1.组号，组员姓名学号以及贡献

第5组

伍德翔18342101 主要负责各个部分的连接及逻辑分析，修改bug

徐康泽18342110 主要实现hash函数、split等新增桶，分页等操作。

魏兆华18342097 主要实现可持久化并、构造和析构函数进行了gtest和YCSB测试

王世界18342094：主要实现insert（），remove（），update等操作

注：我们是团队共同完成的代码，每个人都有一些问题，并且团队讨论解决，代码贡献量基本一致。

2.代码仓库地址

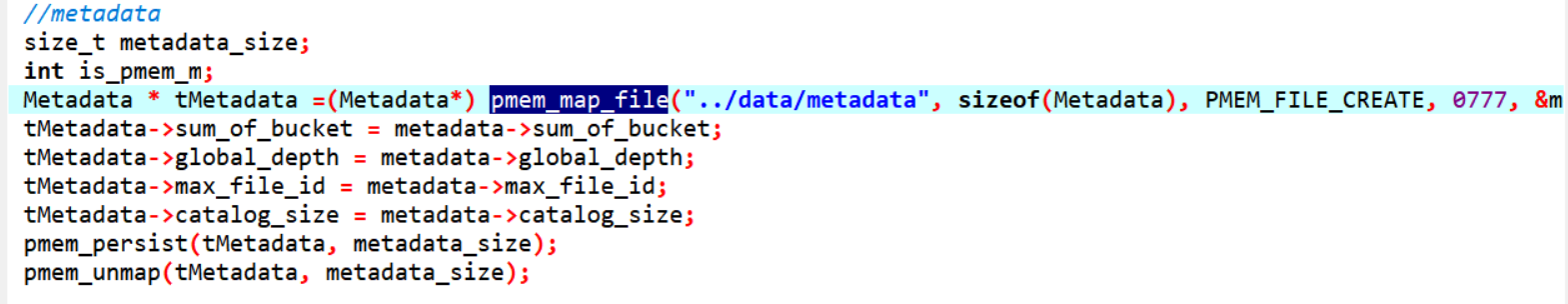
3.代码基本项实现说明

构造函数：

主要进行哈希表的初始化，如果找到data文件夹下有存储数据的文件，那么就直接进行recover（）操作，从文件中读取之前保存的哈希表，实现持久化的还原。如果没有存储数据的文件，那么就进行初始化，我们规定一个页有32个桶，初始化时，将32个桶的首地址放入到free\_list中，将catalog中的buckets\_virtual\_address以及buckets\_pm\_address一一对应，并且将buckets\_virtual\_address连接到桶上。还进行了metadata（元数据）的初始化，记录了下一个数据页的页号，目录的大小，全局深度等信息。

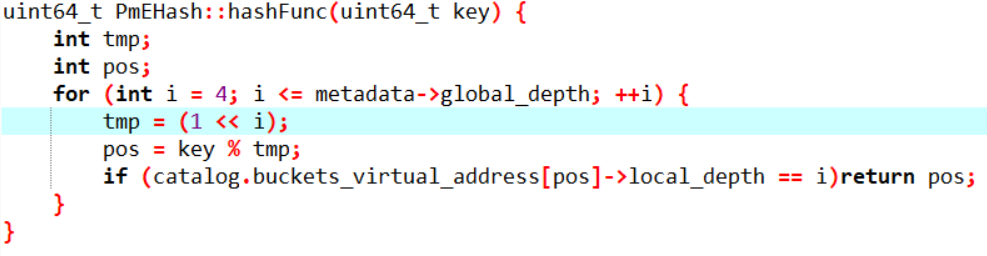
析构函数：

析构函数主要实现将内存中的数据写入到data目录下的文件中，通过pmem\_map\_file依次将catalog，metadata以及数据页放入文件中，实现持久化。

