L0 实验报告：

实验设计思路：

我在本实验考虑完成的是一个类似于打砖块的游戏，就是类似于将屏幕的边界看做墙壁，屏幕底部会有一辆小板车，一个弹球会四处弹射，玩家需要在探求落到底面边界之前用小板车将球再次谈起。我记得原来的游戏中是考虑了班车速度对小求速度和方向的影响的，但此处并没有考虑。

设计的方法并不难，由于速度与方向一定，所以只要记录下小球当前的运动转态，判定此时小球所处的位置，便构成了一个类似于状态机的结构，大体思路便是特定条件下的状态转化。

实验的问题：

本次实验的收获除了了解了基本的绘图原理之外，便是对 while(1)这样一个语句使用的巧妙之处的认识。在本次试验中，我除了在小球运动时运用了一个 while(1)的结构，为了达到可以由玩家自由决定何时何处发球的目的，同样在游戏开始前也运用了一个 while(1)，其搭配函数 uptime()所带来的其妙效果是我所始料未及的，我一个 de 了很久的 bug 就是记录时间的变量没有在每一次循环之后相加，导致一直死循环。

还有就是我曾考虑过将球的图形换成圆形，但是这样原来一个 5*5 的球在原来只需要调用一次 draw_rect 的情况下变成要调用 25 次，考虑到性能的原因，就将小球保留为正方形。

L1 实验报告

实验设计思路：

实验开始我的设计思路是实现一个 buddysystem 加上 slab 配合的操作系统，我也顺利的实现了 buddysystem，但在想要实现 slab 时，遇到了较多技术上的阻碍，因此未能如愿以偿，STFW 之后发现 buddysystem 之所以在分配小内存时表现会比较差是因为由于小内存频繁访问的性质，也即频繁的申请以及释放，就会导致 buddysystem 分割和合并的成本过大，所以我干脆将 buddysystem 中较小的内存不实行合并操作，也即较小的内存一旦被分配出去，即使返还，也只是简单的插入到对应大小的空闲链表之中，以供后续使用而不合并，从

而避免后续的合并拆分开销，但是这样做的弊端就是，一旦分配为小内存意味着这些内存便不可能再被大于它的内存申请所使用了，极端情况下比较容易造成系统实际可使用内存颇小于剩余内存的情况。

实验中我认为非常有帮助的一句代码：

准确来说就是一句 assert，但就这一句 assert 帮我 debug 除了所有现在我一直的错误，让整个系统现在看上运行的比较流畅，这条语句安放在内存回收的函数之中，由于 buddysystem 分出去的内存一定是 2 的幂次，所以其地址的地位一定是满足一定规则的，而这样一条语句便是描述这样一个规则，若不满足就意味着出现了 bug，它对我的 debug 起了至关重要的帮助。

比较精巧的实现以及存在的问题：

由于内存分配需要链表来管理，于是在设计之初，既然我是分配内存的，那我怎么在分配内存之前先获得一段内存来放我的链表呢？后来突然想通就是将每一段分配出去的内存的开始一小段大小分配来存放链表节点刚刚好，但同时我感觉这针对我的实现会存在一些问题，可能我申请的内存很小，甚至都没有这一小段结构体大，那么这样分配造成的浪费便是非常明显的。

L2 实验报告

实验设计思路：

本次实验我的大体思路是分为两个步骤来解决。首先是在不实现信号量的情况下，只实现进程的创建以及多核基础上的切换，大体思路与讲义提供的类似，使用链表实现了注册具体的事件处理的函数，进程的上下文的保存以及切换同时也以此来实现，我使用一个链表来统一管理这些函数。对于进程上下文的保存，我通过定义全局数组来保存，根据数组下标模 CPU 的数量将每个线程都绑定到特定的 CPU，但我将一开始的 os_run 的上下文与一个进程的上下文分开看待，因为一个 CPU 在没有线程可以执行时，总要有一个上下文来执行的，这个初始的死循环的上下文便是这个功能。

信号量的实现大体比较简单，wait 与 signal 以大体对偶的形式实现，wait 则对应计数减

1，signal 对应计数加 1，在某个信号量 count 小于 0 时，signal 会唤醒等待该信号量的一个进程，wait 则使一个线程进入休眠，也即标记一下这个线程，在调度函数中不会调度标记为不可执行的线程。对于如何解决在持有自旋锁的情况下 int0x80，我的解决就是不让其在持有自旋锁的情况下 int0x80，通过声明一个局部变量作标记，就可以避免线程睡眠后仍旧持有某个信号量的自旋锁。

实验中我认为非常有帮助的一句代码：

应该是模仿的 xv6 自旋锁的实现中的各种 panic，主要是在加入了键盘中断之后，由于该中断不属于可屏蔽中断，因而单纯的关 IF 位无法避免中断的到来，因此在 os_trap 时，会出现 AA 型死锁，而这种情况，只要通过 xv6 实现中的 holding 函数判定一下就可避免，也即如果此时已上锁，则不重复上锁。

比较精巧的实现以及存在的问题：

我认为自己信号量的实现是比较优美的，在线程总数相对固定的情况下，对于每个信号量的等待队列用一个计数数组维护比使用链表更加可靠，也更简明。Wait 和 signal 两个 if 语句思路清晰明朗。存在的最大的问题就是我在编程中遇到的最棘手的 bug，不出意外这也是许多在听过讲解前做的人会碰到的迷之 bug，也就是线程切换时一个线程同时在两个 CPU 上运行出现的 stack smash，虽然听过讲解之后明白了原理但自己调试时却完全瞎蒙，因为这个 bug 均匀的导致了 xv6 各种 panic 的触发，程序跑飞以及 kvm 内核错误，调试这个 bug 时，我对自己的自旋锁实现复查了很多遍最终也没找到什么纰漏，然后就是看调度函数有没有粗心的 bug，但都无功而返，最后抱着试试的心态将线程绑定到 CPU 就莫名其妙的解决了，却完全不知道错误的原理，这次从侧面上体现了并发编程 debug 的困难。