以保护，而且使用这些表的代码必须正确地运用互斥信号量。

更进一步，必须格外小心地避免死锁。如果两个临界区都需要表A和表B，其中一个首先申请A，另一个首先申请B，那么迟早会发生死锁，而且没有人知道为什么会发生死锁。理论上，所有的表可以被赋予整数值，而且所有的临界区都应该以升序的方式获得表。这一策略避免了死锁，但是需要程序员非常仔细地考虑每个临界区需要哪个表，以便按照正确的次序安排请求。

由于代码是随着时间演化的，所以也许有个临界区需要一张过去不需要的新表。如果程序员是新接手工作的，他不了解系统的整个逻辑，那么可能只是在他需要的时候获得表，并且在不需要时释放掉。尽管这看起来是合理的，但是可能会导致死锁，即用户会觉察到系统被凝固住了。要做正确并不容易，而且要在程序员不断更换的数年时间之内始终保持正确性太困难了。

8.1.3 多处理机同步

在多处理机中CPU经常需要同步。这里刚刚看到了内核临界区和表被互斥信号量保护的情形。现在让我们仔细看看在多处理机中这种同步是如何工作的。正如我们将看到的，它远不是那么无足轻重。

开始讨论之前，还需要引入同步原语。如果一个进程在单处理机（仅含一个CPU）中需要访问一些内核临界表的系统调用，那么内核代码在接触该表之前可以先禁止中断。然后它继续工作，在相关工作完成之前，不会有任何其他的进程溜进来访问该表。在多处理机中，禁止中断的操作只影响到完成禁止中断操作的这个CPU，其他的CPU继续运行并且可以访问临界表。因此，必须采用一种合适的互斥信号量协议，而且所有的CPU都遵守该协议以保证互斥工作的进行。

任何实用的互斥信号量协议的核心都是一条特殊指令，该指令允许检测一个存储器字并以一种不可见的操作设置。我们来看看在图2-22中使用的指令TSL（Test and Set Lock）是如何实现临界区的。正如我们先前讨论的，这条指令做的是，读出一个存储器字并把它存储在一个寄存器中。同时，它对该存储器字写入一个1（或某些非零值）。当然，这需要两个总线周期来完成存储器的读写。在单处理机中，只要该指令不被中途中断，TSL指令就始终照常工作。