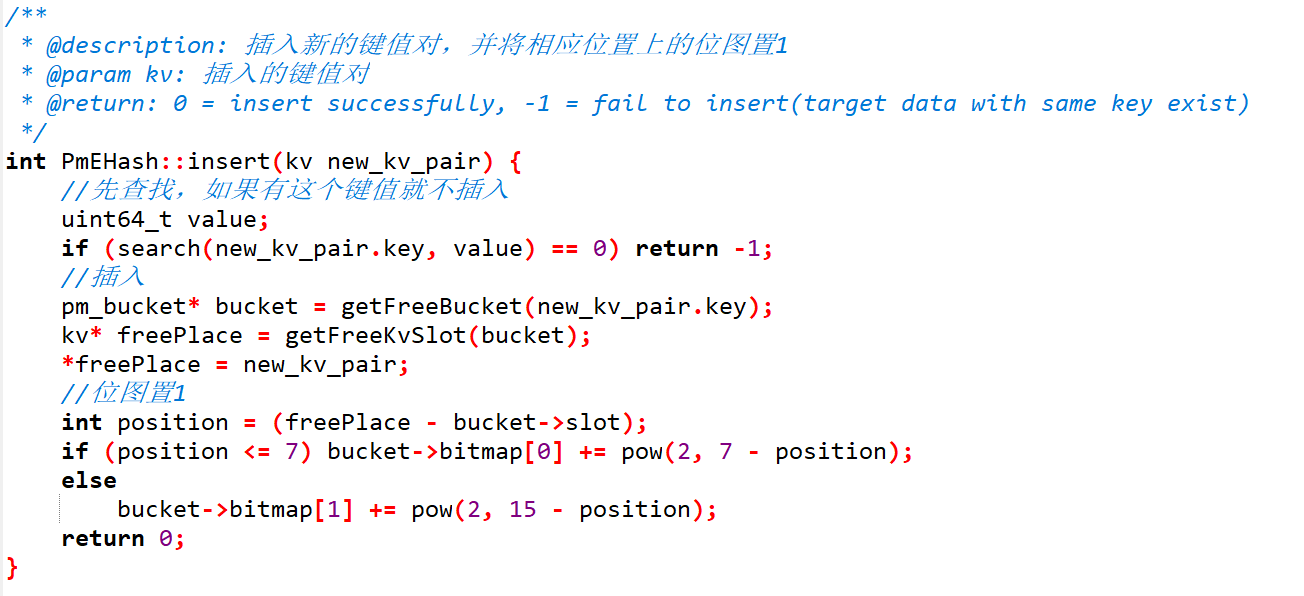


哈希函数：

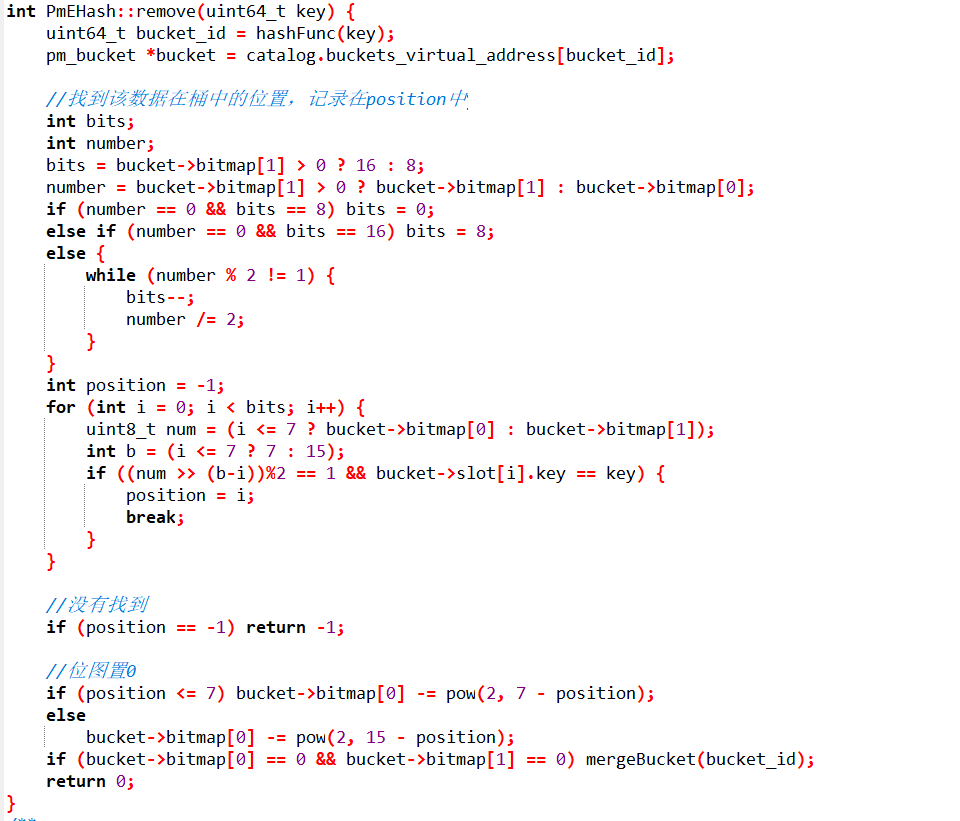
哈希函数根据桶的局部深度与全局深度之间的关系进行判断，实现动态哈希，先从局部深度最小的可能值4开始比较，直到if条件判断成立，此时哈希函数找到了正确的位置，返回桶序号。



