操作系统大作业3

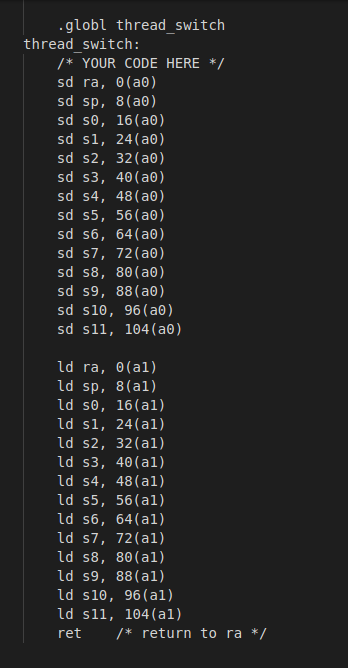
18364039 冷春江

**1、Lab:Multithreading—Uthread:switching between threads**

**1.1 实验目的：**为用户级线程系统设计上下文切换机制，然后实现。

**1.2 实验步骤：**

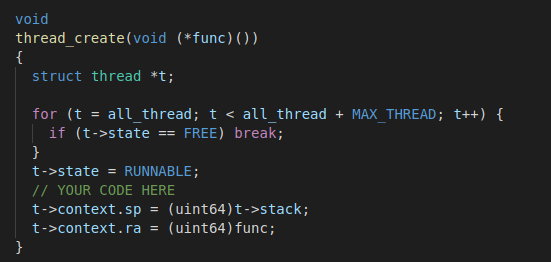
为切换分支 git checkout syscall

****

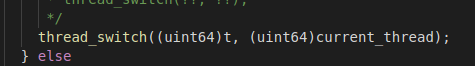
a0寄存器对应了switch 函数的第一个参数，即是当前线程的上下文对象地址。

a1 寄存器对应的是 switch 函数的第二个参数，即是将要切换到的调度器线程的上下文地址。

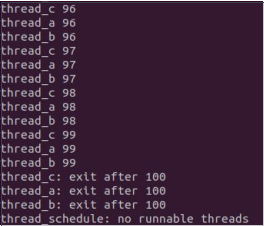
这也符合了，我们在书上学到的有关有关线程切换，我们需要将当前线程的上下文进行保存和切换。



之后增加thread\_switch()的调用



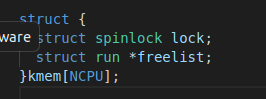
可以看到我们成功的在线程轮转的时候进行了线程切换。



由此可以看出，当前有三个线程不断地切换，打印数字，并在完成打印后，自动退出。

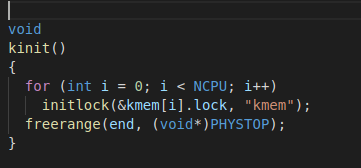
**2、Lab:lock——Memory allocator（20 分）**

**2.1 实验目的**：实现每个 CPU 的空闲列表，并在 CPU 的空闲列表为空时进行窃取



可以看到，Kalloc只有一个freelist，且只被单个锁保护。

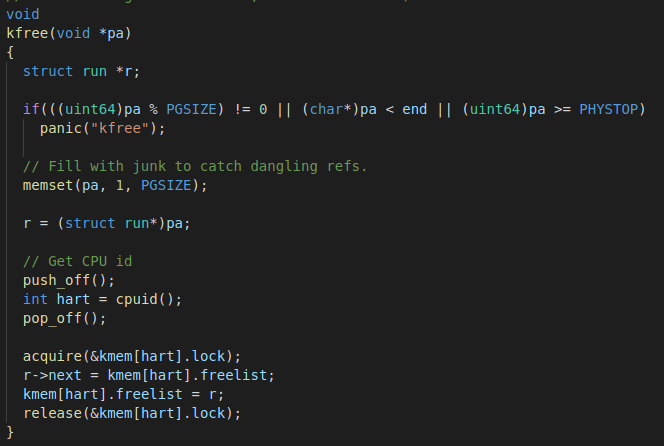
为每一个cpu维护一个列表，同时都有独立的lock。不同的cpu能够并行的利用kalloc和kfree赖处理内存。当一个cpu的freelist为空时，另一个cpu的freelist还有空闲块时，能够从空闲的cpu哪里去获取内存块。



