****



**MIT xv6实验报告**

**——lab8 : lock**

**学生姓名 胡轶然**

**学 号 3019244355**

1. **实验目的**
2. 重新涉及内存分配器和文件块缓存，以提高并行性。
3. 理解xv6系统中锁的工作原理。
4. 了解提高程序并行性的方法。
5. **前期准备**
6. 切换git分支。
7. 阅读指导书第六章，了解与锁和并行相关的知识。
8. 阅读指导书Section3.5和Section8.1~8.3的相关代码。
9. **实验内容及实现步骤**

**任务1 Memory allocator**

1. **问题描述**

本任务要求为每个cpu创建一个空闲内存链表和对应的回旋锁，从而降低多个cpu上的进程同时申请内存时产生的锁竞争现象。

同时，**实行动态的空闲内存管理**，当某cpu空闲内存不足时，从其他cpu的空闲内存链表中“偷”出一部分页供自己使用。

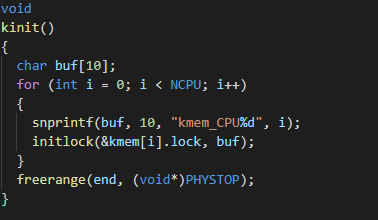
1. **思路与实现步骤**

要实现上述功能，需为每个cpu都分配一个空闲内存链表，即声明一个空闲链表数组，并相应地修改kinit、kalloc和kfree函数中，对空闲内存链表的初始化、分配和释放等操作。

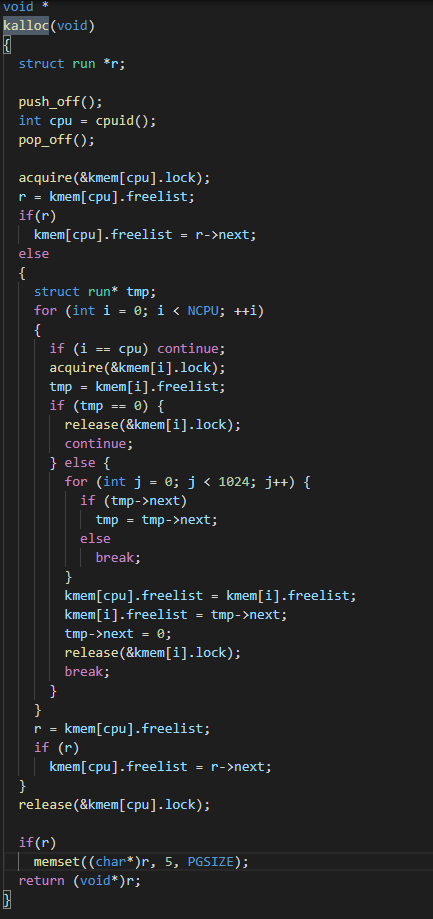
此外，还要在kalloc函数中添加动态管理内存的逻辑，即当某cpu空闲内存不足时，从其他cpu的空闲内存链表中“偷”出一部分页供自己使用。

具体的实现步骤如下：

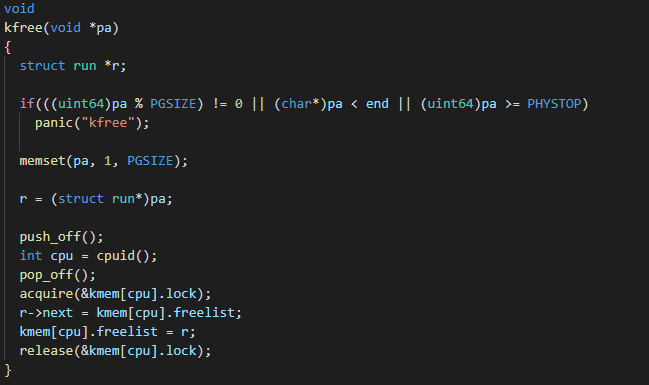
1. 创建空闲内存结构体数组，长度为cpu数量（NCPU）。
2. 修改kinit函数，初始化所有cpu的回旋锁，并将所有空闲内存挂到调用函数的那个cpu的空闲内存链表。



1. 修改kalloc函数，具体内容为
2. 获取cpu id，执行此操作前要关闭中断，以防止此过程中发生进程切换。
3. 获取cpu的自旋锁和空闲内存链表的头指针。
4. 若链表内有空闲页，则取出首个空闲页。
5. 若链表没有空闲页，则遍历其他cpu的空闲内存链表，找出一个有空闲页的cpu，并取出一定数量的空闲页，连接到当前cpu的空闲链表中，然后再从当前cpu的空闲链表中取出一个空闲页。
6. 释放cpu的自旋锁。