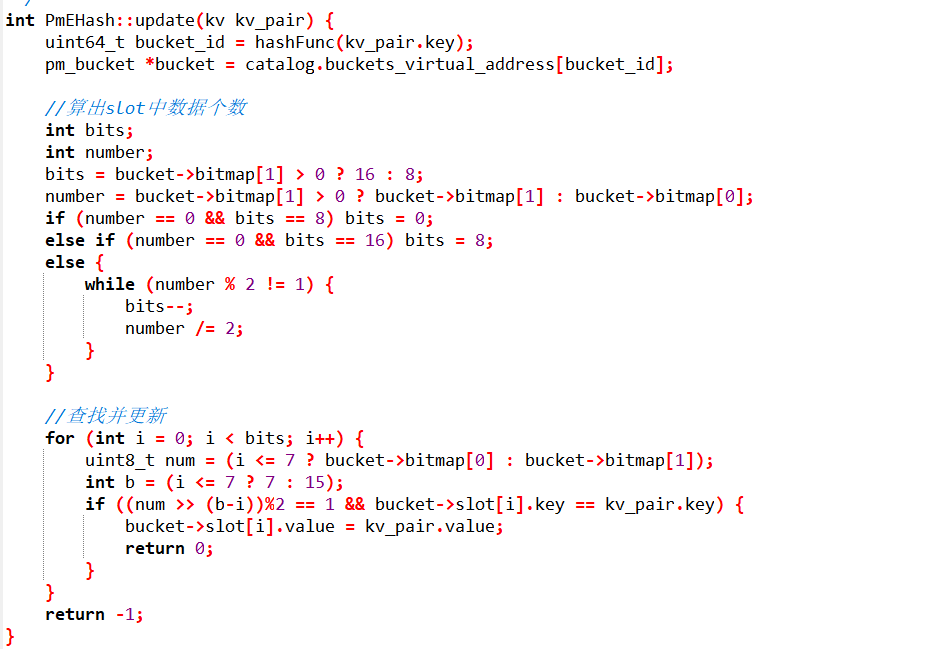
insert函数：先用search函数查询，如果找到了就返回-1，不进行插入操作，如果没有找到就进行插入操作。插入操作先调用getFreeBucket得到对应桶的地址存在bucket，然后调用getFreeKvSlot得到桶中空闲的槽位的地址存在freePlace，进行赋值。插入完成后用freePlace减去槽位的首地址得到该槽的下标，将对应槽位的位图置1。



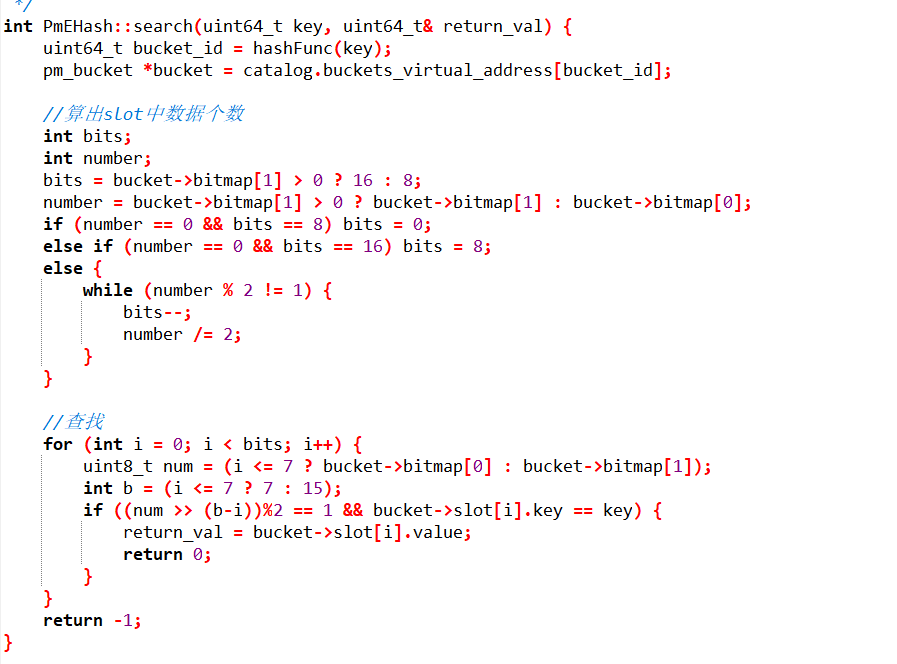
remove函数：先调用hash函数得到桶号存在bucket\_id中，通过catalog的buckets\_virtual\_address来找到桶的地址存在bucket中。通过while循环得到位图中最后一位为1的位在第几位存在bits中，如果位图为0，则bits为0或8。在桶中slot[0]到slot[bits-1]找到对应位位图为1且与目标key相等的槽位下标，存在position中，没有找到position为-1。若position为-1则返回-1，romove失败，否则将对应位的位图置0，如果位图置0后桶里没有元素，就调用mergebucket回收桶。