对锁进行不断的初始化。

之后，如果我们想要改变freelist的话，我们要进行头插法，对整个list进行操作，而外界给的操作函数就是trypop()和trypush(),便于直接调用。

之后我们要对kfree做修改



在push\_off()之后，我们获得了对应的cpu的id，然后将被释放的块插到对应的freelist中

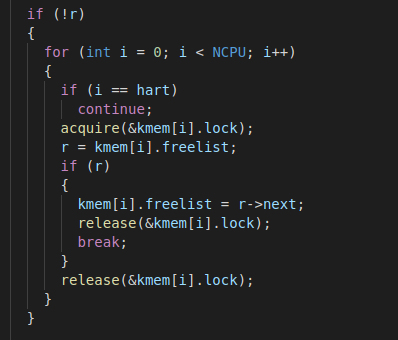
之后我们要去修改主要的kalloc函数。

如下所示，如果当前的cpu的freelist不为空，则直接从里面整一个就好了。

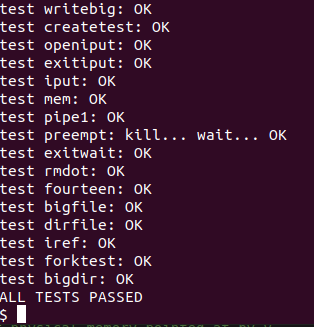
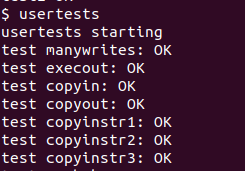
如果cpu的freelist为空，则需要去查询其他cpu的freelist，找到空闲的块r之后，

将其插入到自己的freelist中，

将r从自己的freelist中弹出，初始化，并返回给调用者



结果：

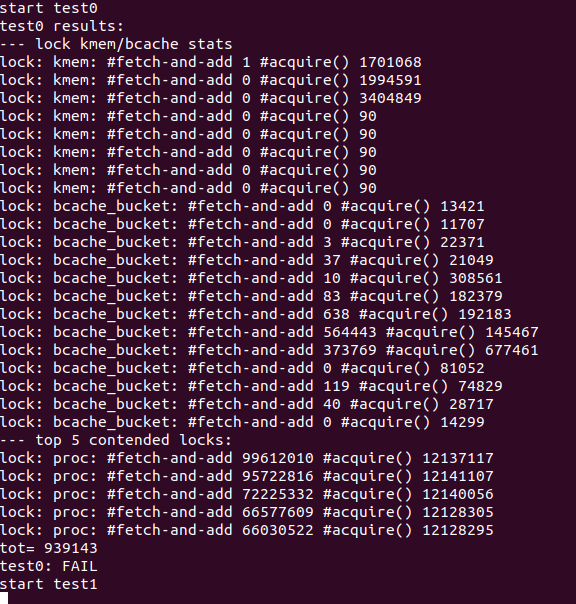


**3、Lab:Lock——Buffer cache**

**3.2 实验目的：**修改块缓存，以便在运行 bcachetest 时，bcache 中所有锁的获取循环迭代

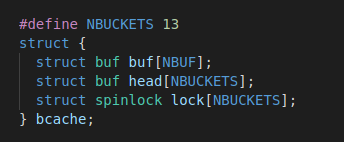
次数接近于零。最终输出应类似于以下所示（尽管不完全相同）。确保用户测试仍然通过。完

