对如何模拟的探讨

CPU如何模拟单个CPU，从某个角度讲，一个CPU的本质是指令的执行流，因此模拟一个CPU存在理论上通过多个线程同步互斥，使多个线程在同一时间内只有一个线程在运行，实现不并发的单cpu执行流实现模拟。  
上述讲到了关于单个CPU可以通过多个线程同步互斥模拟，而这里我们采用用一个CPUwithTime线程模拟CPU处于内核态的所有指令的执行流，而用多个Process线程模拟用户态进程的执行流，且每一个Process线程模拟用户态中单个进程的执行流  
如何模拟中断，中断的本质就是周期性的检查固定执行流之外的事件并执行响应，只不过周期的大小为一个指令周期，且硬件支持，响应外部事件的代价极小，正确和完备的理解中断是是理解操作系统的难点却也是核心，在本次模拟中，中断的模拟有2种方式：  
CPUwithTime线程完成内核处理流程后立刻唤醒获得下一个获得用户态执行权限的Process进程，并睡眠一段时间，  
除非@@@@Process进程在CPUwithTime线程睡眠期间主动用interrupt打断CPUwithTime线程的睡眠，并挂起等待再次被CPUwithTime线程唤醒，CPUwithTime线程进入interrupt异常的处理程序，去Process进程的私有空间内拿到参数（获得中断的类型和参数），并执行kernel  
类的方法响应Process进程此次的主动interrupt打断的目的，从而达到操作系统为用户态程序提供资源管理的服务的目的，没错，这种中断模拟的是访管指令，是用户态主观意愿切换到核心态唯一的方法，也是实现系统调用的本质  
否则@@@@Process线程将在CPUwithTime线程睡眠期间度过一段完整的与内核无关的时间段，但是CPUwithTime线程苏醒后将剥夺它的执行权力，并进入默认的流程，即先检查在睡眠期间的外部事件请求，比如磁盘IO传输完毕的中断等等，并给予服务，注意到这种中断我们模拟时是非interrupt打断的，如果没有任何外部事件请求，则默认进入时间片中断的处理程序，比如倘若时间片用完了，这可能进入shedule进程调度程序，选择另一个Process线程拥有CPUwithTime线程睡眠后的执行权  
综上2种中断的模拟完成，分别实现模拟自愿性中断和强迫性中断

以此为基石我们模拟了并为用户态进程提供了6个系统调用方法

*create\_process 创建进程和资源（包括模拟私有进程用户空间，内核态的pcb）*

*exit\_process 销毁进程和它占用的资源（包括模拟私有进程用户空间，内核态的pcb，以及释放其他进程发送给它的挂在pcb上的消息缓冲块资源），并仿造linux系统，终止的进程的所有子进程的父亲进程都将变成pid为1的init进程，以保持由create\_process和exit\_process系统调用产生的process\_tree保持完整树状结构，（父子进程关系现在除了保持进程树结构之外并不重要，但是如果扩展程序的话，增加*6个系统调用方法之外的wait\_join系统调用，想与子进程达机制上的同步，就显的尤为重要*）*