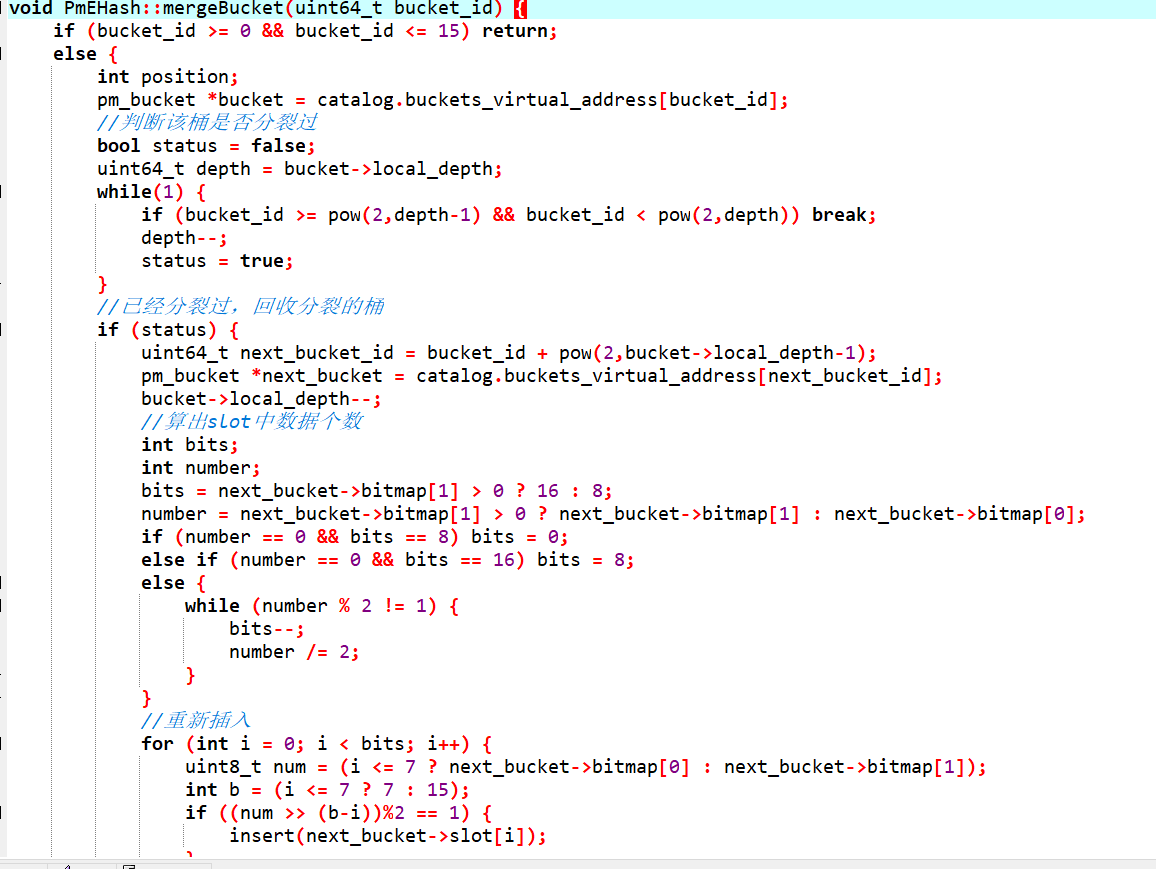
update函数：先调用hash函数得到桶号存在bucket\_id中，通过catalog的buckets\_virtual\_address来找到桶的地址存在bucket中。通过while循环得到位图中最后一位为1的位在第几位存在bits中，如果位图为0，则bits为0或8。在桶中slot[0]到slot[bits-1]找到对应位位图为1且与目标key相等的槽，将对应的vlaue更新，然后返回0，update操作成功，否则返回-1，update操作失败。



search函数：先调用hash函数得到桶号存在bucket\_id中，通过catalog的buckets\_virtual\_address来找到桶的地址存在bucket中。通过while循环得到位图中最后一位为1的位在第几位存在bits中，如果位图为0，则bits为0或8。在桶中slot[0]到slot[bits-1]找到对应位位图为1且与目标key相等的槽，将对应的value值取出来存在return\_val中。

