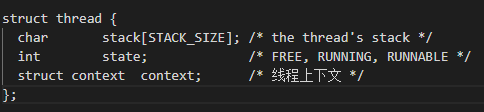
**操作系统大作业3**

**18364057 林文龙**

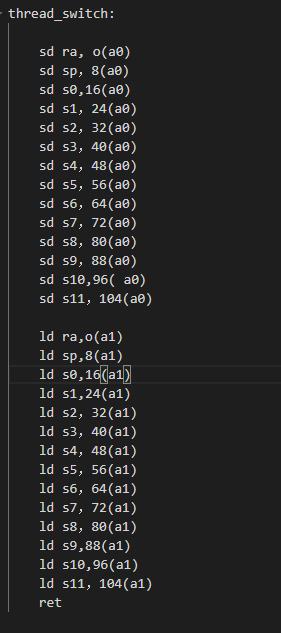
**1.Xv6 lab: Multithreading/Uthread: switching between threads**

实验目的：实现用户级线程切换的功能，给自定义的线程添加上下文信息用于保存状态，然后在thread\_schedule里添加交换上下文代码

首先，修改struct thread，令其可以保存寄存器信息



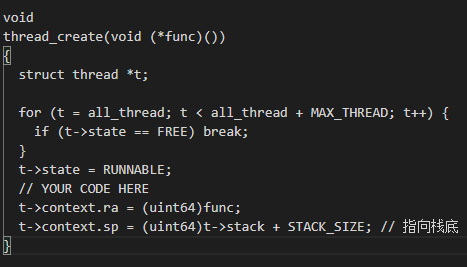
在thread\_schedule添加的switch代码：



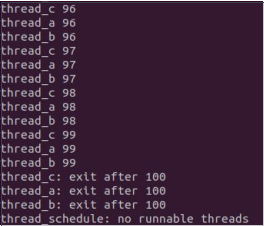
其中a0寄存器对应了switch函数的第一个参数，即当前线程的上下文对象地址

a1寄存器对应的是switch函数的第二个参数，即为将要切换的调度器的上下文进行保存和切换

之后就是要实现一个thread\_create函数



之后在线程运行的时候就可以看到线程轮转时进行了切换

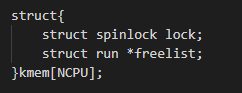


可以看到线程的切换与打印，在完成打印后退出

**2. Xv6 lab: Lock/Memory allocator**

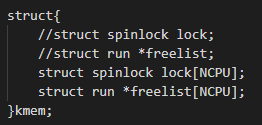
实验目的：在这个实验中，我们需要重新设计代码以提高并行性。多核计算机上并行性差的一个常见问题就是高锁争用，我们需要为xv6的内存分配器和块缓存进行操作以减少争用

在这一块中，我们需要实现每个cpu的空闲列表，并在一个cpu为空时进行窃取。运行kalloctest可以查看是否减少了锁争用，并检查他是否可以分配所有内存

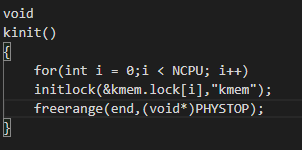


出现大量竞争的原因是因为kalloc()只有一个被单个锁保护的freelist，为了消除锁争用，我们要重新设计内存分配器，以避免单一锁和冲突

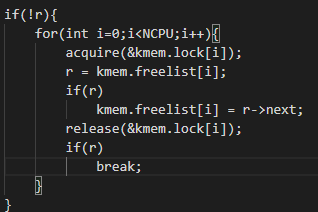
重新设计的内存分配



但由于我们修改了kmem，我们还应该对kinit函数中对内存的操作进行修改



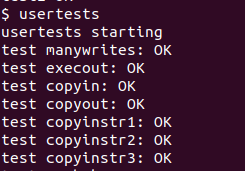
主要挑战是如何处理一个CPU没有空闲列表，但另一个cpu有空闲内存的情况，在这种情况下，该cpu必须窃取另一个cpu的空闲列表的一部分，窃取可能会发生争用，我们希望这种情况少出现

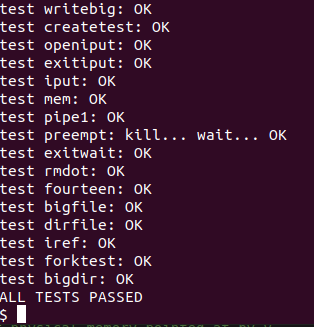


利用r，如果当前cpu的freelist为空，则需要从别的cpu偷一块

如果偷到了，就退出，没偷到就继续直到窃取到为止

运行结果

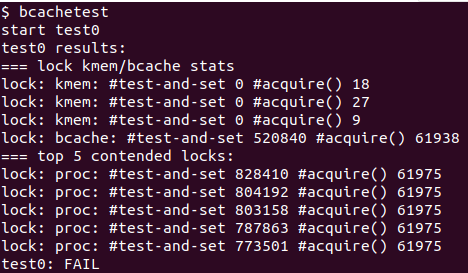




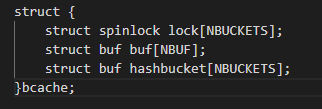
**3. Xv6 lab: Lock/Buffer cache**

实验目的：修改块缓存，以便在运行bcachetest时，bcache所有锁的测试和集数接近于0，理想情况下，块缓存中涉及的所有锁的测试和集的总和为0

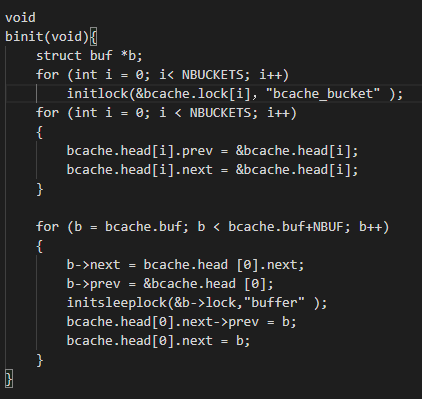
未修改前竞争情况：



首先，出现大量竞争的情况是因为有多个进程大量读写磁盘，而cache只有一个lock的时候就会出现大量竞争，我们需要将将其从链表修改为哈希表，并为每个bucket分配锁，并去掉head