*send receive*

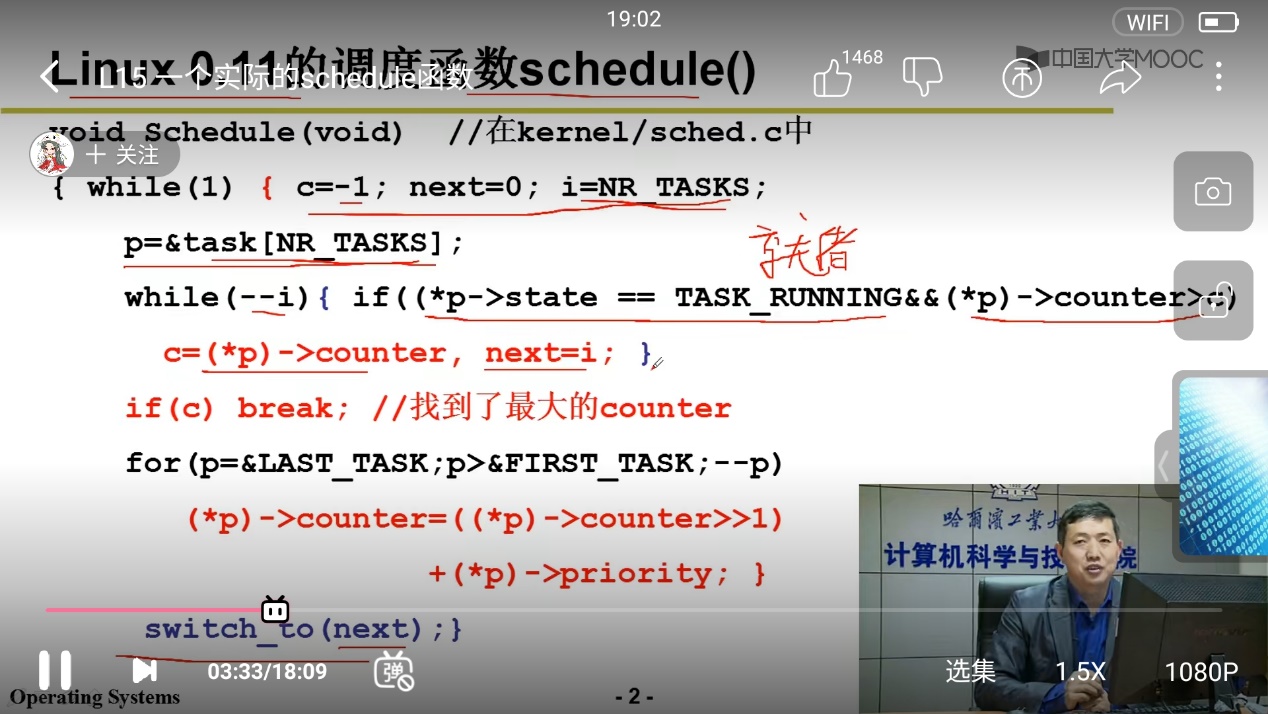
*缓冲机制，不想多谈，但是我们对吉林大学课件给出的实现方式进行了扩展，绑定在接受进程pcb上的缓冲块并不是以一维链式结构的形式组织的，而是二维的链式结构，通俗的讲就是list<list<Block>>,这样的设计很好的解决了一次进入内核拿的不是固定的一个块，而是一个一维的list<Block>，进程间通讯一次的消息量大小不会受到Block块大小的限制*

*require\_disk\_read 请求磁盘读，它的内核服务是申请缓冲块后挂到设备的IO链上并将请求进程挂到等待队列，等到下一次来自磁盘设备发出的强迫性中断，并唤醒相应等待的进程，（注意到require\_disk\_read系统调用是一个自愿性的中断，而磁盘完成一个Block传输后发出中断是强迫型中断，它们在内核中的服务程序是不一致的，比如前者是挂起进程，后者是唤醒相应进程）*

*require\_disk\_write 同理，但是磁盘写请求不需要让请求进程等待阻塞（注意到后面4种系统调用send receive require\_disk\_read require\_disk\_read 用到的缓冲块来只同一个缓冲池，这也体现了模拟一个系统环境的优势，不同的内核系统服务之间并不是相互孤立的）*

*（此外这里有必要提及，磁盘应对相应io链请求采用的是最简单的电梯扫描算法，该算法服务属于磁盘强迫性IO中断的服务，也就是说只在CPUwithTimer线程中进行）*

作为课设要求的调度算法模块，有必要提及Kernel中shedule的实现，仿造的Linux0.11版本的调度算法，一种结合优先数，有效充分利用前后台程序效率，调度开销极小的基于时间片的调度算法，不过基于多核CPU的构架，需要对其进行简单改进，加入多核调度互斥的机制。附图c语言实现图



基于多核CPU系统的拓展：  
当一个操作系统工作在多核CPU环境下时，需要注意2点：

1. 对schedule函数访问资源采取互斥机制，在单核CPU下，由于进程的‘’假性‘’并发，需要对加入共享资源互斥机制，但schedule所访问资源（比如pcb就绪队列）一般情况下不会冲突，它是进程的‘’假性‘’并发的交接点，造成了其具有如此的性质，但是在多核CPU下，需要对schedule函数访问资源采取互斥机制，否则就会出现不可预期的错误，在我们的模拟课设实现的过程中也确实验证了这一点
2. 关于”CPU无关中断”的分散措施，中断可以有很多分类，从程序的角度看，可以分成自愿型和强迫型，从CPU芯片本身看，分为内部中断和外部中断，在多核系统实现的过程中，我盟发现一个有趣的现象，”CPU无关中断”和”CPU有关中断”，这是我们自己造的词，比如磁盘IO中断，它发送中断其实是绑定到一个CPU的，即有一个CPU专门处理磁盘IO中断的中断请求，假设磁盘IO中断同时向3个CPU发送中断，则3个CPU都会进行磁盘IO中断的处理请求，不仅浪费CPU志愿，还会出现预期外的不可控，这种中断都应该绑定固定的CPU。

而且经过查阅资料发现确实在多核环境下，有些中断是绑定CPU的，比如现在加一个打印机设备，它的中断可以绑定到CPU2上，把多个“CPU无关中断”分散到多个CPU中，防止单个CPU应为绑定太多“CPU无关中断”而自身影响效率