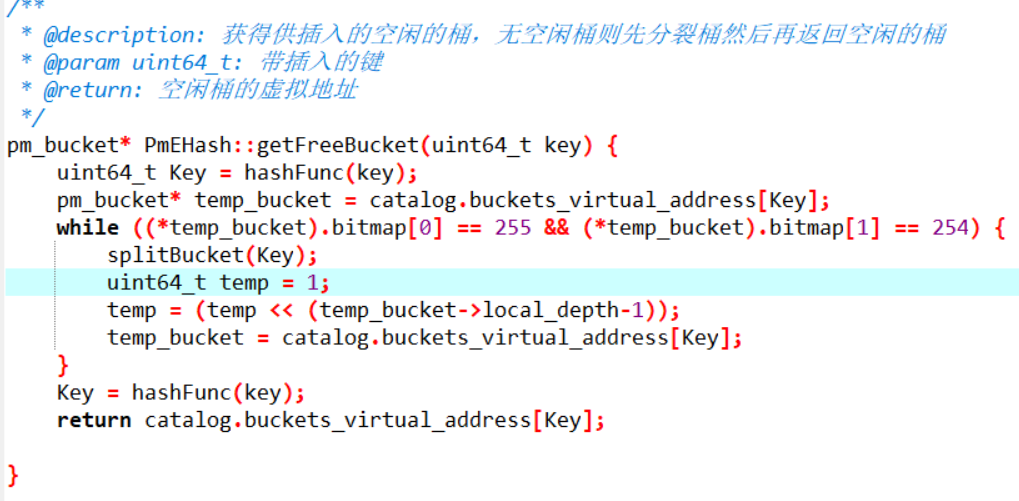


mergeBucket函数：如果要回收的桶为0-15则返回-1，不进行回收。通过catalog的buckets\_virtual\_address来找到桶的地址存在bucket中。用while循环来检验该桶是否进行过分裂，如果bucket\_id在pow(2,depth-1)和pow(2,depth)之间那么该桶没有进行过分裂，否则进行过分裂。如果该桶分裂过，算出分裂的桶号存在next\_bucket\_id中，通过catalog的buckets\_virtual\_address来找到桶的地址存在next\_bucket中，将bucket桶的局部深度减1，将next\_bucket中的数据重新插入，将next\_bucket的位图置0，将next\_bucket\_id对应的 buckets\_pm\_address中的fileId置0，将next\_bucket\_id对应的buckets\_virtual\_address中置为NULL，算出next\_bucket在数据页中桶位置的下标存在position中。如果没有分裂过，算出分裂出bucket的桶的桶号，通过catalog的buckets\_virtual\_address来找到桶的地址存在last\_bucket中，比较bucket与last\_bucket的局部深度，如果相同，将last\_bucket的局部深度减1，将last\_bucket\_id对应的 buckets\_pm\_address中的fileId置0，将last\_bucket\_id对应的buckets\_virtual\_address中置为NULL，算出last\_bucket在数据页中桶位置的下标存在position中，如果不同，则将position置为-1。如果position不为-1则进行回收桶的操作，将position对应的桶加入到free\_list的第一个，将datapage中对应桶的位图置0.



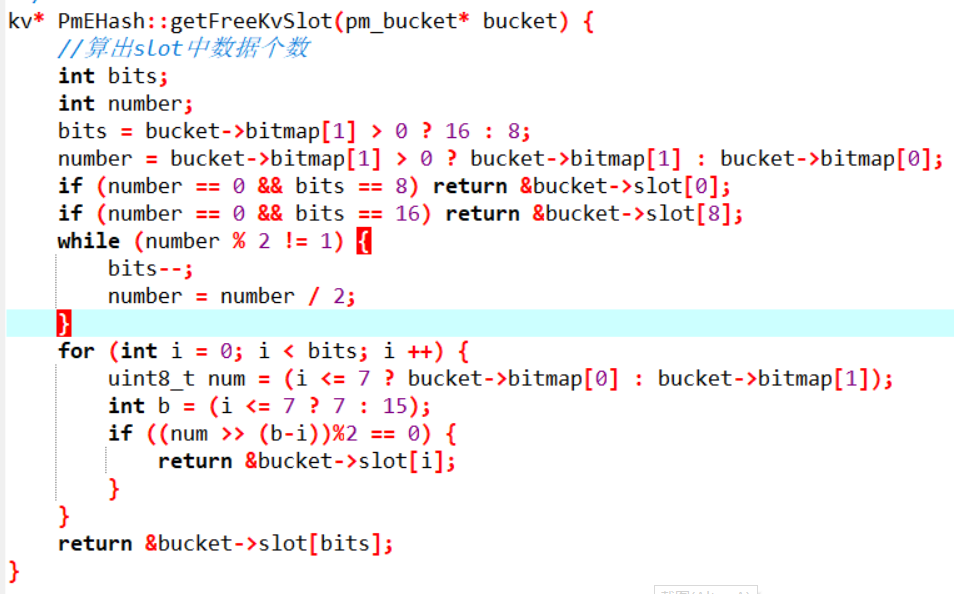
getFreeBucket：找到空闲的桶。

通过哈希函数找到数据应当插入的桶，若该桶已满，则调用splitBucket函数，对桶进行分裂，并再次校对该桶是否满，循环校对直到该桶有空闲空间。返回空闲桶的虚拟地址。