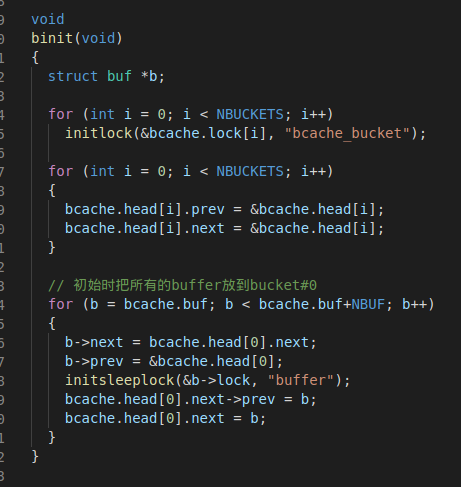
成后，等级应通过所有测试



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。

使用哈希桶来实现，有点类似于hashmap的处理方法，就是当我们出现哈希碰撞之后，我们在bucket后面接上node。这样就可以有效的解决哈希碰撞的问题

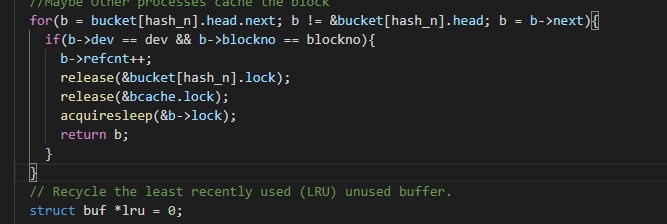




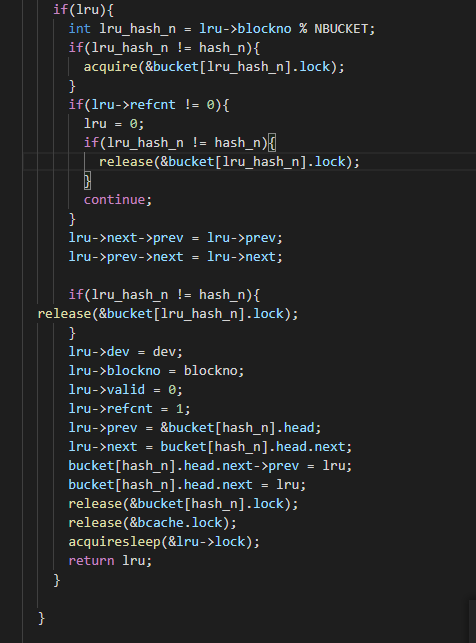
修改节点，将每个bucket的头结点指向自己，

接着此时的buffer没有和磁盘块对应，那么我们就使用头插法，将新的buffer不断地插入到hashbucket的头部。

52e55970ae581723ef1f8c26e0014d4



修改bget函数，如果没有找到，那么就需要到其他的bucket中去寻找，由于在3中将所有的buffer都插入到第一个bucket中，到其他bucket中去寻找的情况将不会少。

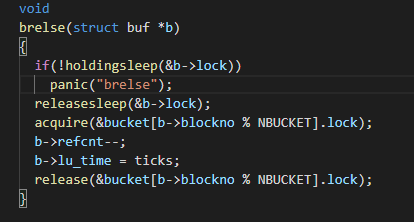


中间的循环表示，我们先获取当前的bucket的锁，接着在其他的bucket中找到buffer，将该buffer从原来的bucket的链表断开，并插入到对应的bucket中去。

第三个框中的表明，如果当前 bucket 没有找到合适的 buffer，则释放当前 bucketd 的锁，

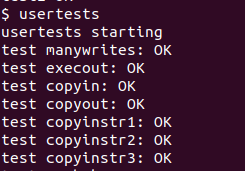
并转向到下一个 bucket。

最后修改函数brelse,该函数的作用是释放一个被锁的buffer，并将其移到链表头部。



结果

8f4a90a79f85f3fd5bbccd3b1a60d6d



Usertests也同样通过

**4、Lab:file system——Large files（20 分）**

**4.1 实验目的：**在这个实验中，我们需要增加 xv6 文件的最大大小。

Bn先减去一级索引的块个数256，再通过取余数操作，得到具体的文件位置。接着找到间接索引，从间接索引读取索引地址，然后去写数据

第二步：修改 kernel/fs.h 中的宏定义以及结构体 dinode 的 addrs。

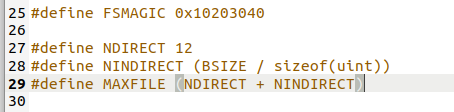
参数解释：

NDIRECT直接索引指向块的数量 11

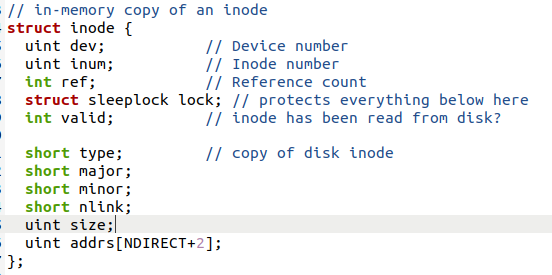
NINDIRECT一级索引能够指向块的数量 256

NDOUBLE\_INDIRECT二级索引能够指向的块的数量 256\*256

MAXFILE直接索引、一级索引、二级索引一共指向的块的数量 11+256+256\*256



修改 kernel/file.h 中的 inode 的 addrs。



修改 kernel/param.h 中的 FSSIZE 为 200000。

6e3570aadb376cdc019b48615becb9a

第五步：修改 fs.c 中的 bmap 函数——作用：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。

结果