1. 修改kfree函数，具体内容如下：
   1. 关闭中断
   2. 获取cpu id。
   3. 允许中断。
   4. 将被释放的页添加到该cpu空闲内存链表的头部（需加锁）。



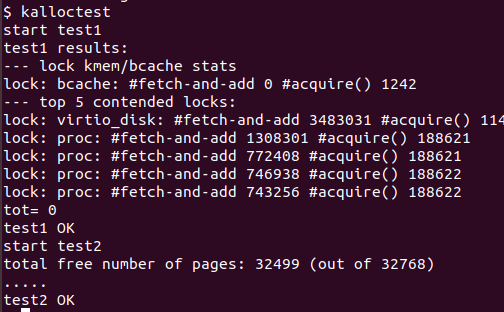
1. **问题与解决方法**

**问题一**：在最初的实现思路中，空闲内存不足的cpu**每次只从其他cpu处“偷”1个空闲块**。经讨论，当某进程大量申请内存时，**这样的分配机制会导致cpu频繁地“偷”取空闲内存，导致程序运行效率下降。**

**解决方案**：决定让内存不足的cpu每次从其他cpu处窃取**一定数量**的空闲内存块，以避免频繁偷取导致效率下降。当被窃取的cpu的空闲内存数量不足窃取数量上限时，窃取其全部空闲内存。

1. **结果**

Kalloc test



**任务2 Buffer cache**

**1. 问题描述**

由于磁盘读写速度与cpu运算速度存在差距，计算机将读取过的数据缓存在内存中，称为“数据高速缓冲区”（buffer cache）。

Xv6使用一个链表来存储所有的buffer cache，同时使用LRU（最近最少使用）算法来处理块的牺牲和替换。由于所有的buffer cache都连接在同一个链表中，同时只能有一个进程读取缓冲区中的数据，这就导致了大量的锁竞争现象。

本任务要求重新设计buffer cache，允许多个进程并发地读写缓冲区数据。

**2. 思路与实现步骤**

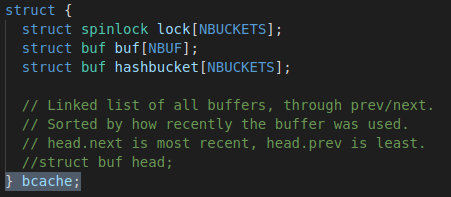
按指导书要求重新设计了buffer cache，将其数据结构从单个链表改为由多个链表构成的数组，并使用哈希函数“对质数取余数“建立链表与磁盘文件块的对应关系。这样，当进程申请读写某个文件块时，只需要申请对应链表的锁，而不会影响其他链表中buffer cache的访问。

在采用哈希链表的同时，要保持缓存块总数不变(30个)，否则就改变了内存大小。因此，要设计一套为哈希链表动态分配内存块的逻辑，其实现思路与任务1相似：初始化时将所有buffer块连接到同一个链表下，每当要访问的文件块对应的链表中不含空闲的buffer块时，就从其他链表中“偷”出1个空闲文件块。

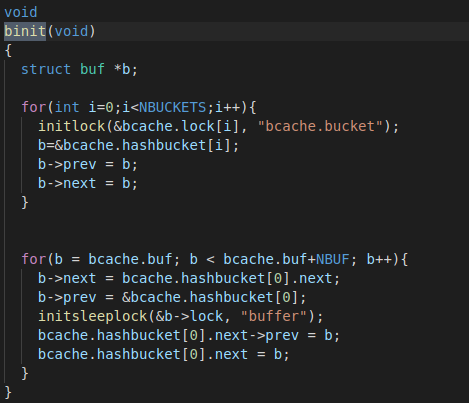
需要注意的是，**在增加链表个数的同时，要保证buffer cache总数不变，否则就是变相地将buffer扩大了若干倍，涉及到了硬件层面的修改，与此任务的原意相悖**。

具体实现如下：

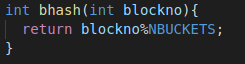
1. 在param.h文件中添加宏定义NBUCKETS=13，用于哈希值的计算。
2. 首先将存储buffer块的数据结构由“一个链表+一个锁“修改为”NBUCKET个链表+NBUCKET个锁”，同时保持buffer cache总数不变。在结构体bcache中以数组形式定义链表头节点、数据节点和回旋锁。