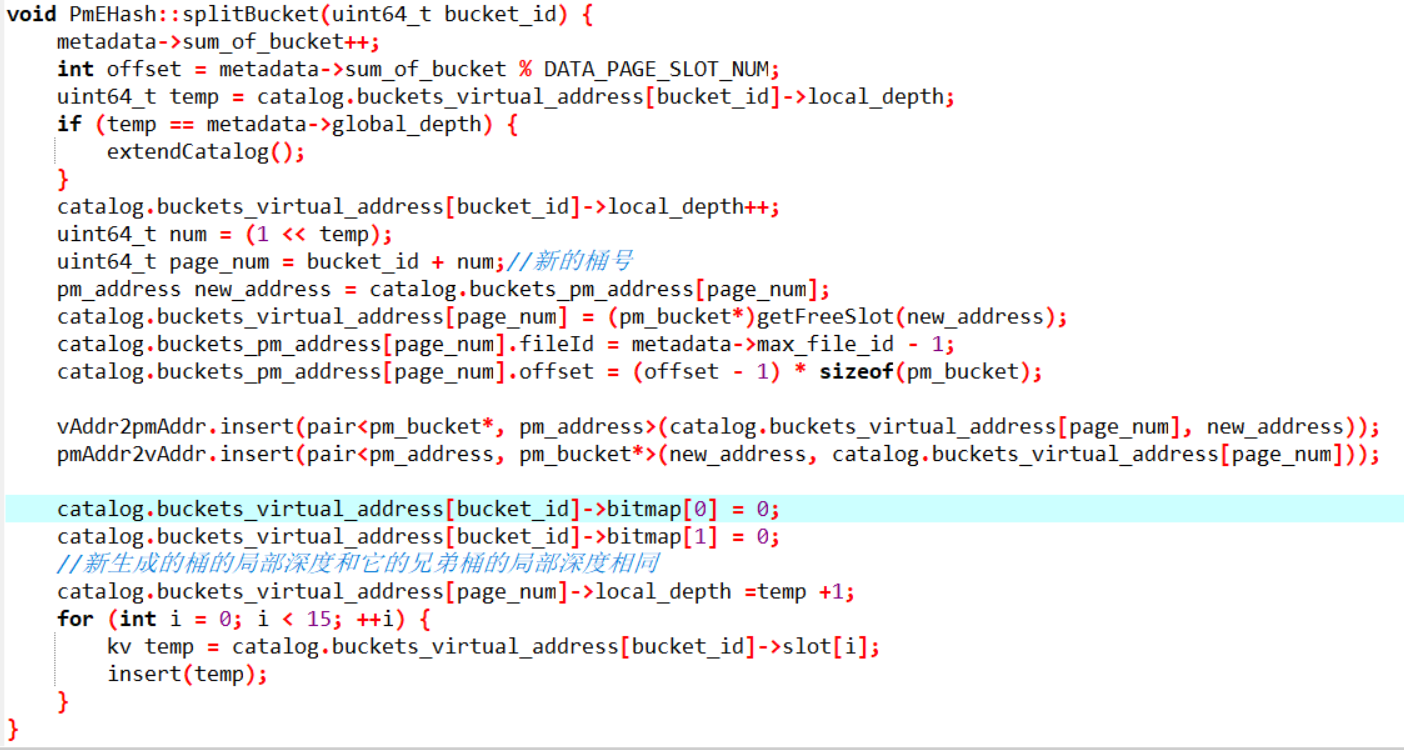
getFreeKvSlot（）函数：找到桶里面空闲的位置。

根据bucket中的位图来计算第一个空闲位置在哪。



splitBucket（）函数：对桶进行分裂操作。

需要注意的是当该桶的局部深度与全局深度相同的时候，要进行目录的倍增操作，当调用了该函数后，新分裂的桶与原来的桶的局部深度都要++，将新生成桶的物理地址与虚拟地址记录在目录中。最后，要将原先数据满的桶的数据进行重新分配。



extendCatalog（）函数：对目录进行倍增操作，先对metadata->catalog\_size的值乘以2，申请新的数组空间来存储物理地址和虚拟地址，再将旧的目录数据拷贝过去，最后删除旧目录并且更新元数据中的全局深度和目录大小。

getFreeSlot（）函数：若free\_list为空则申请新的数据页，否则从free\_list中取出空闲桶。

allocNewPage（）函数：申请新的数据页，将新的空闲的桶放入free\_list中并初始化新桶的位图。

Recover（）函数

Recover函数主要实现从文件中读取原哈希表的过程，通过pmem\_map\_file打开文件，并将文件中的数据读出到现有的哈希表中，实现初始化。主要有三个部分：

1. 读出catalog中的数据。里面主要有buckets\_pm\_address中的fileid和offset，用于从数据页中找到对应的数据。
2. 读取metadata中的数据，主要用于记录catalog的大小，下一页号等信息。
3. 从每个页中读取相应的信息，并且生成新页，将新页push到datapage（vector）中，用于在内存中存储。
4. 完成后进行mapAllPage函数，将目录与桶相连