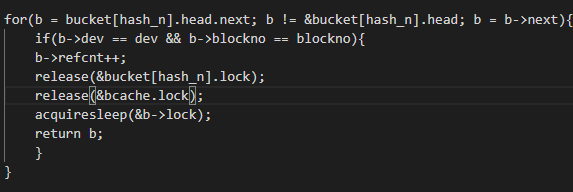
由于我们修改了bcache，因此binit和bget也需要作出修改



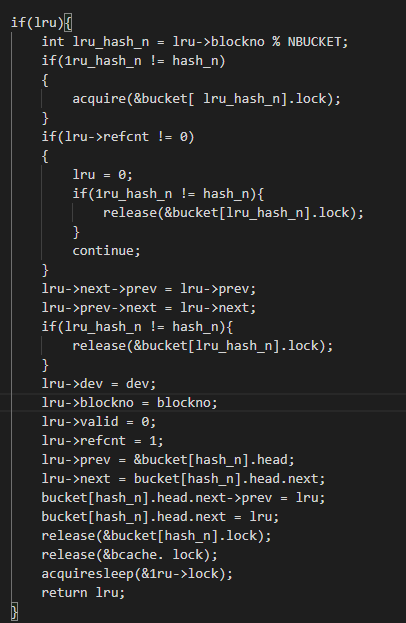
修改节点，将每个bucket的头结点指向自己

此时的buffer没有和磁盘块对应，那么我们使用头插法，将新的buffer不断插到hashbucket的头部





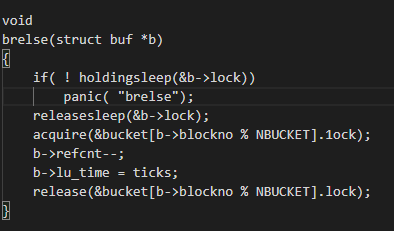
修改bget



循环表示，我们先获取bucket的锁接着在其他的bucket中找到buffer，将该buffer从原来的bucket中断开，并插到对应的bucket中

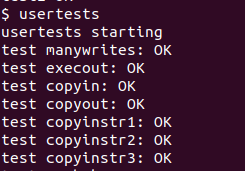
如果没有找到合适的buffer，则释放当前的bucket的锁，并转向下一个bucket

最后修改brelse，该函数的作用是释放一个被锁的buffer，并将其转移至链表头部



运行结果：

8f4a90a79f85f3fd5bbccd3b1a60d6d

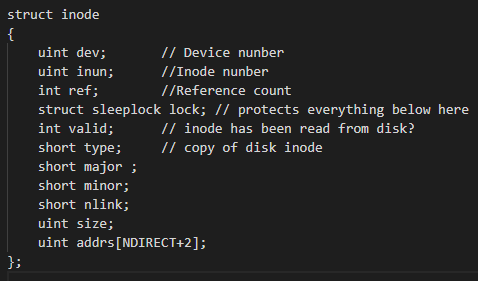


**4. Xv6 lab: File System/Large files**

实验目的：在这个实验中，我们需要增加xv6文件的最大大小，目前xv6的文件被限制为268哥块，也就是268bsize字节（一个直接索引和一个一级间接指引（12+256））

我们需要更改文件，以支持每个节点的二级间接索引，其中可以包含256个一级索引

首先我们修改kernel/file.h 中的 inode的addrs



修改kernel/param.h 中的FSSIZE 为200000



接下来，修改fs.c中的bmap函数，将文件的逻辑块号映射到磁盘块号

bmap（）会根据需要分配新的块来保存文件内容，并在需要时分配一个间接块来保存块地

址。

首先，一级间接索引可以容纳 BSIZE/sizeof(uint)=1024/4=256 个地址。

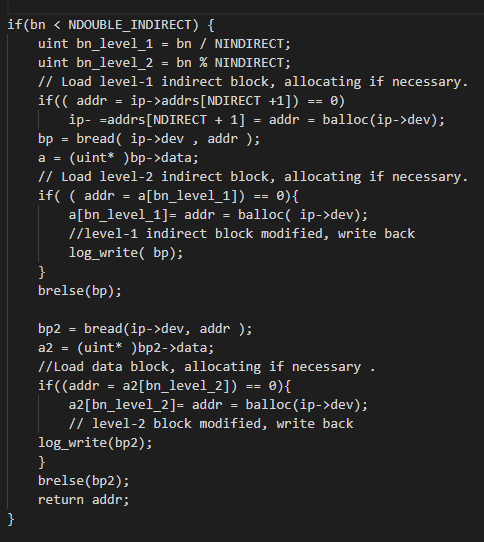
接着，二级间接索引是通过 256 个一级间隔索引地址来寻找中间块，再去找到最终的数据块，

从而实现存储更大文件的目的。

为了鉴别是属于哪种类型的索引类型，在 bmap 函数中设置 bn 参数来完成鉴别功能。

直接索引和一级间接索引已在原函数中给出，分析并模仿，写下二级间接索引的相关代码，

加在一级间接索引的后方。



然后进行测试