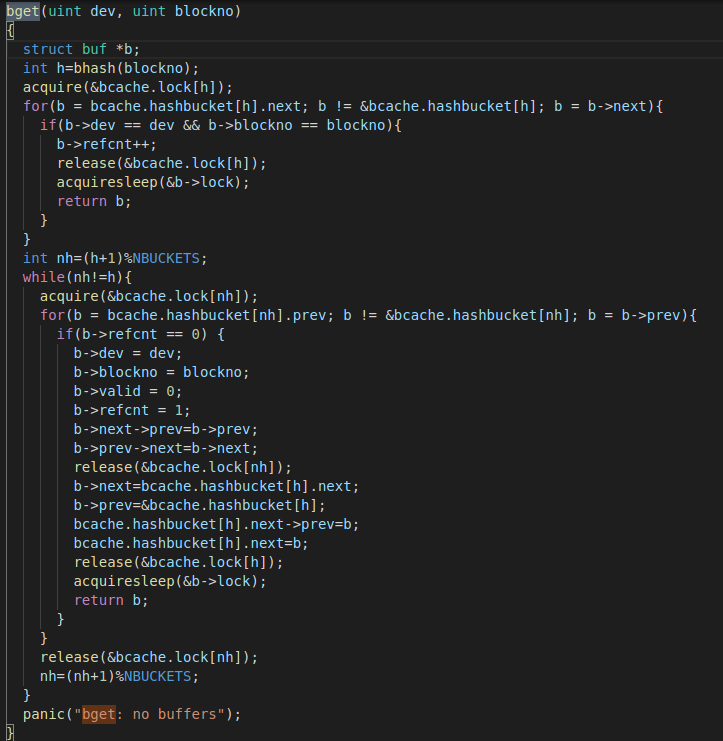
1. 修改binit函数，具体操作如下
   1. 初始化自旋锁数组。
   2. 将hashbucket数组中元素作为链表头节点，搭建NBUCKETS个空的双向循环链表。
   3. 将buf中数组所有buffer cache块都连接到一个链表下。



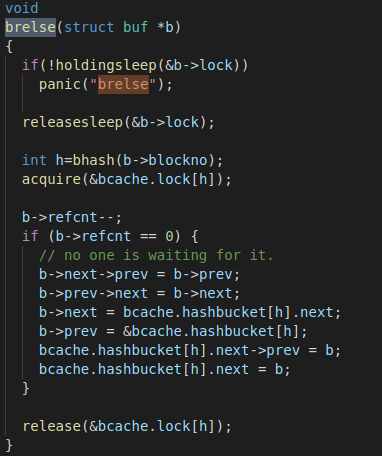
1. 添加哈希函数：“对质数求余数”，用于关联文件块与链表。



1. 修改bget函数，具体操作如下
   1. 调用哈希函数，根据文件块编号计算哈希值。
   2. 对相应的链表上锁，并根据文件块号在链表中寻找buffer cache。
   3. 若在链表中没有找到目标文件块，即发生了MISS事件，则在链表中寻找一个空闲块，并将目标文件块内容缓存到空闲块中。
   4. 当要访问的文件块对应的链表中不含空闲的buffer块时，就从其他链表中“偷”出1个空闲文件块，连接到当前链表中，然后写入目标文件块内容。（此部分具体操作类似任务1中的steal操作，不再赘述）



1. 修改brelse、bpin、bunpin等使用了bache中回旋锁的函数，此部分的修改较为简单，只需根据块号计算哈希码，然后打开、关闭对应的文件缓存锁即可。



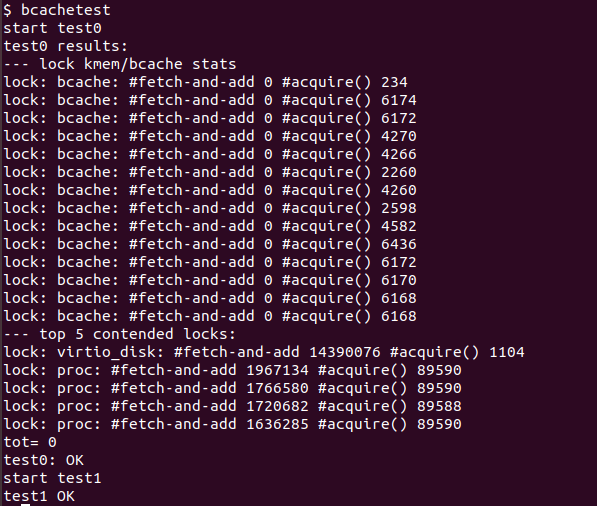
**3. 问题与解决方案**

**问题一**：为追求简便地解决问题，**错误地为每个链表都分配了NBUF(buffer cache总数，30)个buffer cache，相当于变相地将buffer cache硬件扩大了NBUCKETS(链表数量，13)倍。**虽然也能够通过测试，但涉及了硬件层面的修改，与题意不符。

**解决方案**：在与同学交流时无意间发现了此问题，经讨论后决定保持buffer cache总数不变（30），改为**在链表间动态地分配buffer cache，即当进程向某个链表请求buffer，而链表的空闲buffer不足时，系统会从其他链表处“偷”1个空闲的buffer给进程使用。**

**4. 结果**

Bcache test



1. **实验结果**

通过make grade测试。