mapAllPage函数

主要进行目录与页以及页中的桶的匹配。通过上一步中找到的fileid和offset进行寻址，通过基址+偏移量的方法进行寻址，然后将目录与桶对应。

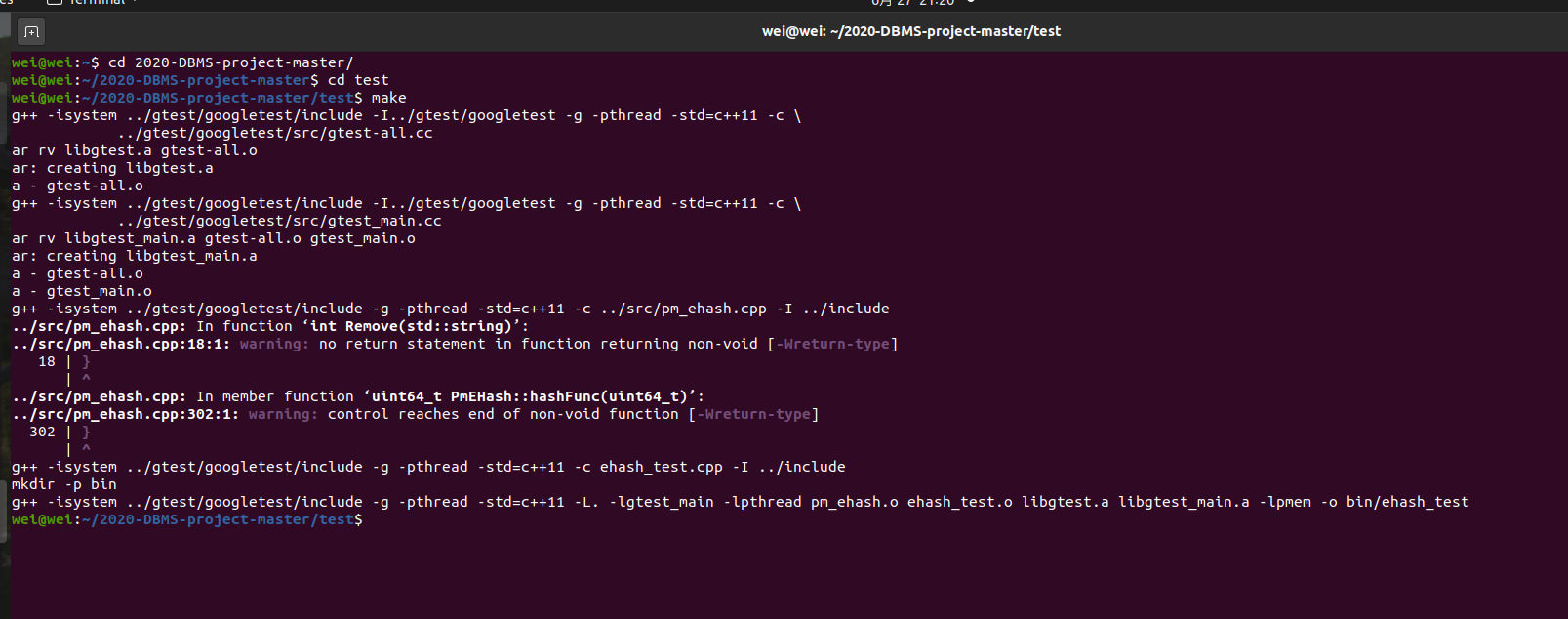
4.加分项实现说明

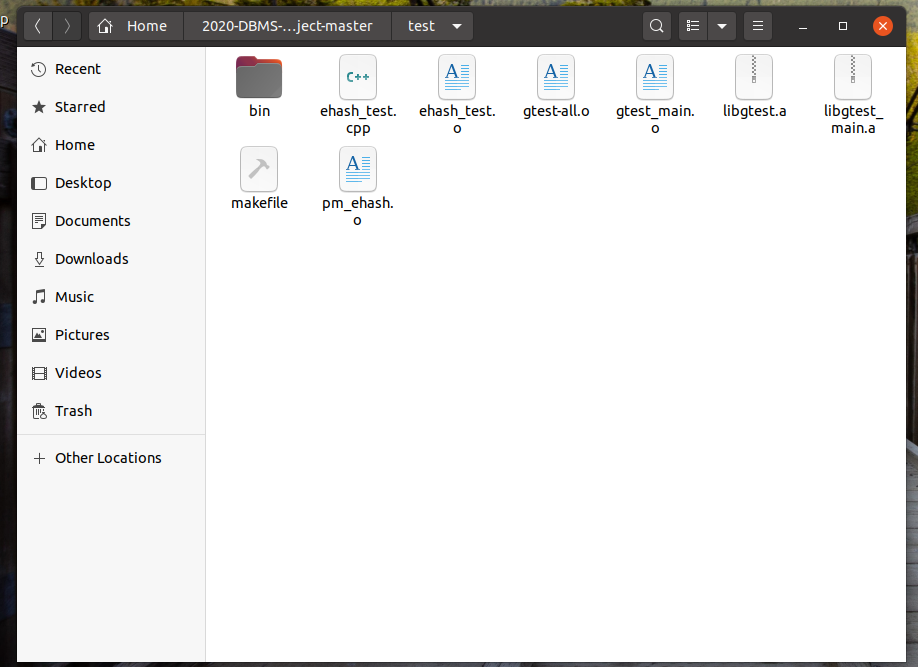
加分项没有实现

5.实验效果截图，gtest以及ycsb测试

gtest测试：

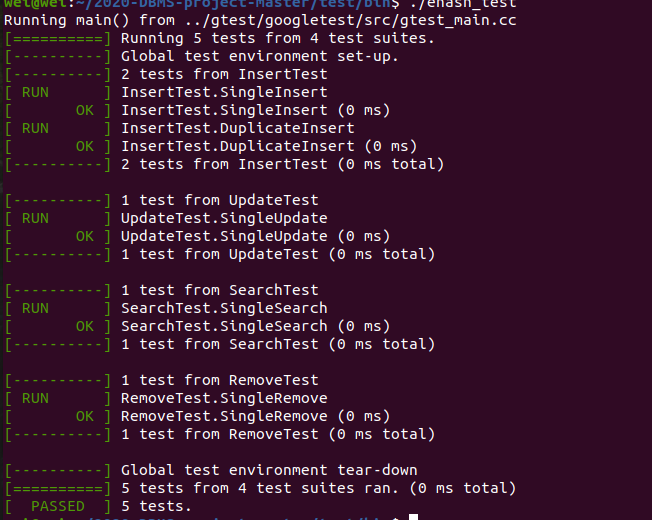
将写好的文件放入src目录下，然后在test目录下运行make指令，得到下面的结果



在test文件下生成了许多连接之后的文件，出现了bin目录

在bin目录下运行ehash\_test文件 （./ehash\_test）进行测试

得到测试结果：



说明gtest通过。

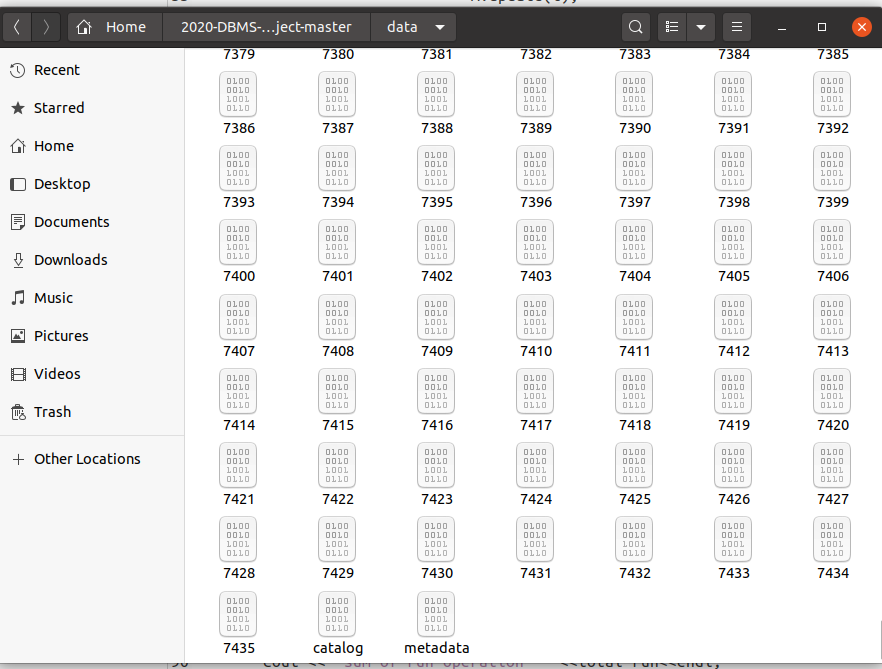
Ycsb测试：

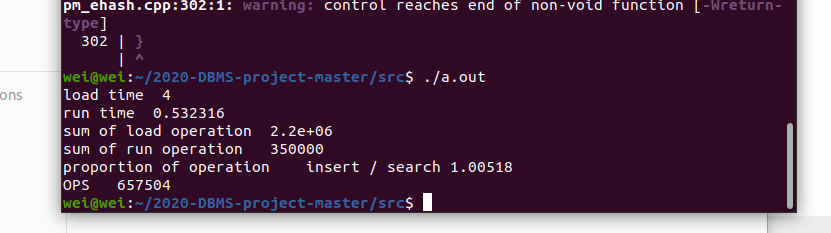
之后进行ycsb测试，使用的实验测例为220w-rw-50-50-run.txt

进行多次实验后得到下面的结果，得到load时间为4s,run时间为0.5s，

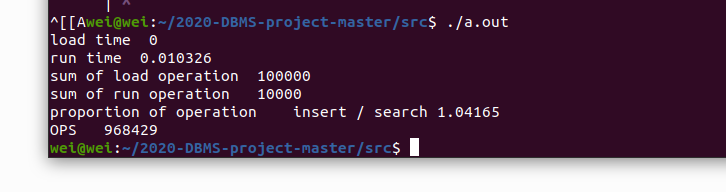
总load数据2200000个，进行run操作350000次，插入与查找的比例接近为1:1，ops为657504；

查看数据页：



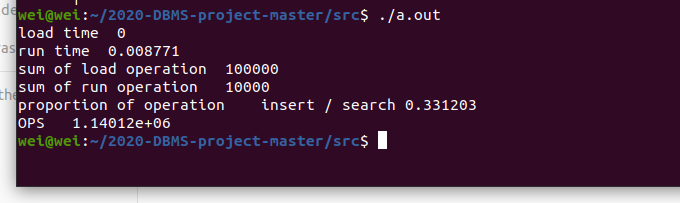


用10 w-rw-50-50-run.txt进行测试，得到下面的结果：



插入和搜索的比例大概是1:1

用10 w-rw-75-25-run.txt进行测试，得到下面的结果：



## 实验心得：

本次实验中，我们初步实现了一个简单的数据库，通过动态1的方法对数据进行插入、查找、删除、更新等操作。实验中我们遇到了许多困难：

1、对linux环境不熟悉，在debug过程中多次遇到段错误等问题，多次调试后才得到结果，消耗了比较多的时间，在make过程中也有一些疑惑，通过在网上查找多次后解决。

2、在文件存储的过程中遇到了许多问题，比如在动态分配空间大小的过程中遇到了许多问题，导致数据写出总是出现错误，后来通过固定分配空间解决。

通过本次实现，我们对动态hash表的过程，目录与数据页的对应，linux下的文件操作有了更多的认识，熟悉了动态哈希的过程，明白了物理地址和虚拟地址的相互对应。